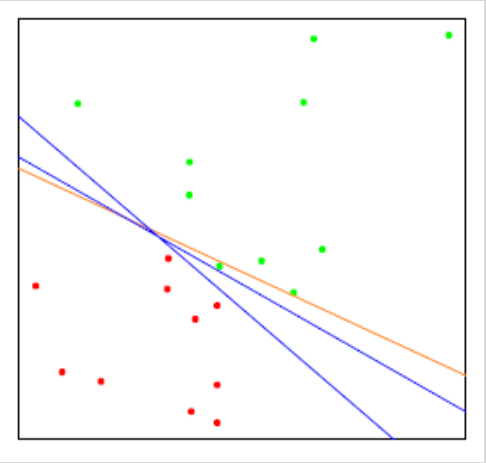


DEEP-EST – Tutorial

Machine Learning and Modular Supercomputing

Prof. Dr. – Ing. Morris Riedel

Adjunct Associated Professor

School of Engineering and Natural Sciences, University of Iceland

Research Group Leader, Juelich Supercomputing Centre, Germany

LECTURE 2

Parallel and Scalable Classification using Artificial Neural Networks

January 21th, 2019, HiPEAC Conference

Conference Centre, Valencia, Spain

HELMHOLTZ
RESEARCH FOR GRAND CHALLENGES

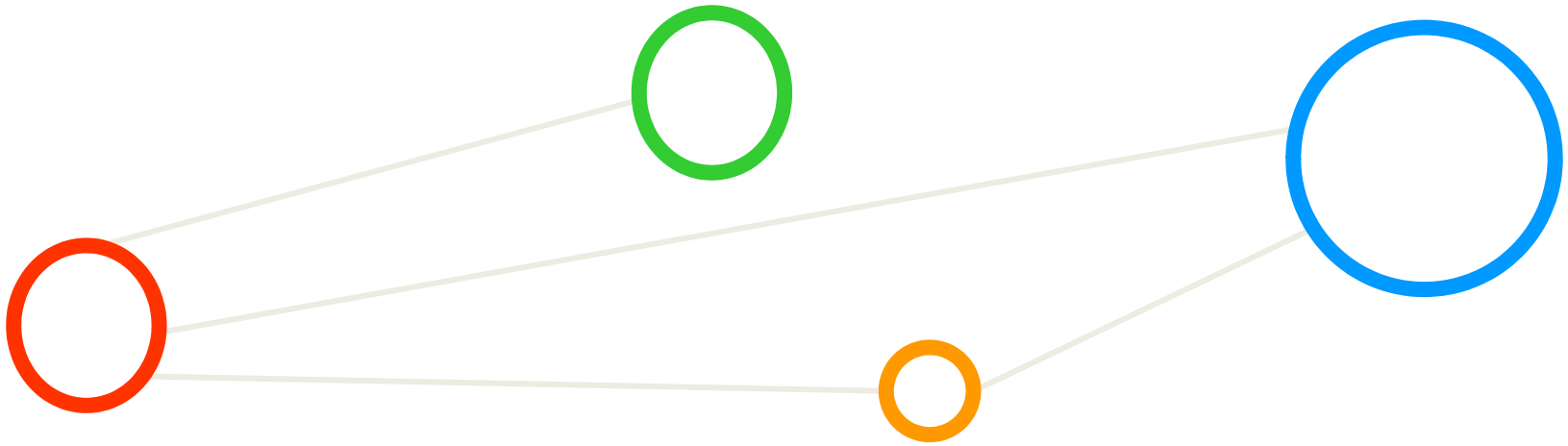
UNIVERSITY OF ICELAND
SCHOOL OF ENGINEERING AND NATURAL SCIENCES
FACULTY OF INDUSTRIAL ENGINEERING, MECHANICAL ENGINEERING AND COMPUTER SCIENCE

JÜLICH Forschungszentrum
JÜLICH SUPERCOMPUTING CENTRE

HiPEAC

DEEP Projects

Outline

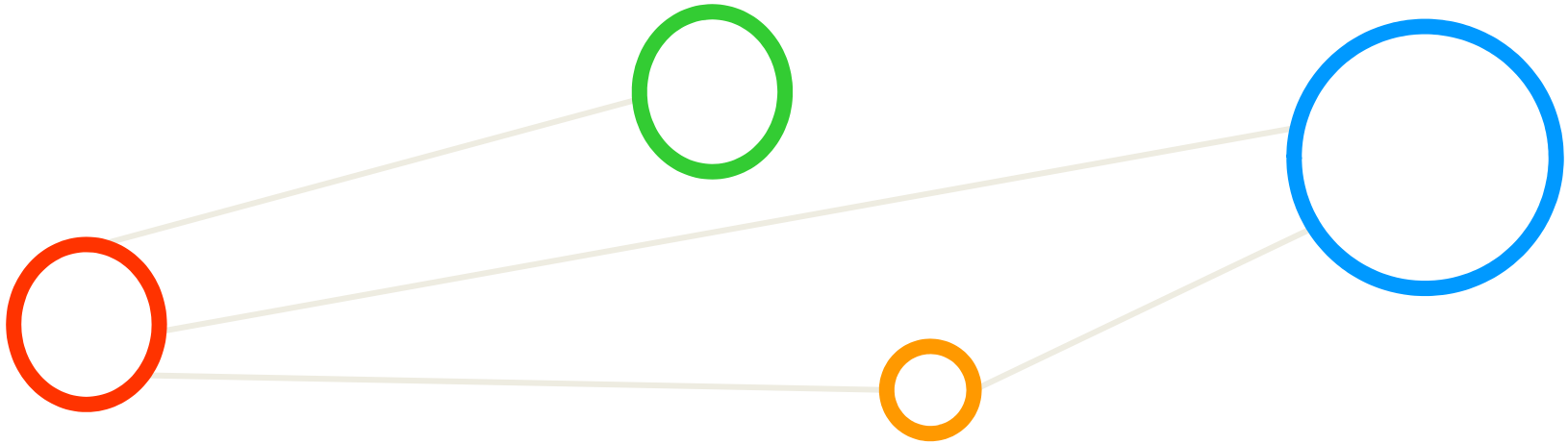


Outline

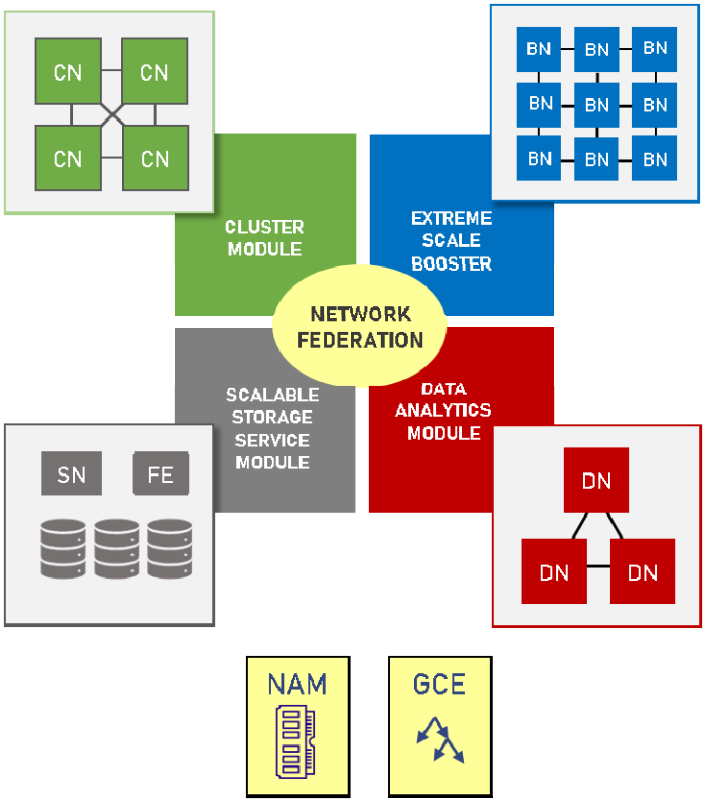
- Introduction & Supervised Classification
  - DEEP-EST & Modular Supercomputing with Accelerators
  - Software Tools Keras & TensorFlow
  - Simple Learning Example with Linear Perceptron Model
  - Selected Mathematical Building Blocks
  - Supervised Learning

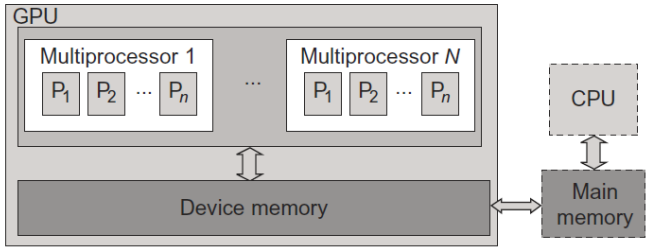
- Artificial Neural Networks (ANNs)
  - MNIST Application Example &Perceptron Limits
  - Artificial Neural Networks (ANNs) & Layers
  - Backpropagation Learning Algorithm
  - Regularization & Validation to Combat Overfitting
  - Keras Examples for MNIST

Introduction & Supervised Classification

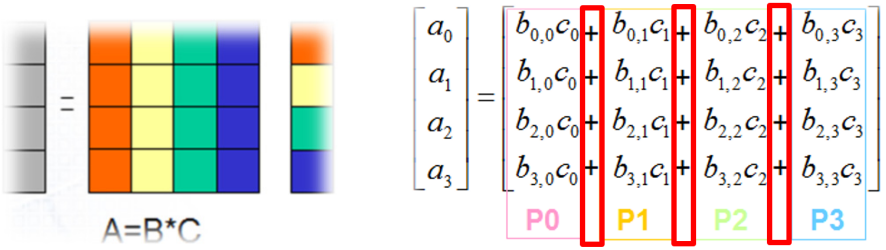


Modular Supercomputing with DEEP-EST & Accelerators





[27] Distributed & Cloud Computing Book



[2] DEEP Projects Web Page

- **Many machine & deep learning algorithms and models require many matrix-matrix or matrix-vector calculations that can run nicely in parallel**
- **Accelerators & General Purpose Graphical Processing Units (GPGPUs) – short GPU fit nicely this demand with their many-core architecture**

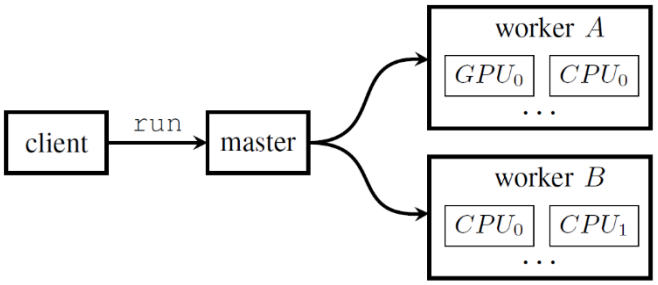
Software: Keras & Tensorflow Backend – GPU Support

- Keras is a high-level deep learning library implemented in Python that works on top of existing other rather low-level deep learning frameworks like Tensorflow, CNTK, or Theano
- The key idea behind the Keras tool is to enable faster experimentation with deep networks
- Created deep learning models run seamlessly on CPU and GPU via low-level frameworks

Keras *[24] Keras Python Deep Learning Library*

- Tensorflow is an open source library for deep learning models using a flow graph approach
- Tensorflow nodes model mathematical operations and graph edges between the nodes are so-called tensors (also known as multi-dimensional arrays)
- The Tensorflow tool supports the use of CPUs and GPUs (much more faster than CPUs)
- Tensorflow work with the high-level deep learning tool Keras in order to create models fast

[25] Tensorflow Deep Learning Framework



[26] A Tour of Tensorflow

TensorFlow

Software & Libraries – Watch for Dependencies

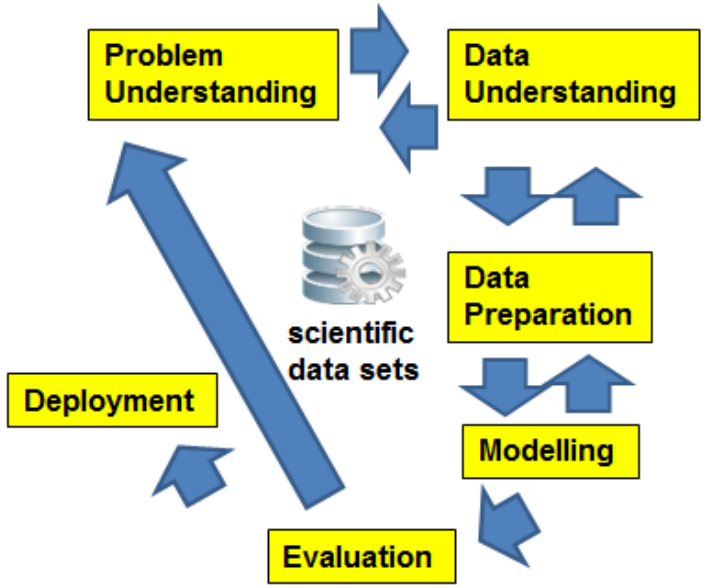
```
[riedel1@juron1-adm ~]$ module list

Currently Loaded Modules:
  1) lsf/10.1          (S)   8) openmpi/2.0.2-gcc_5.4.0  15) protobuf-python/3.5.1
  2) python/2.7.14           9) hdf5/1.8.18-parallel     16) tensorflow/1.7.0-gcc_5.4.0-cuda_9.1.85
  3) cython/0.26.1          10) h5py/2.8.0               17) cuDNN/7.1
  4) OpenBLAS/v0.2.19       11) scipy/1.0.1              18) cuda/9.1.85
  5) protobuf/3.3.0         12) keras/2.2.4              19) jinja2/2.10
  6) pyyaml/3.12            13) numpy/1.14.2             20) pyzmq/17.0.0b3
  7) gcc/5.4.0              14) matplotlib/2.1.0         21) jupyter/1.0.0

  Where:
   S:  Module is Sticky, requires --force to unload or purge
```

Systematic Process to Support Learning From Data

- Systematic data analysis guided by a ‘standard process‘
  - Cross-Industry Standard Process for Data Mining (CRISP-DM)

- **A data mining project is guided by these six phases:**
  (1) Problem Understanding;
  (2) Data Understanding;
  (3) Data Preparation;
  (4) Modeling;
  (5) Evaluation;
  (6) Deployment

(machine learning takes place)



- Lessons Learned from Practice
  - Go back and forth between the different six phases

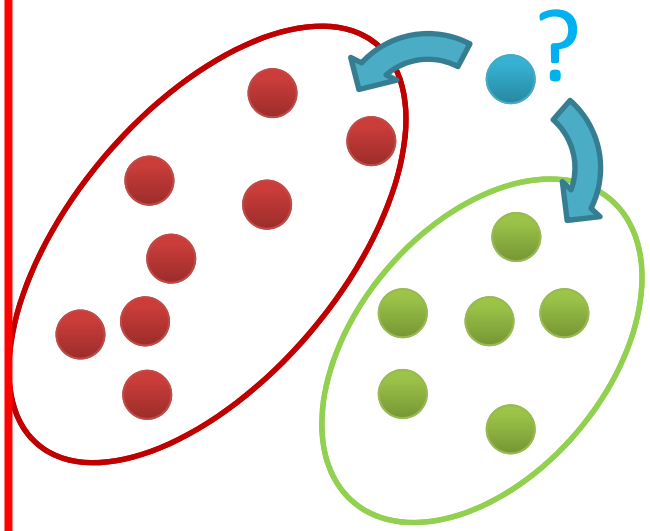
[10] C. Shearer, CRISP-DM model, Journal Data Warehousing, 5:13

➢ **A more detailed description of all six CRISP-DM phases is in the Appendix A of the slideset**

Machine Learning Methods Overview

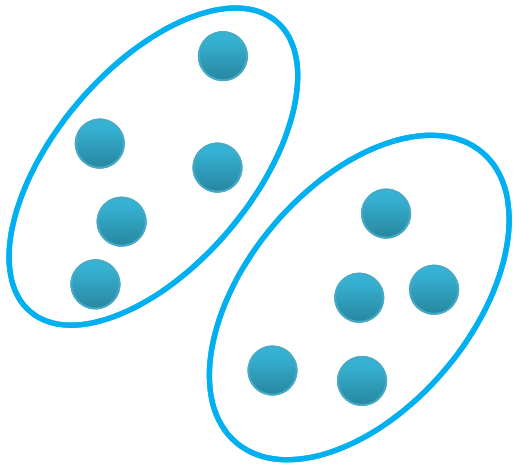
- **Machine learning methods can be roughly categorized in classification, clustering, or regression augmented with various techniques for data exploration, selection, or reduction**

Classification



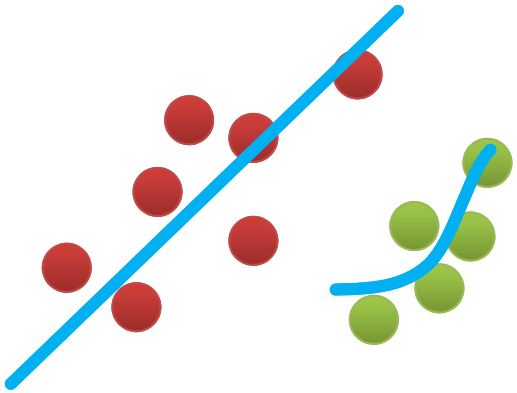
- Groups of data exist
- New data classified to existing groups

Clustering



- No groups of data exist
- Create groups from data close to each other

Regression



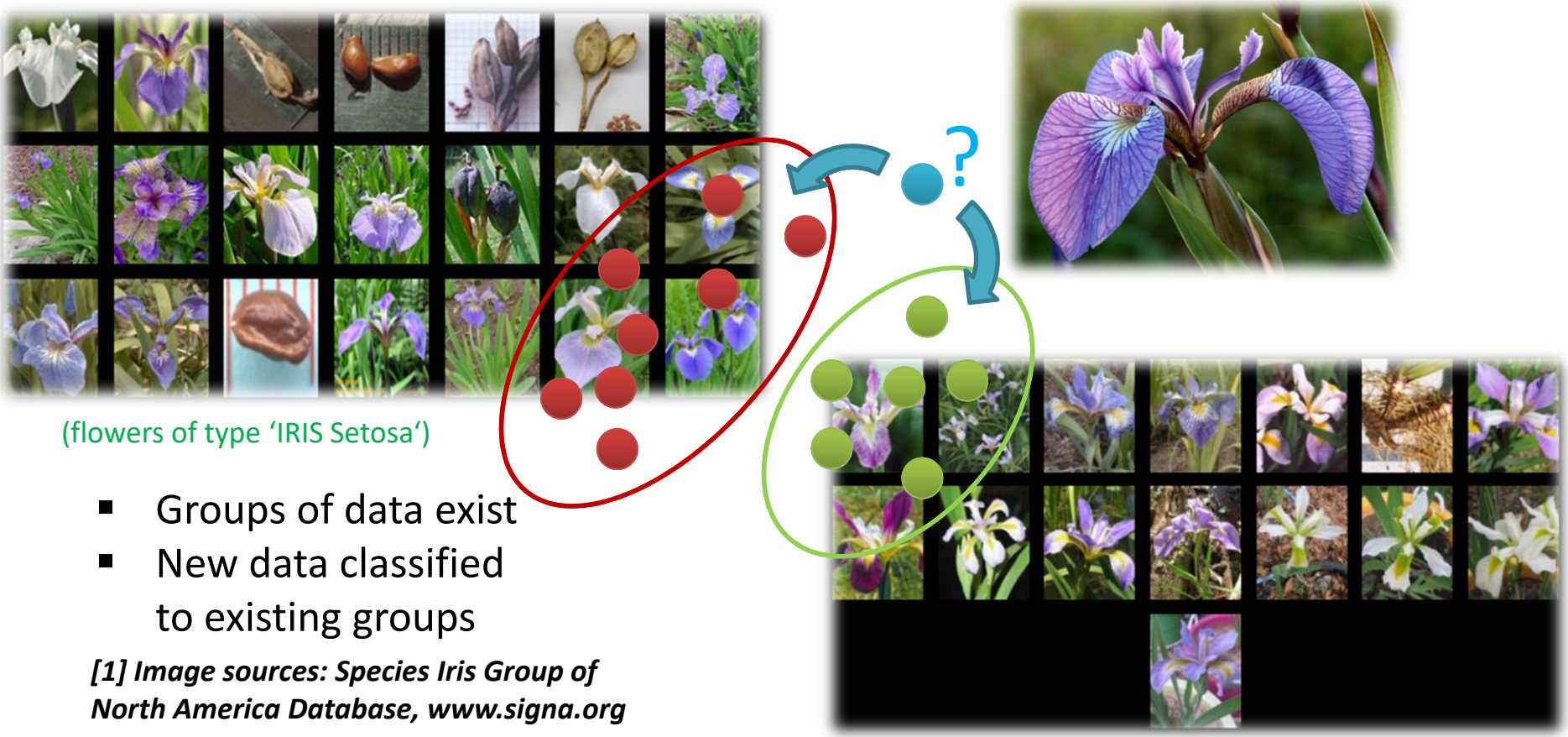
- Identify a line with a certain slope describing the data

➢ **This short tutorial focusses on classification algorithms and models with practical examples**

Simple Application Example: Classification of a Flower

(1) Problem Understanding Phase

(what type of flower is this?)



(flowers of type 'IRIS Setosa')

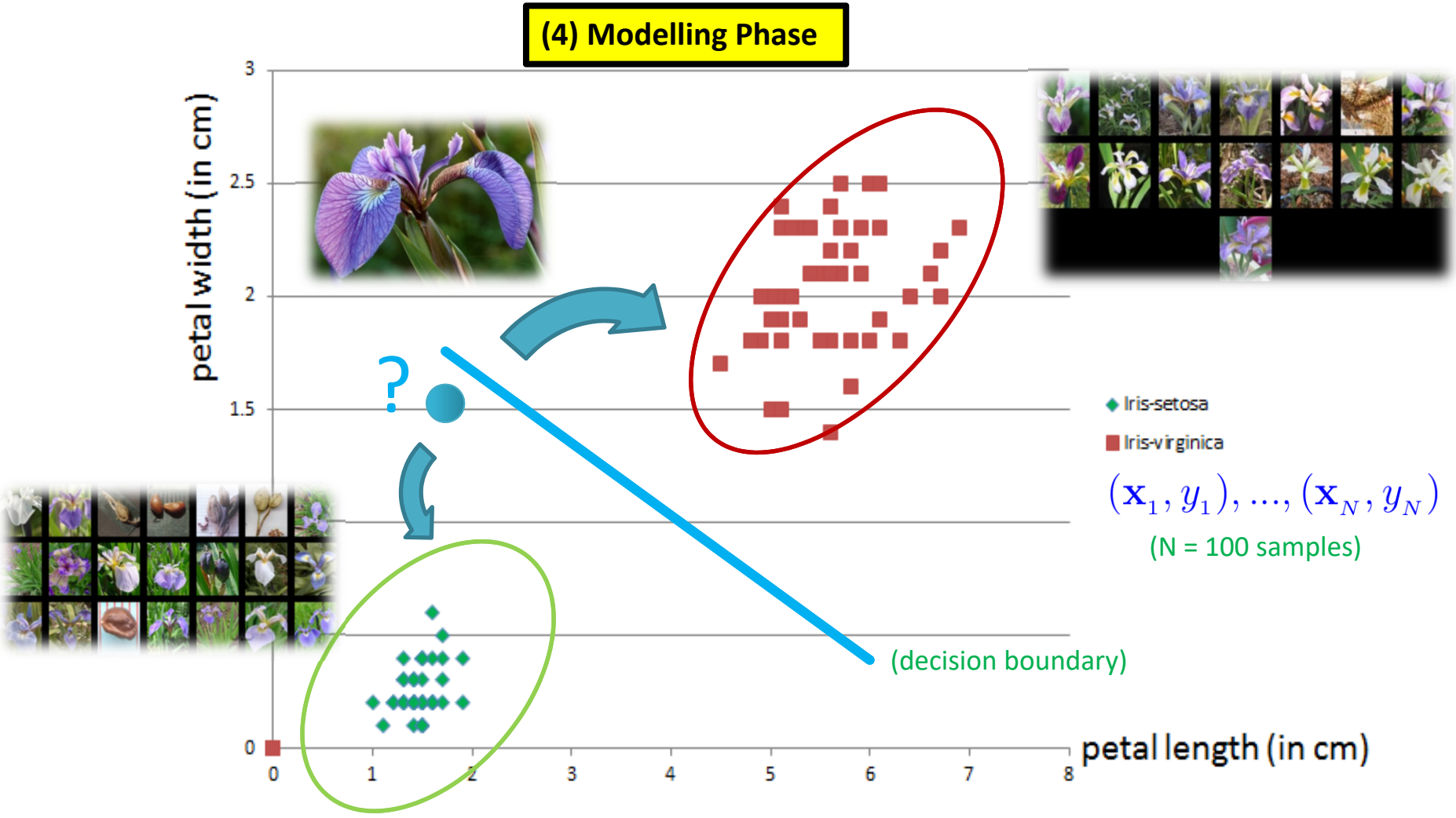
- Groups of data exist
- New data classified to existing groups

[1] Image sources: Species Iris Group of North America Database, www.signa.org

(flowers of type 'IRIS Virginica')

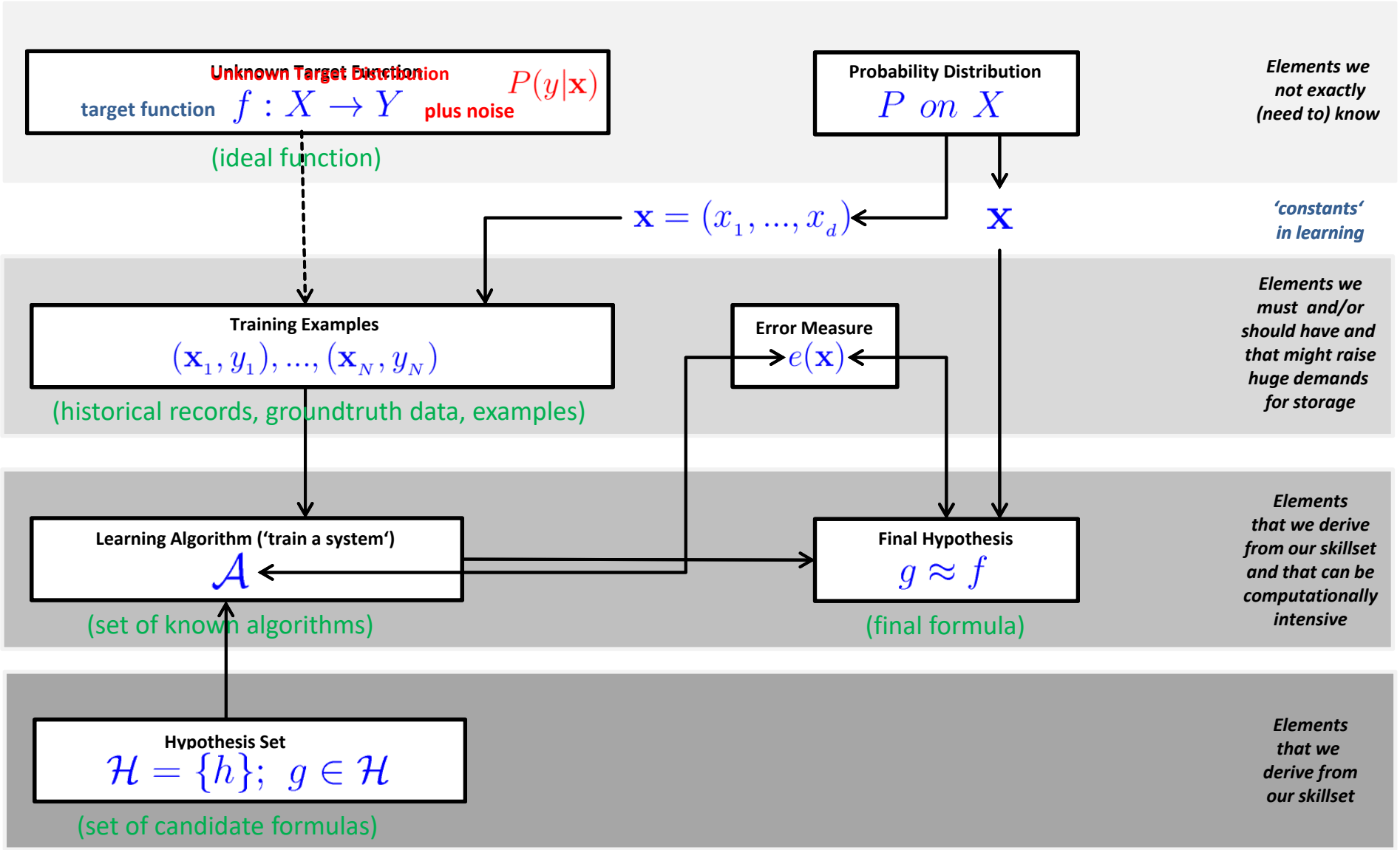
➢ **Appendix E offers a simple introduction to machine learning with the Perceptron Learning Model**

Predicting Task: Obtain Class of a new Flower ‘Data Point‘



[1] Image sources: Species Iris Group of North America Database, www.signa.org

Supervised Learning – Overview & Summary



Different Models – Understanding the Hypothesis Set

Hypothesis Set

$$\mathcal{H} = \{h\};\ g \in \mathcal{H}$$

$$\mathcal{H} = \{h_1, \ldots, h_m\};$$

(all candidate functions derived from models and their parameters)

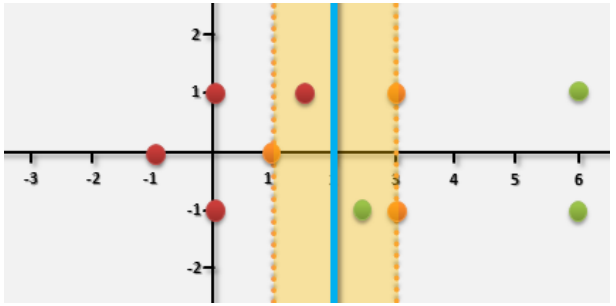
- **Already a change in model paramters of h_1, …, h_m means a completey different model**

‘select one function‘ that best approximates

Final Hypothesis

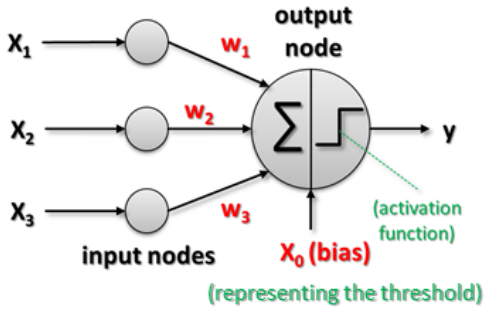
$$g \approx f$$

h_1



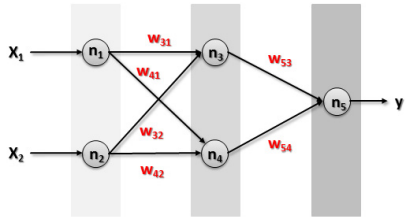
(e.g. support vector machine model)

h_2



(e.g. linear perceptron model)

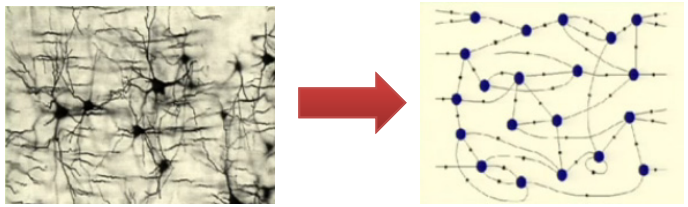
h_m

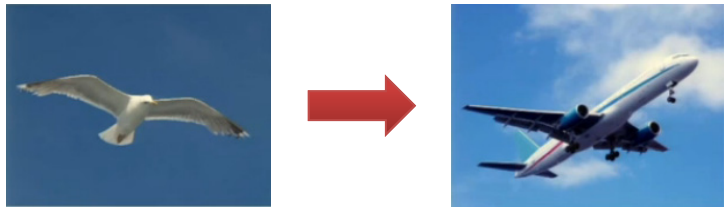


(e.g. artificial neural network model)

Learning Models derived from Biological Inspiration

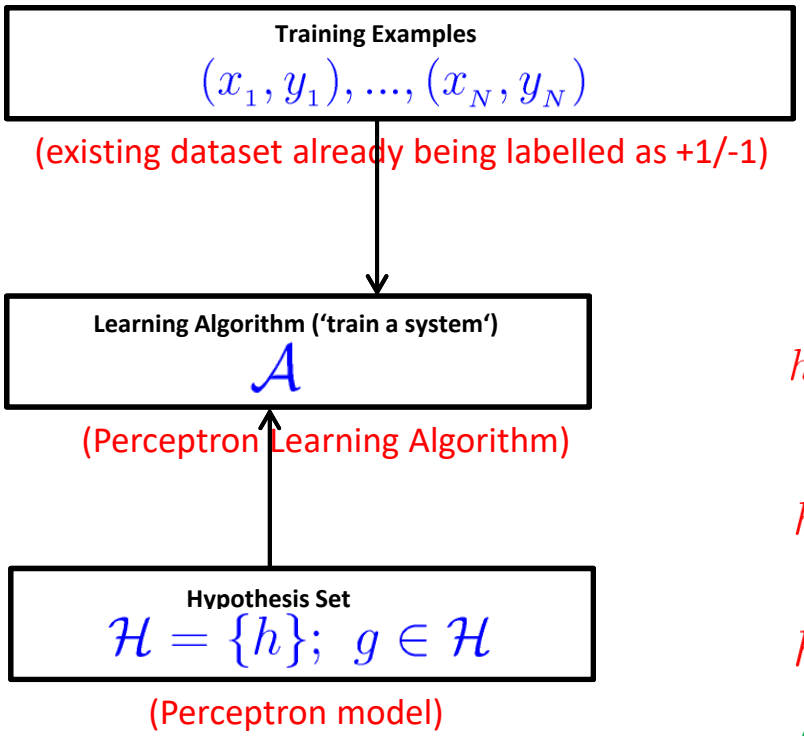
- Biological Inspiration
  - Humans learn (a biological function) → machines can learn
  - Means we are interested in 'replicating' the 'biological function'
- Approach: Replicating the 'biological structure'
  - Neurons connected to synapses (large number)
  - Action of neurons depends on 'stimula of different synapses'
  - Synapses have 'weights'
  - Principle: neurons are in the following like a 'single perceptron'
  - Neural network: put together a 'bunch of perceptrons' in layers
  - Deep learning network: create many layers with 'smart functionalites'

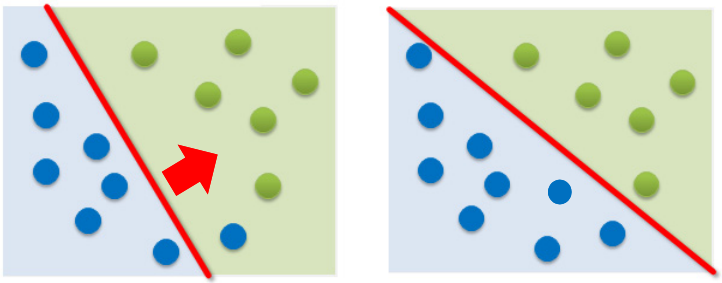




Perceptron Learning Algorithm using Perceptron Model

- When: If we believe there is a linear pattern to be detected
  - Assumption: Linearly seperable data (lets the algorithm converge)





[8] Rosenblatt, 1958

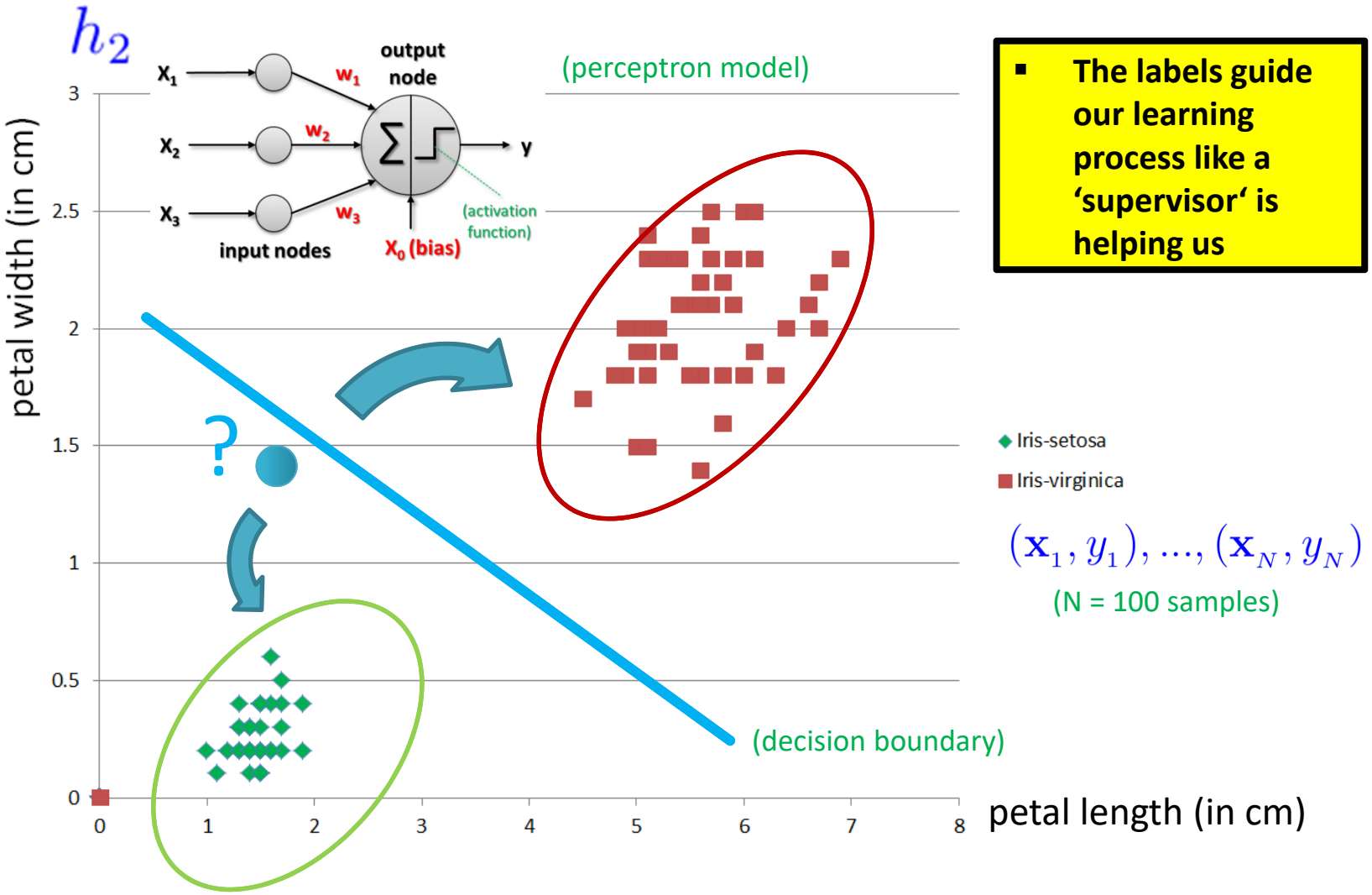
$$h(\mathbf{x}) = sign\left(\left(\sum_{i=0}^{d} w_i x_i\right)\right); x_0 = 1$$

$$h(\mathbf{x}) = sign\left(\left(\sum_{i=1}^{d} w_i x_i\right) + w_0\right); w_0 = -threshold$$

$$h(\mathbf{x}) = sign(\mathbf{w}^T \mathbf{x})$$ **(vector notation, using transpose)**

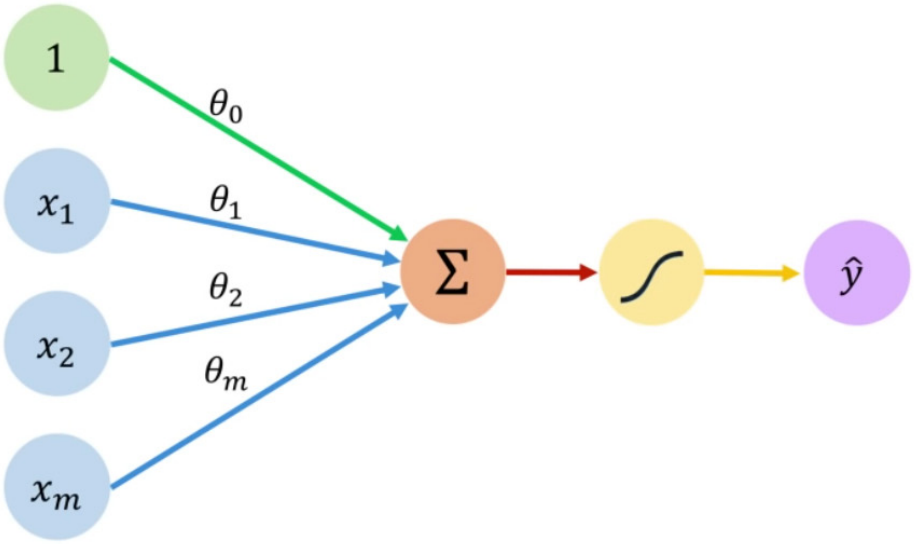
(transpose = reflecting elements along main diagonal)

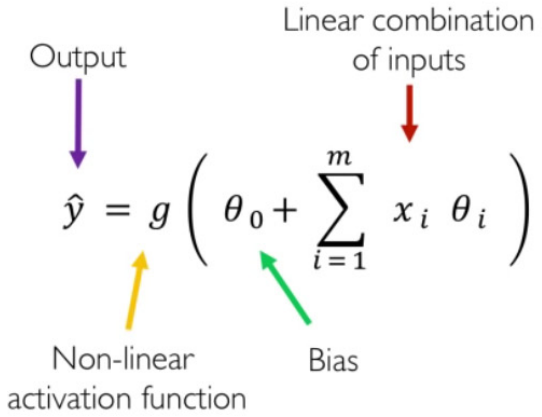
Learning Approaches – Supervised Learning Example



- **The labels guide our learning process like a 'supervisor' is helping us**

Perceptron Learning Model – Summary & Computing

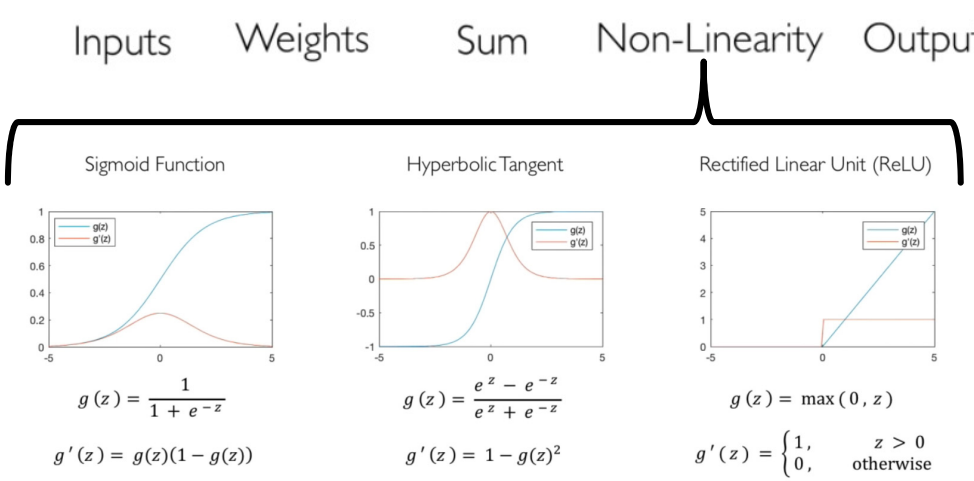


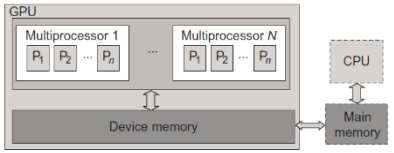


(use linear algebra & dot products)

$$\hat{y} = g(\theta_0 + \boldsymbol{X}^T \boldsymbol{\theta})$$

where: $\boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and $\boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_m \end{bmatrix}$

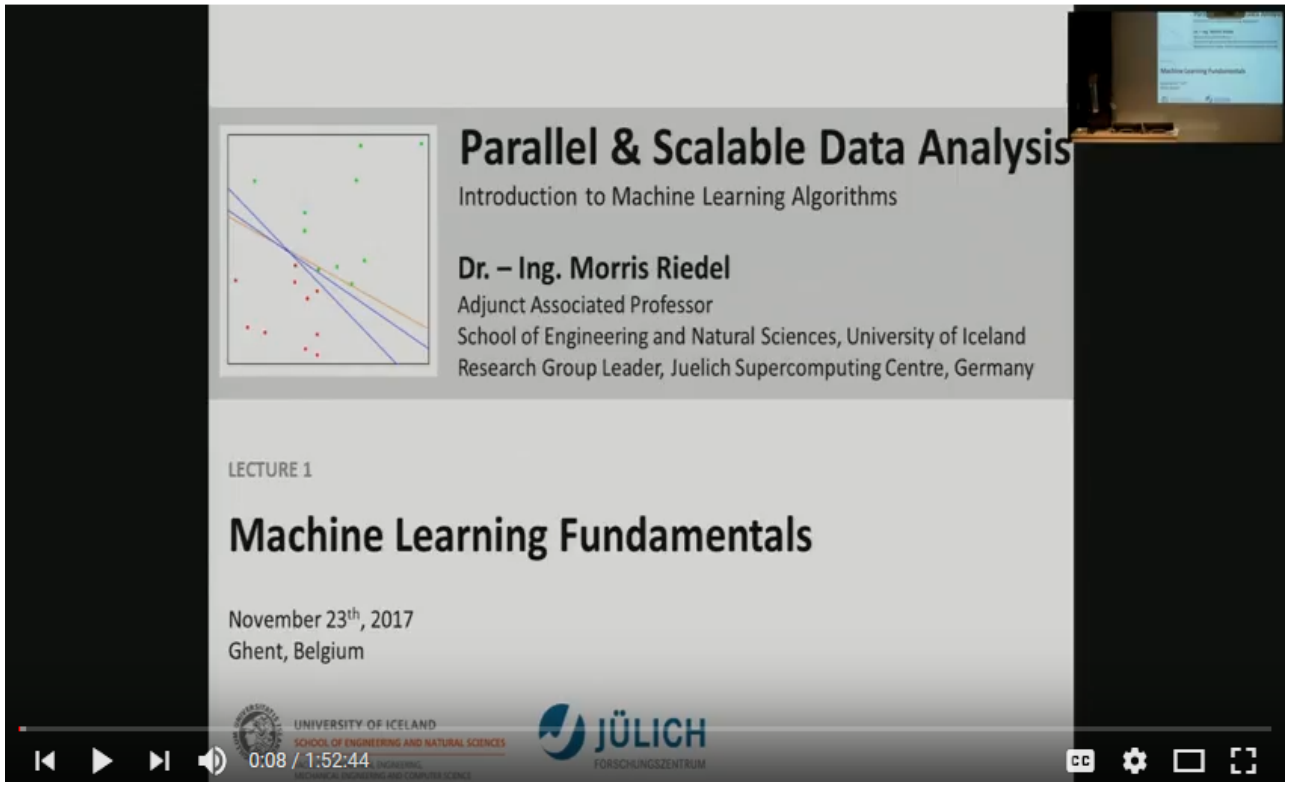




[27] Distributed & Cloud Computing Book

[28] Introduction to Deep Learning

[YouTube Lectures] Machine Learning Fundamentals



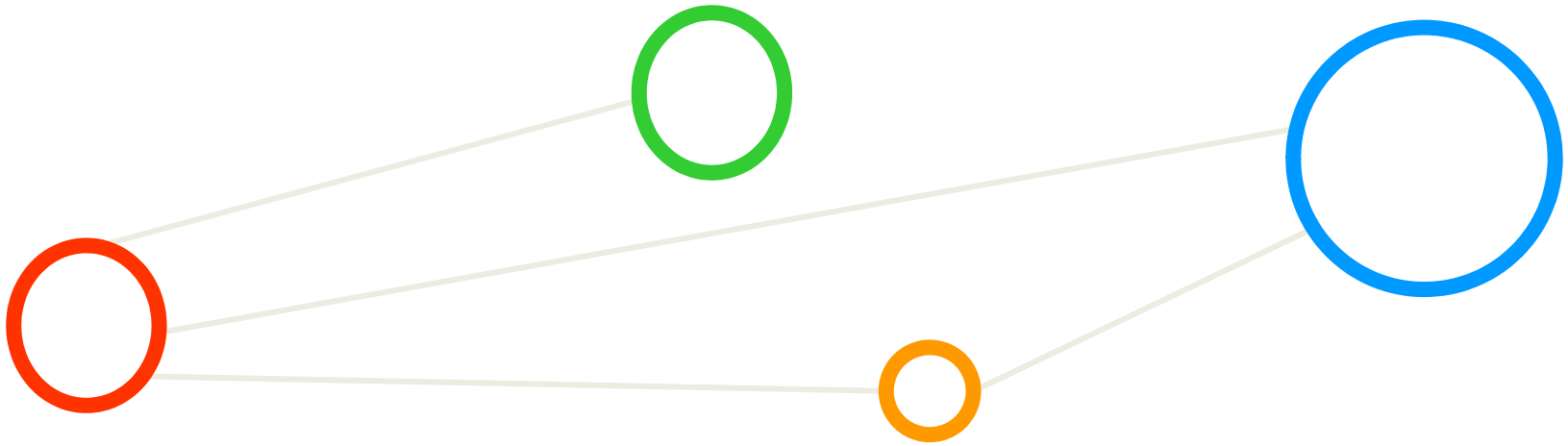
[22] Morris Riedel, 'Introduction to Machine Learning Algorithms', Invited YouTube Lecture, six lectures, University of Ghent, 2017

➢ **Appendix C & D show details of Support Vector Machines (SVMs) and Kernel Methods**

[Video] Remote Sensing

[19] YouTube Video, "What is Remote Sensing?"

Artificial Neural Networks (ANNs)



ANN – MNIST Dataset – Create ANN Blueprint

- Data Preprocessing (i.e. data normalization, reshape, etc.)

1. Define a neural network topology
   - Which layers are required?
   - Think about input layer need to match the data – what data we had?
   - Maybe hidden layers?
   - Think Dense layer – Keras?
   - Think about final Activation as Softmay (cf. Day One) → output probability
2. Compile the model → model representation for Tensorflow et al.
   - Think about what loss function you want to use in your problem?
   - What is your optimizer strategy, e.g. SGD (cf. Day One)
3. Fit the model → the model learning takes place
   - How long you want to train (e.g. NB_EPOCHS)
   - How much samples are involved (e.g. BATCH_SIZE)

ANN – Handwritten Character Recognition MNIST Dataset

- Metadata
  - Subset of a larger dataset from US National Institute of Standards (NIST)
  - Handwritten digits including corresponding labels with values 0 to 9
  - All digits have been size-normalized to 28 * 28 pixels
    and are centered in a fixed-size image for direct processing
  - Not very challenging dataset, but good for experiments / tutorials

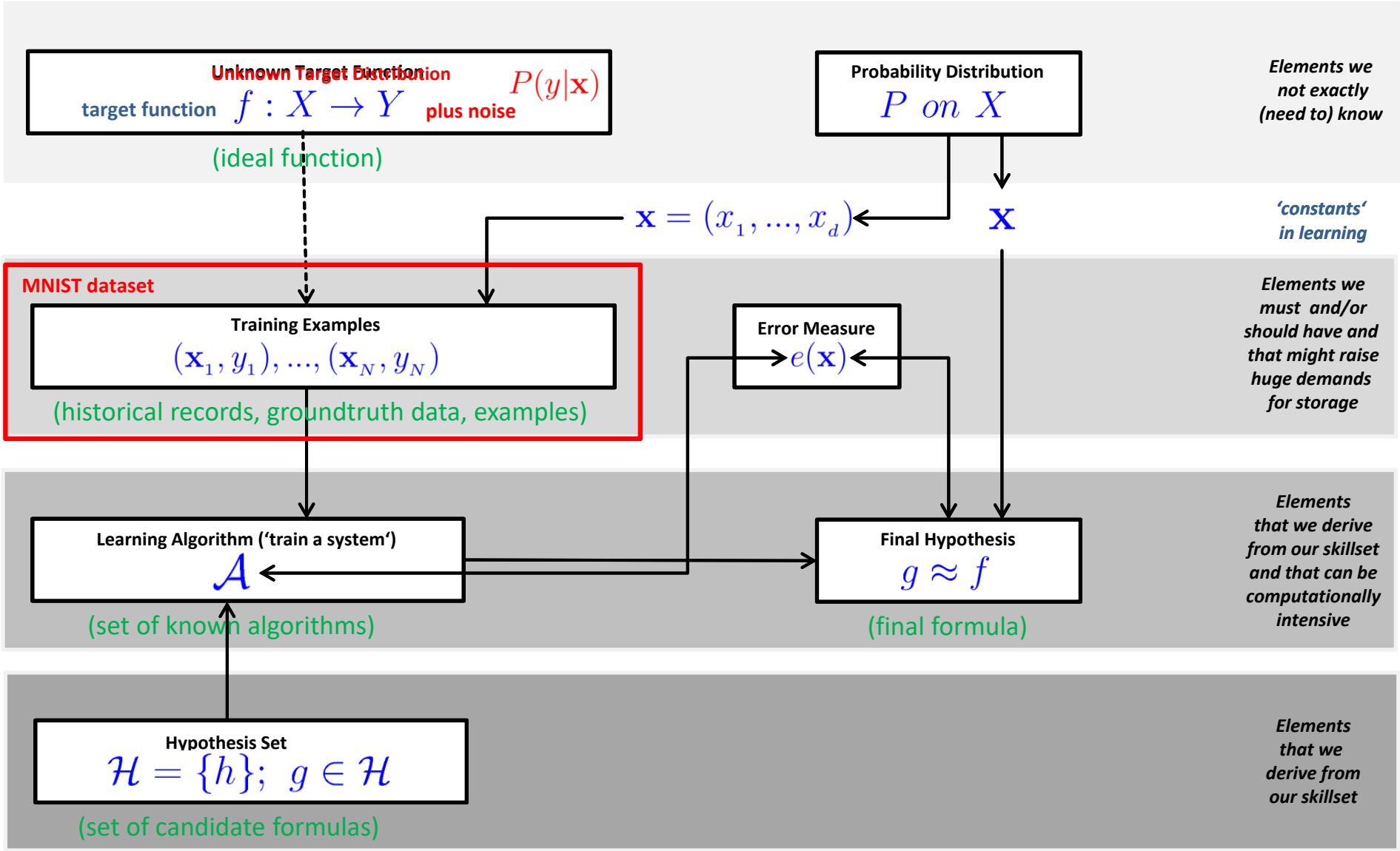
- Dataset Samples
  - Labelled data (10 classes)
  - Two separate files
    for training and test
  - 60000 training samples (~47 MB)
  - 10000 test samples (~7.8 MB)

MNIST Dataset for the Tutorial

- When working with the dataset
  - Dataset is not in any standard image format like jpg, bmp, or gif
  - File format not known to a graphics viewer
  - One needs to write typically a small program to read and work for them
  - Data samples are stored in a simple file format that is designed for storing vectors and multidimensional matrices
  - The pixels of the handwritten digit images are organized row-wise with pixel values ranging from 0 (white background) to 255 (black foreground)
  - Images contain grey levels as a result of an anti-aliasing technique used by the normalization algorithm that generated this dataset.
- Available already for the tutorial
  - Part of the Tensorflow tutorial package and Keras tutorial package

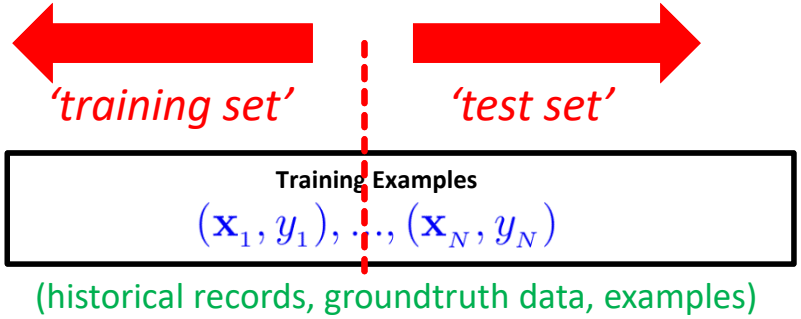
```
# download & unpack MNIST data
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

Supervised Learning – Training Examples



Model Evaluation – Training and Testing Phases

- Different Phases in Learning
  - Training phase is a hypothesis search
  - Testing phase checks if we are on right track (once the hypothesis clear)
- Work on ‘training examples‘
  - Create two disjoint datasets
  - One used for training only (aka training set)
  - Another used for testing only (aka test set)



  - Exact seperation is rule of thumb per use case (e.g. 10 % training, 90% test)
  - Practice: If you get a dataset take immediately test data away (‘throw it into the corner and forget about it during modelling‘)
  - Reasoning: Once we learned from training data it has an ‘optimistic bias‘

MNIST Dataset for the Tutorial

- When working with the dataset
  - Dataset is not in any standard image format like jpg, bmp, or gif
  - One needs to write typically a small program to read and work for them
  - Data samples are stored in a simple file format that is designed for storing vectors and multidimensional matrices (here numpy binary files)
  - The pixels of the handwritten digit images are organized row-wise with pixel values ranging from 0 (white background) to 255 (black foreground)
  - Images contain grey levels as a result of an anti-aliasing technique used by the normalization algorithm that generated this dataset.

```
/homea/hpclab/train001/data/mnist
[train001@jrl09 mnist]$ pwd
/homea/hpclab/train001/data/mnist
[train001@jrl09 mnist]$ ls -al
total 53728
drwxr-xr-x  2 train001 hpclab      512 Jun  6 12:17 .
drwxr-xr-x 10 train001 hpclab      512 Jun  6 12:17 ..
-rw-r-----  1 train001 hpclab  7840080 Jun  6 12:17 x_test.npy
-rw-r-----  1 train001 hpclab 47040080 Jun  6 12:17 x_train.npy
-rw-r-----  1 train001 hpclab    10080 Jun  6 12:17 y_test.npy
-rw-r-----  1 train001 hpclab    60080 Jun  6 12:17 y_train.npy
```

MNIST Dataset – Exploration – One Character Encoding

```
[train001@jrl09 mnist]$ python explore-mnist-training.py
Samples of 28 x 28 pixel matrices reserved for training
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   3   18  18  18  126 136 175 26  166 255 247 127 0   0   0   0
0   0   0   0   0   0   0   0   30  36  94  154 170 253 253 253 253 253 225 172 253 242 195 64  0   0   0   0
0   0   0   0   0   0   0   49  238 253 253 253 253 253 253 253 253 251 93  82  82  56  39  0   0   0   0   0
0   0   0   0   0   0   0   18  219 253 253 253 253 253 198 182 247 241 0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   80  156 107 253 253 205 11  0   43  154 0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   14  1   154 253 90  0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   139 253 190 2   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   11  190 253 70  0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   35  241 225 160 108 1   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   81  240 253 253 119 25  0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   45  186 253 253 150 27  0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   16  93  252 253 187 0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   249 253 249 64  0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   46  130 183 253 253 207 2   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   39  148 229 253 253 253 250 182 0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   24  114 221 253 253 253 253 201 78  0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   23  66  213 253 253 253 253 198 81  2   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   18  171 219 253 253 253 253 195 80  9   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   55  172 226 253 253 253 253 244 133 11  0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   136 253 253 253 212 135 132 16  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0


Label:
5
```

MNIST Dataset – Data Exploration Script Training Data

```
import numpy as np

# n x 28 x 28 pixel training data
X_train = np.load("/homea/hpclab/train001/data/mnist/x_train.npy")

# n x 1 training labels
Y_train = np.load("/homea/hpclab/train001/data/mnist/y_train.npy")
```

```
print("Samples of 28 x 28 pixel matrices reserved for training")

# function for showing a character
def character_show(character):
    for y in character:
        row = ""
        for x in y:
            row += '{0: <4}'.format(x)
        print row
```

```
# view first 10 characters
for i in range (0,9):
  character_show(X_train[i])
  print("\n")
  print("Label:")
  print(Y_train[i])
  print("\n")
```

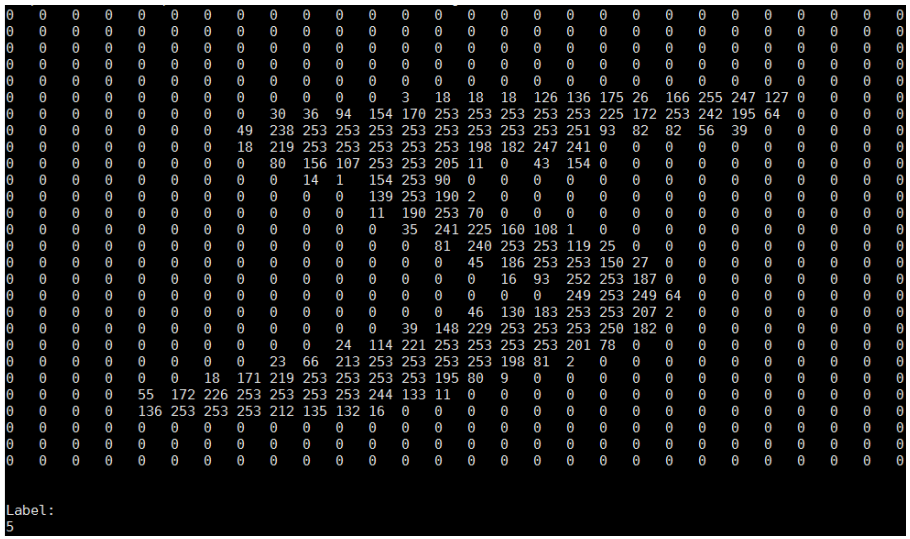
- Loading MNIST training datasets (X) with labels (Y) stored in a binary numpy format
- Format is 28 x 28 pixel values with grey level from 0 (white background) to 255 (black foreground)

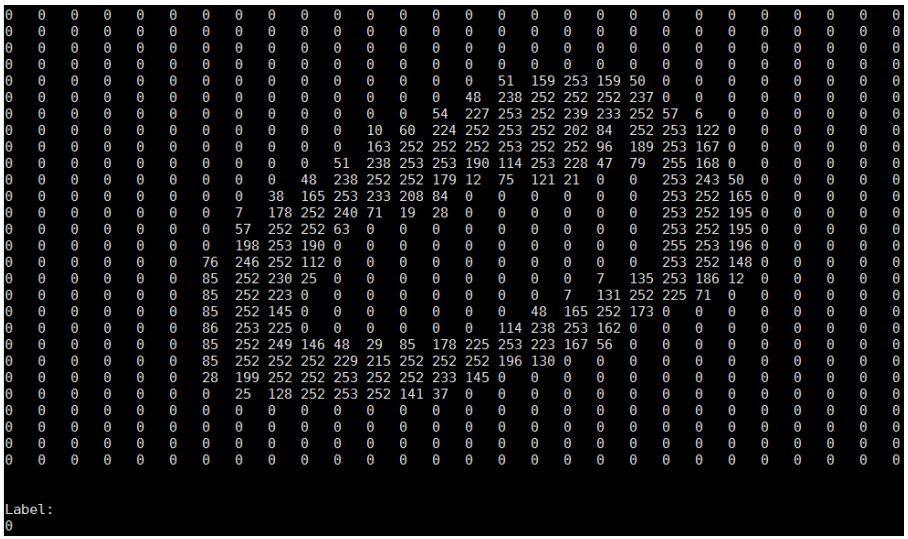
- Small helper function that prints row-wise one ‘hand-written‘ character with the grey levels stored in training dataset
- Should reveal the nature of the number (aka label)

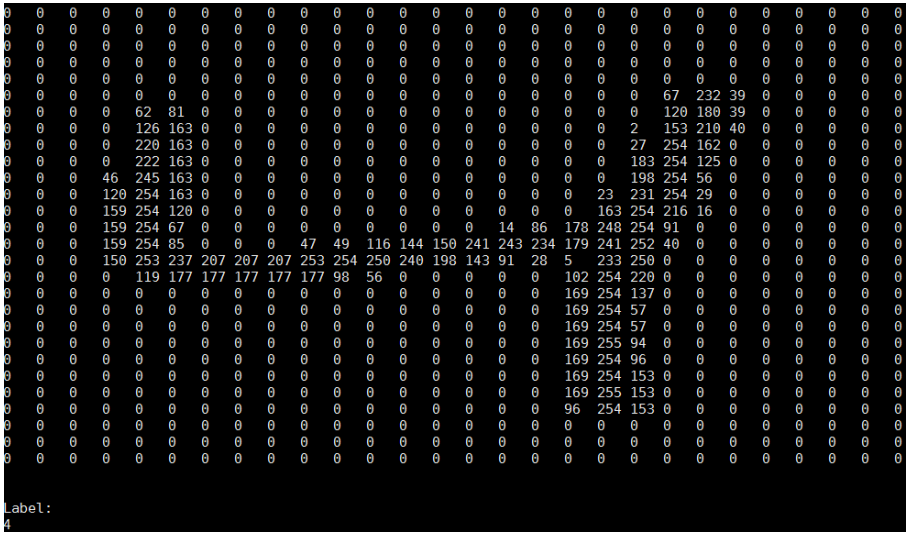
- Loop of the training dataset and the testing dataset (e.g. first 10 characters as shown here)
- At each loop interval the ‘hand-written‘ character (X) is printed in ‘matrix notation‘ & label (Y)

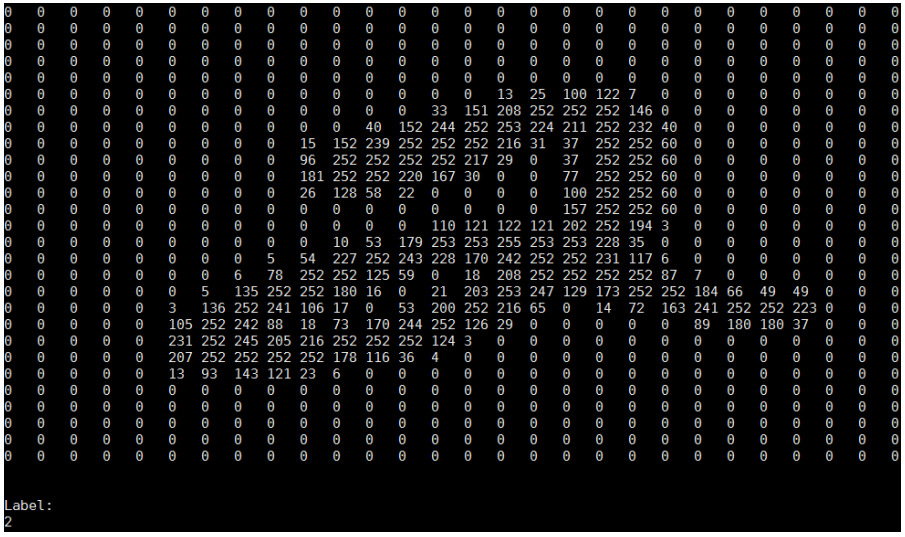
Exercises – Explore Data

MNIST Dataset – Exploration – Selected Training Samples









MNIST Dataset – Reshape & Normalization

```
import numpy as np

# n x 28 x 28 pixel training data
X_train = np.load("/homea/hpclab/train001/data/mnist/x_train.npy")

# n x 1 training labels
Y_train = np.load("/homea/hpclab/train001/data/mnist/y_train.npy")

# n x 28 x 28 pixel testing data
X_test = np.load("/homea/hpclab/train001/data/mnist/x_test.npy")

# n x 1 testing labels
Y_test = np.load("/homea/hpclab/train001/data/mnist/y_test.npy")

# reshape for the neural network 28 x 28 = 784
RESHAPED= 784
X_train = X_train.reshape(60000, RESHAPED)
X_test = X_test.reshape(10000, RESHAPED)
# specify type
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# normalization
X_train /= 255
X_test /= 255

# data output: number of samples
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')

# data output: number of values / sample
print(X_train.shape[1], 'input pixel values per train samples')
print(X_test.shape[1], 'input pixel values per test samples')

# data output: character
print(X_train[0])
```

- **Loading MNIST training datasets (X) and testing datasets (Y) stored in a binary numpy format with labels for X and Y**
- **Format is 28 x 28 pixel values with grey level from 0 (white background) to 255 (black foreground)**

- **Reshape from 28 x 28 matrix of pixels to 784 pixel values considered to be the input for the neural networks later**
- **Normalization is added for mathematical convenience since the computing with numbers get easier (not too large)**

Exercises – Reshape & Normalize Data

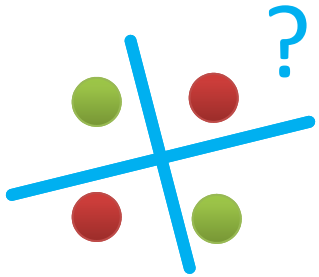
MNIST Dataset – Reshape & Normalization – Example

```
[train001@jrl09 mnist]$ python mnist-reshape.py
(60000, 'train samples')
(10000, 'test samples')
(784, 'input pixel values per train samples')
(784, 'input pixel values per test samples')
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.01176471 0.07058824 0.07058824 0.07058824
 0.49411765 0.53333336 0.6862745  0.10196079 0.6509804  1.
 0.96862745 0.49803922 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.11764706 0.14117648 0.36862746 0.6039216
 0.6666667  0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
 0.88235295 0.6745098  0.99215686 0.9490196  0.7647059  0.2509804
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.19215687
 0.93333334 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
 0.99215686 0.99215686 0.99215686 0.9843137  0.3647059  0.32156864
 0.32156864 0.21960784 0.15294118 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.07058824 0.85882354 0.99215686
 0.99215686 0.99215686 0.99215686 0.99215686 0.7764706  0.7137255
 0.96862745 0.94509804 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.3137255  0.6117647  0.41960785 0.99215686
 0.99215686 0.8039216  0.04313726 0.         0.16862746 0.6039216
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
```

(numbers are between 0 and 1)

Simple Application Example: Limitations of Perceptrons

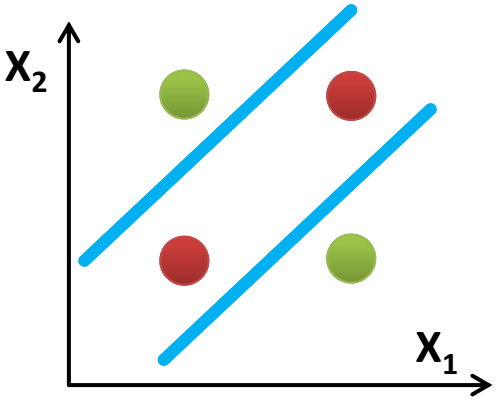
- Simple perceptrons fail: 'not linearly seperable'



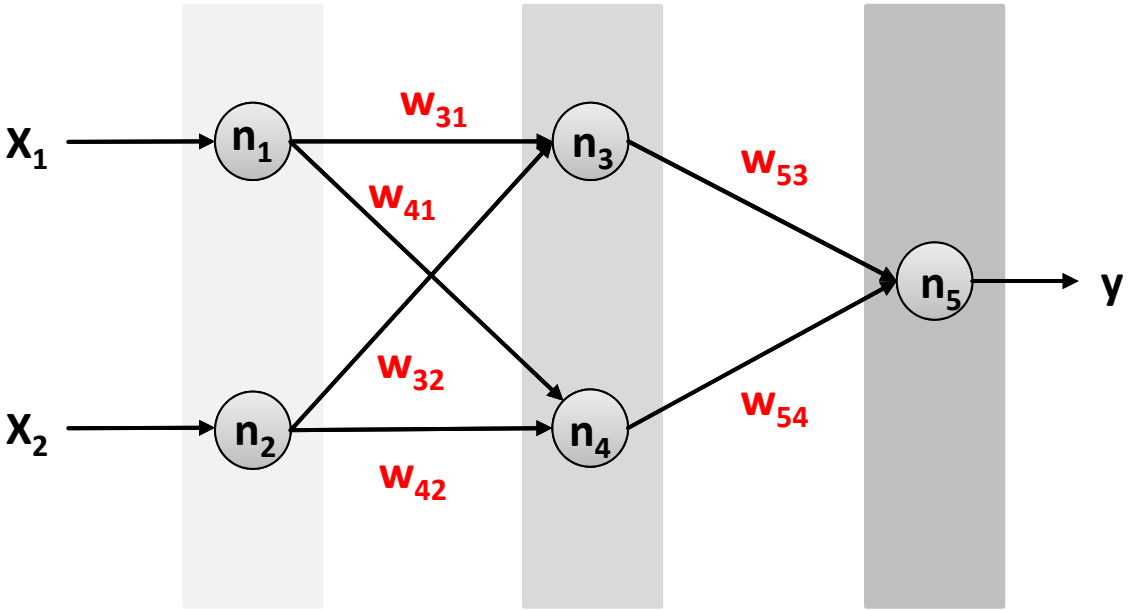
X_1	X_2	Y
0	0	-1
1	0	1
0	1	1
1	1	-1

Labelled Data Table

(Idea: instances can be classified using two lines at once to model XOR)



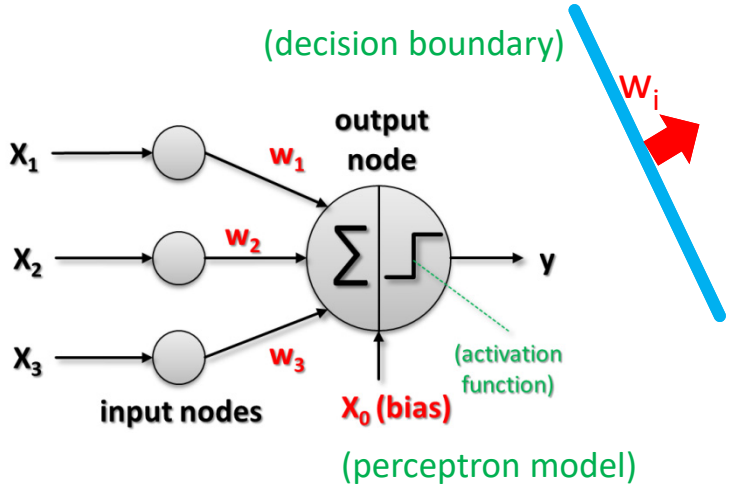
Decision Boundary

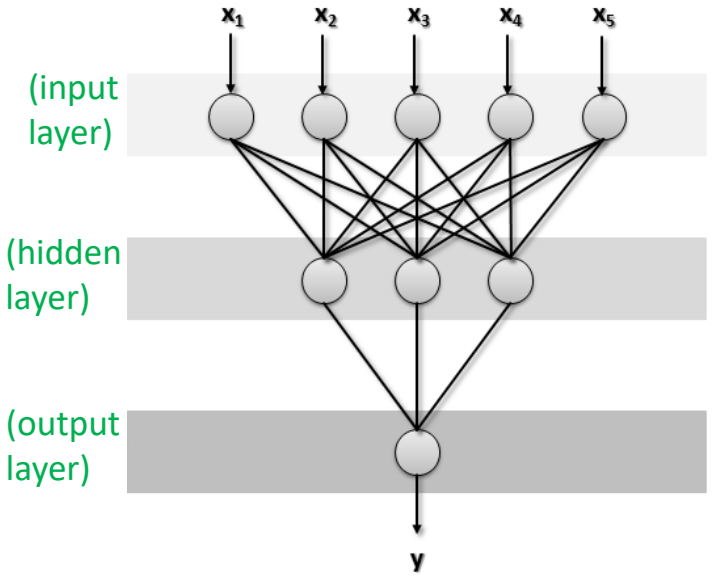


Two-Layer, feed-forward Artificial Neural Network topology

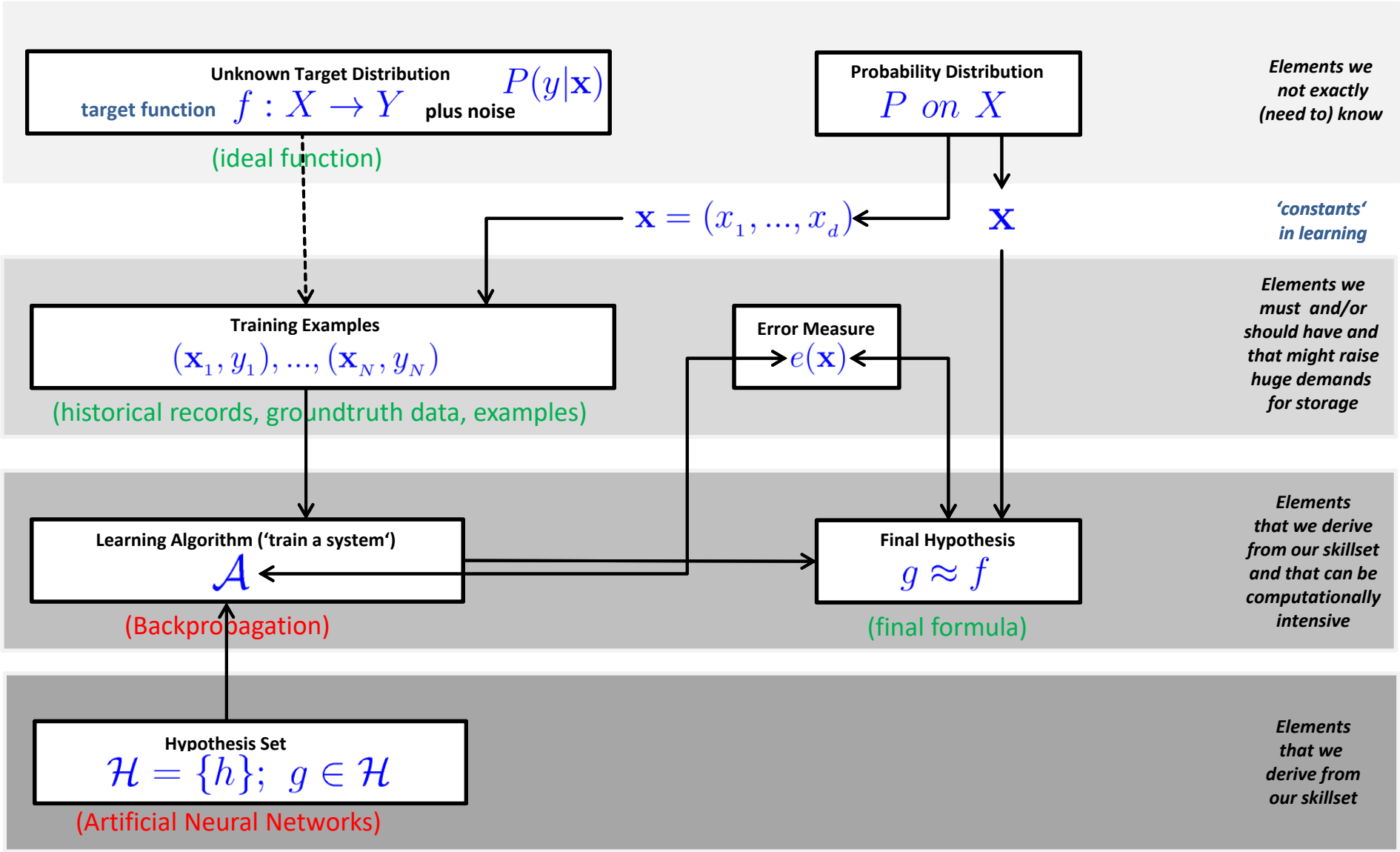
Multi Layer Perceptrons – Artificial Neural Networks

- Key Building Block
  - Perceptron learning model
  - Simplest linear learning model
  - Linearity in learned weights w_i
  - One decision boundary
- Artificial Neural Networks (ANNs)
  - Creating more complex structures
  - Enable the modelling of more complex relationships in the datasets
  - May contain several intermediary layers
  - E.g. 2-4 hidden layers with hidden nodes
  - Use of activation function that can produce output values that are nonlinear in their input parameters

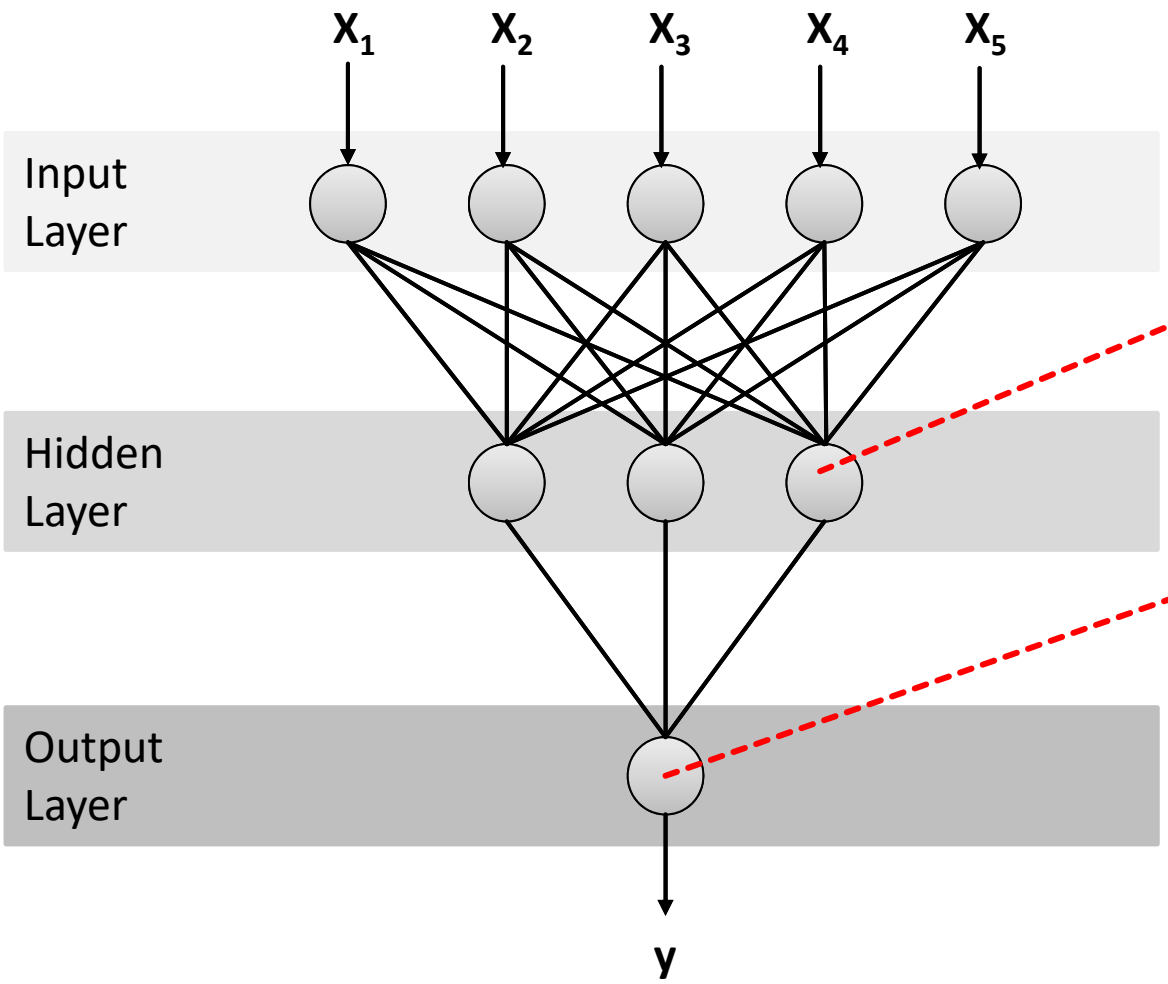


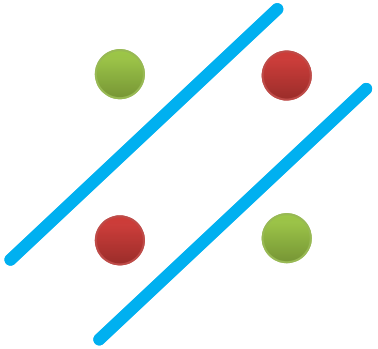


Solution Tools: Artificial Neural Networks Learning Model



Artificial Neural Networks (ANN) – Layers & Nodes





- **Think each hidden node as a 'simple perceptron' that each creates one hyperplane**
- **Think the output node simply combines the results of all the perceptrons to yield the 'decision boundary' above**
- **Feed-forward neural network: nodes in one layer are connected only to the nodes in the next layer ('a constraint of network construction')**

ANN - Learning Algorithm & Optimization

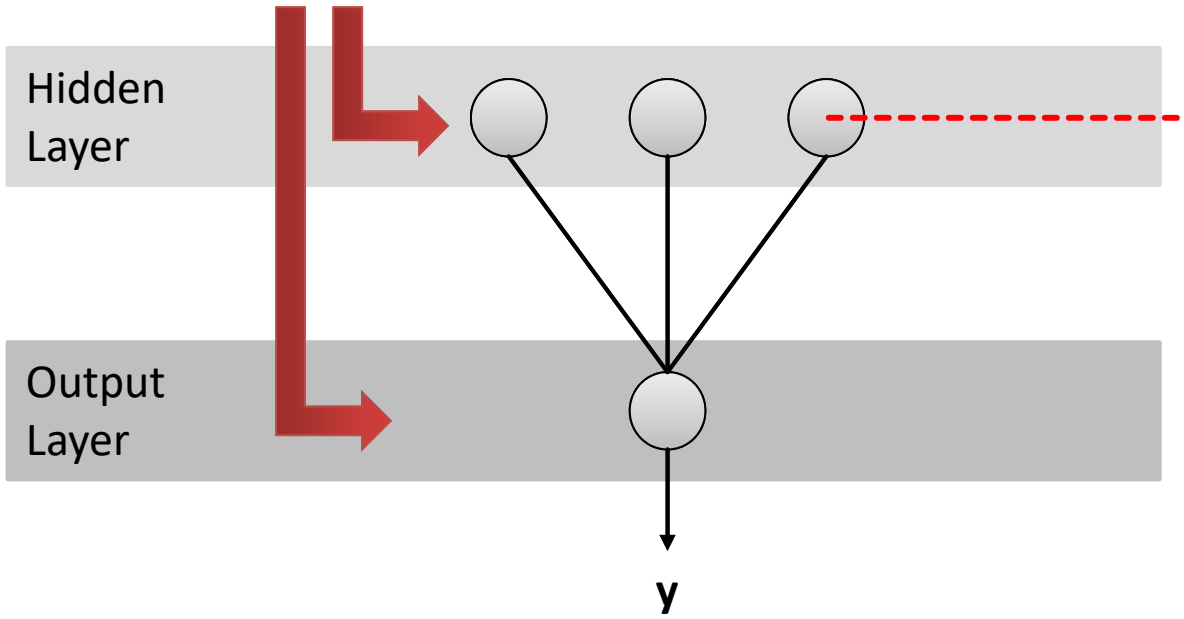
- Determine a set of weights **w** that 'minimize the total sum of squared errors':

y = sign ($\mathbf{w} \cdot \mathbf{x}$)

Linear perceptron

Sum of squared errors depend on w, because predicted class y is a 'function of the weights' assigned to the hidden and output nodes

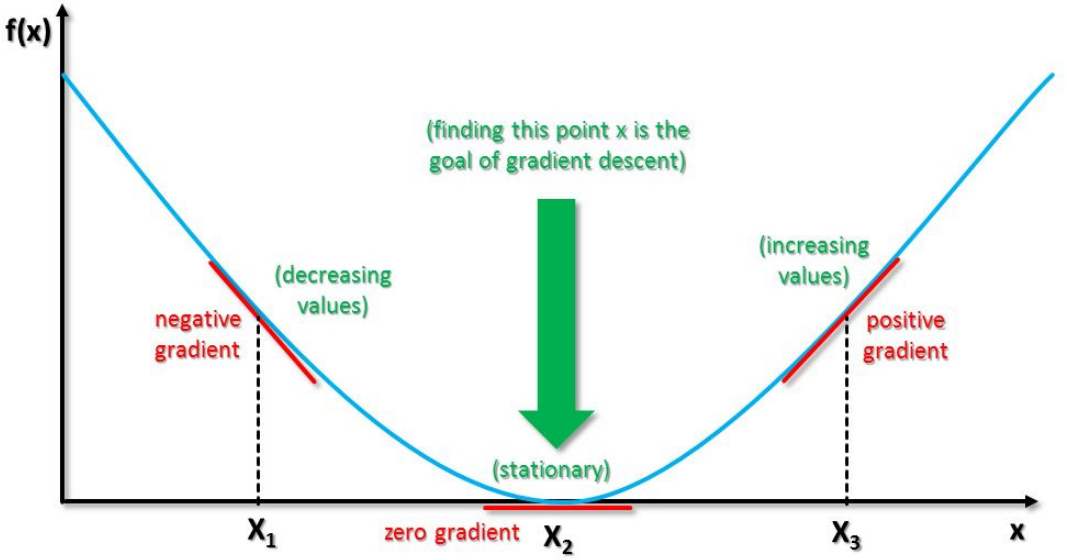
$$E(w) = \frac{1}{2}\sum_{i=1}^{N}(y_i - \acute{y}_i)^2$$



- Error term, associated with each hidden node

- Error function is quadratic in its parameters and a global minimum can be easily found
- Other objective / loss functions possible, e.g. categorical cross-entropy

Gradient Descent Method (1)



(minimization: substract gradient term because we move towards local minima)

(the derivative of f with respect to a)

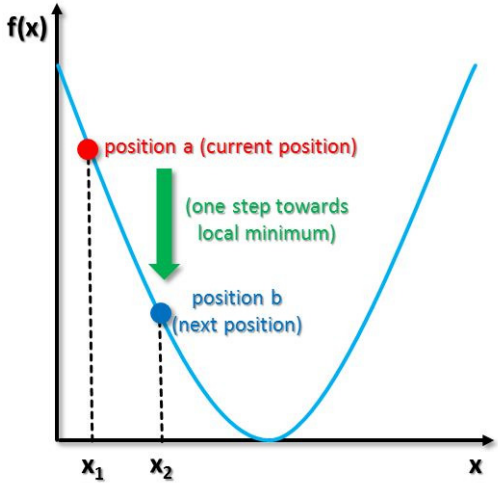
$$b = a - \gamma \nabla f(a)$$

(new position after the step)

(old position before the step)

(weighting factor known as step-size, can change at every iteration, also called learning rate)

(gradient term is steepest ascent)



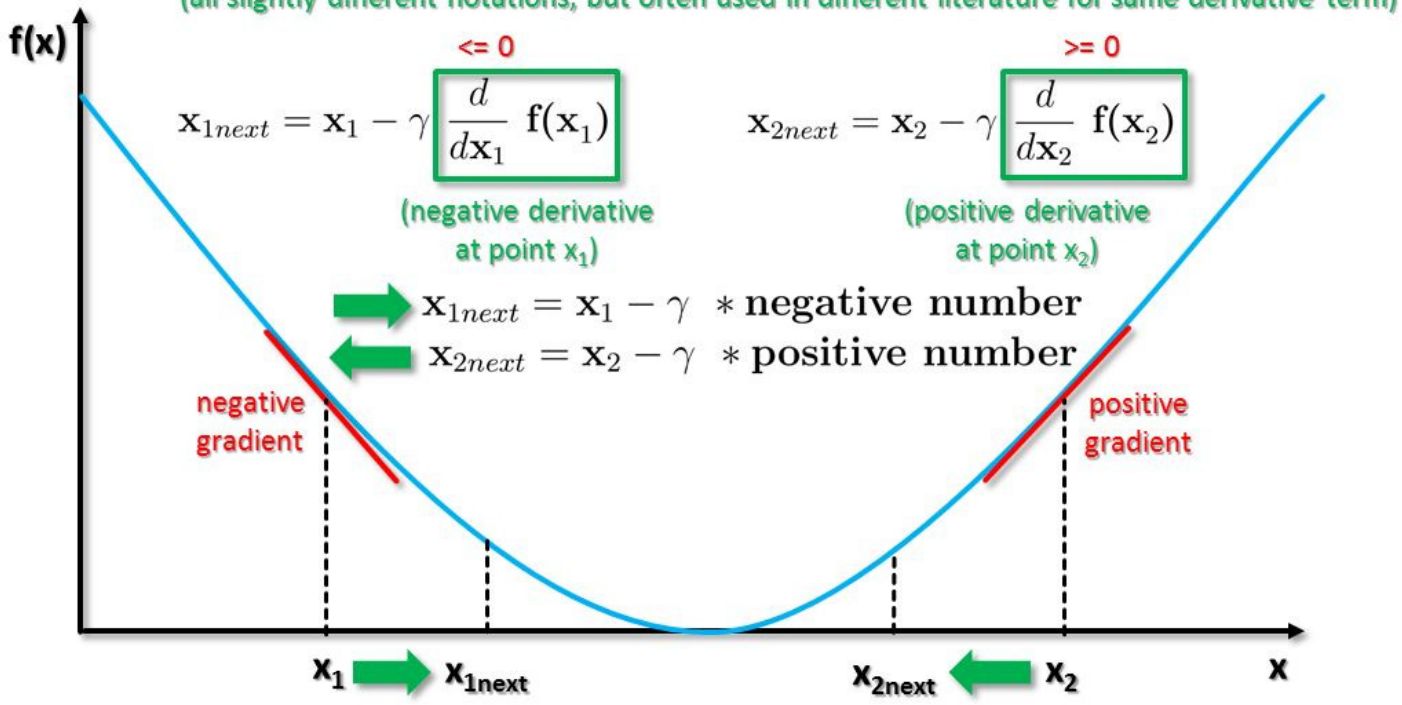
[16] Big Data Tips, Gradient Descent

Gradient Descent Method (2)

- **Gradient Descent (GD) uses all the training samples available for a step within a iteration**
- **Stochastic Gradient Descent (SGD) converges faster: only one training samples used per iteration**

$$\mathbf{b} = \mathbf{a} - \gamma \nabla \mathrm{f}(\mathbf{a}) \quad \mathbf{b} = \mathbf{a} - \gamma \frac{\partial}{\partial \mathbf{a}} \mathrm{f}(\mathbf{a}) \quad \mathbf{b} = \mathbf{a} - \gamma \frac{d}{d\mathbf{a}} \mathrm{f}(\mathbf{a})$$

(all slightly different notations, but often used in different literature for same derivative term)



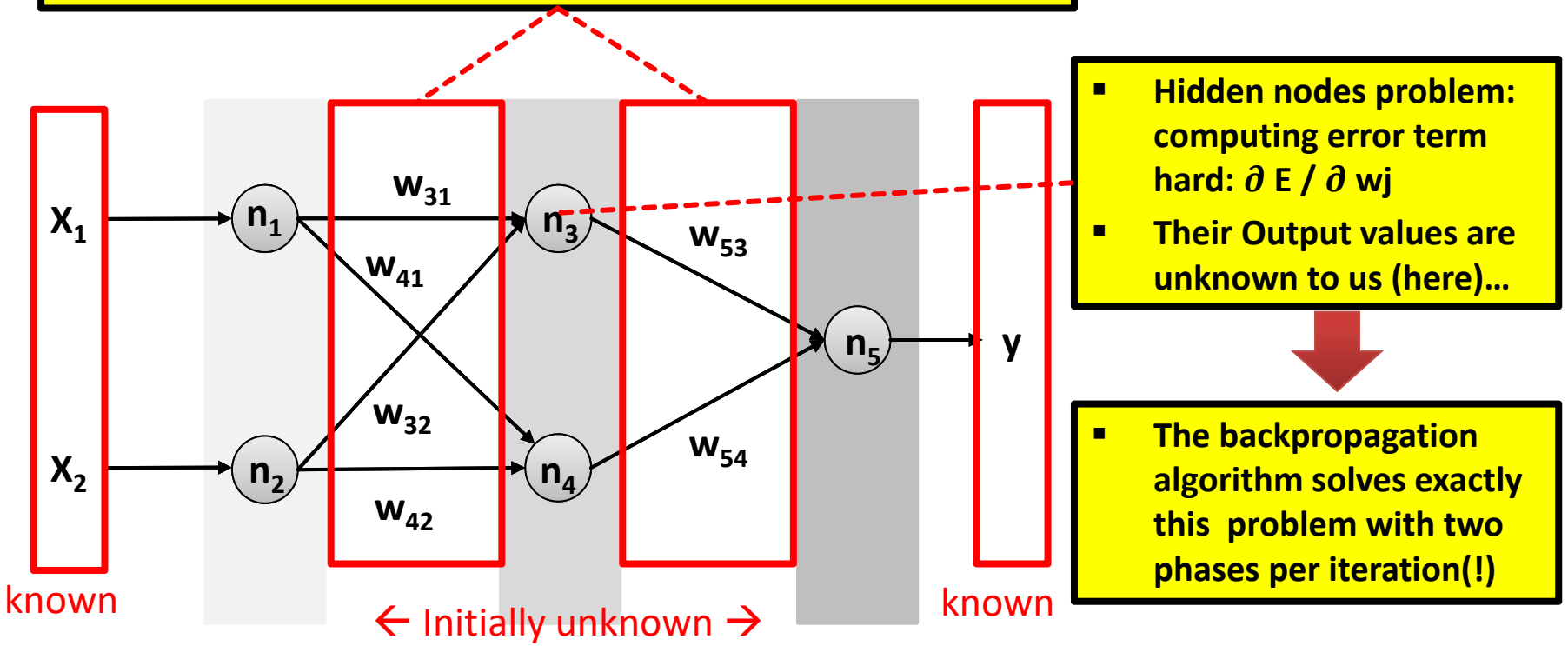
[16] Big Data Tips, Gradient Descent

ANN – Backpropagation Algorithm (BP) Basics

- One of the most widely used algorithms for supervised learning
  - Applicable in multi-layered feed-forward neural networks

- **'Gradient descent method' can be used to learn the weights of the output and hidden nodes of a artificial neural network**

[11] Introduction to Data Mining

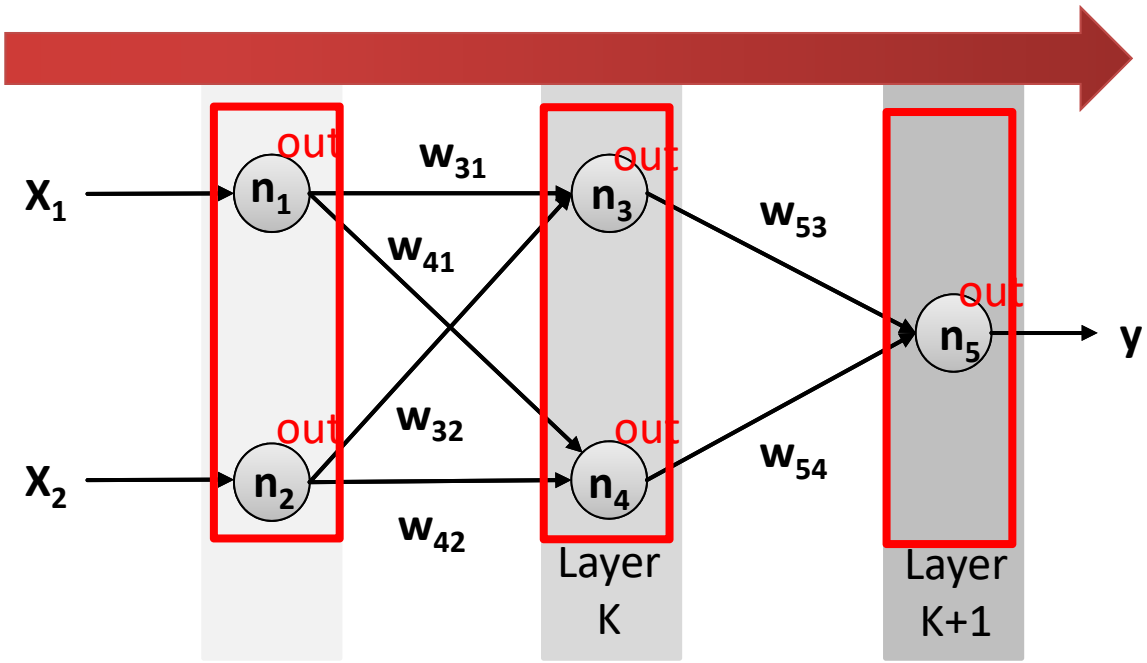


- **Hidden nodes problem: computing error term hard: $\partial E / \partial w_j$**
- **Their Output values are unknown to us (here)…**

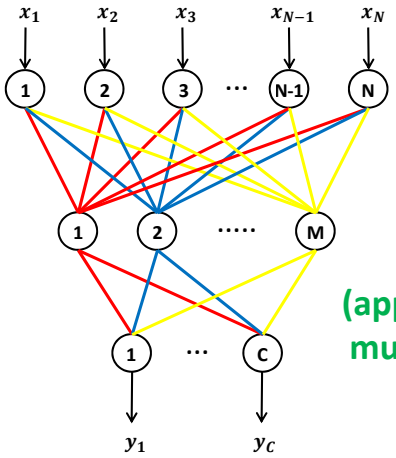
- **The backpropagation algorithm solves exactly this problem with two phases per iteration(!)**

ANN – Backpropagation Algorithm Forward Phase

1. ‘Forward phase (does not change weights, re-use old weights)’:
   - Weights obtained from the previous iteration are used to compute the output value of each neuron in the network (‘initialize weights randomly’)
   - Computation progresses in the ‘forward direction’, i.e. outputs ‘out’ of the neurons at level k are computed prior to level k+1



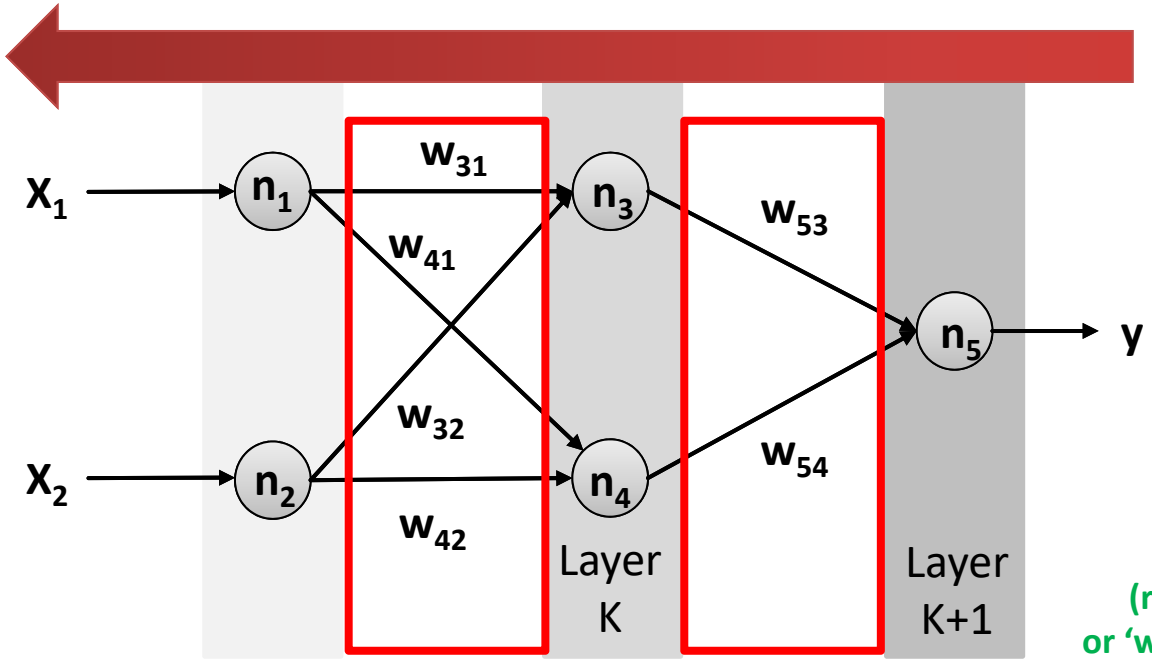
- **Use corresponding ‘activation function’ but with ‘old weights’**



(applicable for multiclass to)

ANN – Backpropagation Algorithm Backward Phase

2. ‘Backward phase (‘learning’ → change the weights in the ANN)’:
   - Weight update formula is applied in the ‘reverse direction’
   - Weights at level K + 1 are updated before the weights at level k
   - Idea: use the errors for neurons at layer k + 1 to estimate errors for neurons at layer k



$$w_j <- wj - \lambda \frac{\partial E(w)}{\partial w_j}$$

weight update formula
of the ‘gradient descent method’

Now that can compute
the error one-by-one

$$E_{in}(\mathbf{w}) + \frac{\lambda}{N}\mathbf{w}^T\mathbf{w}$$

(regularization method ‘weight decay‘
or ‘weight drop‘ is used in neural networks‘)

ANN – MNIST Dataset – Create ANN Blueprint

✓ Data Preprocessing done (i.e. data normalization, reshape, etc.)

1. Define a neural network topology
   - Which layers are required?
   - Think about input layer need to match the data – what data we had?
   - Maybe hidden layers?
   - Think Dense layer – Keras?
   - Think about final Activation as Softmay (cf. Day One) → output probability
2. Compile the model → model representation for Tensorflow et al.
   - Think about what loss function you want to use in your problem?
   - What is your optimizer strategy, e.g. SGD (cf. Day One)
3. Fit the model → the model learning takes place
   - How long you want to train (e.g. NB_EPOCHS)
   - How much samples are involved (e.g. BATCH_SIZE)

High-level Tools – Keras

- Keras is a high-level deep learning library implemented in Python that works on top of existing other rather low-level deep learning frameworks like Tensorflow, CNTK, or Theano
- The key idea behind the Keras tool is to enable faster experimentation with deep networks
- Created deep learning models run seamlessly on CPU and GPU via low-level frameworks

```
keras.layers.Dense(units,
                   activation=None,
                   use_bias=True,
                   kernel_initializer='glorot_uniform',
                   bias_initializer='zeros',
                   kernel_regularizer=None,
                   bias_regularizer=None,
                   activity_regularizer=None,
                   kernel_constraint=None,
                   bias_constraint=None)

keras.optimizers.SGD(lr=0.01,
                     momentum=0.0,
                     decay=0.0,
                     nesterov=False)
```

- Tool Keras supports inherently the creation of artificial neural networks using Dense layers and optimizers (e.g. SGD)
- Includes regularization (e.g. weight decay) or momentum

[17] Keras Python Deep Learning Library

Exercises – Create a Simple ANN Model – One Dense Layer

ANN – MNIST Dataset – Parameters & Data Normalization

```
import numpy as np
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.utils import np_utils

# ANN parameters
NB_CLASSES = 10
NB_EPOCH = 200
BATCH_SIZE = 128
VERBOSE = 1
N_HIDDEN = 128
OPTIMIZER = 'SGD'

# n x 28 x 28 pixel training data
X_train = np.load("/homea/hpclab/train001/data/mnist/x_train.npy")

# n x 1 training labels
Y_train = np.load("/homea/hpclab/train001/data/mnist/y_train.npy")

# n x 28 x 28 pixel testing data
X_test = np.load("/homea/hpclab/train001/data/mnist/x_test.npy")

# n x 1 testing labels
Y_test = np.load("/homea/hpclab/train001/data/mnist/y_test.npy")

# reshape for the neural network 28 x 28 = 784
RESHAPED= 784
X_train = X_train.reshape(60000, RESHAPED)
X_test = X_test.reshape(10000, RESHAPED)
# specify type
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# normalization
X_train /= 255
X_test /= 255

# data output: number of samples
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')

# data output: number of values / sample
print(X_train.shape[1], 'input pixel values per train samples')
print(X_test.shape[1], 'input pixel values per test samples')
```

- **NB_CLASSES: 10 Class Problem**
- **NB_EPOCH: number of times the model is exposed to the training set – at each iteration the optimizer adjusts the weights so that the objective function is minimized**
- **BATCH_SIZE: number of training instances taken into account before the optimizer performs a weight update**
- **OPTIMIZER: Stochastic Gradient Descent ('SGD') – only one training sample/iteration**

- **Data load shuffled between training and testing set in files**
- **Data preparation, e.g. X_train is 60000 samples / rows of 28 x 28 pixel values that are reshaped in 60000 x 784 including type specification (i.e. float32)**
- **Data normalization: divide by 255 – the max intensity value to obtain values in range [0,1]**

ANN – MNIST Dataset – A Simple Model

- The Sequential() Keras model is a linear pipeline (aka 'a stack') of various neural network layers including Activation functions of different types (e.g. softmax)
- Dense() represents a fully connected layer used in ANNs that means that each neuron in a layer is connected to all neurons located in the previous layer
- The non-linear Activation function 'softmax' represents a generalization of the sigmoid function – it squashes an n-dimensional vector of arbitrary real values into a n-dimenensional vector of real values in the range of 0 and 1 – here it aggregates 10 answers provided by the Dense layer with 10 neurons

```
# convert label vectors to binary matrices of classes
Y_train = np_utils.to_categorical(Y_train, NB_CLASSES)
Y_test = np_utils.to_categorical(Y_test, NB_CLASSES)

# simple ANN model
model = Sequential()
model.add(Dense(NB_CLASSES, input_shape=(RESHAPED,)))
model.add(Activation('softmax'))
model.summary()

# compilation
model.compile(loss='categorical_crossentropy', optimizer=OPTIMIZER, metrics=['accuracy'])

# fit the model
history = model.fit(X_train, Y_train, batch_size=BATCH_SIZE, epochs=NB_EPOCH, verbose=VERBOSE)

# evaluation
score = model.evaluate(X_test, Y_test, verbose=VERBOSE)
print('Test score: ', score[0])
print('Test accuracy: ', score[1])
```

$$L_i = -\Sigma_j t_{i,j} \log(p_{i,j})$$

- Loss function is a multi-class logarithmic loss: target is *ti,j* and prediction is *pi,j*
- Train the model ('fit')

Model Evaluation – Testing Phase & Confusion Matrix

- Model is fixed
  - Model is just used with the testset
  - Parameters are set
- Evaluation of model performance
  - Counts of test records that are incorrectly predicted
  - Counts of test records that are correctly predicted
  - E.g. create confusion matrix for a two class problem

Counting per sample		Predicted Class	
		Class = 1	Class = 0
Actual Class	Class = 1	f_{11}	f_{10}
	Class = 0	f_{01}	f_{00}

(serves as a basis for further performance metrics usually used)

Model Evaluation – Testing Phase & Performance Metrics

Counting per sample		Predicted Class	
		Class = 1	Class = 0
Actual Class	Class = 1	f_{11}	f_{10}
	Class = 0	f_{01}	f_{00}

(100% accuracy in learning often points to problems using machine learning methos in practice)

- Accuracy (usually in %)

$$Accuracy = \frac{number\ of\ correct\ predictions}{total\ number\ of\ predictions}$$

- Error rate

$$Error\ rate = \frac{number\ of\ wrong\ predictions}{total\ number\ of\ predictions}$$

ANN – MNIST Dataset – A Simple Model – Output

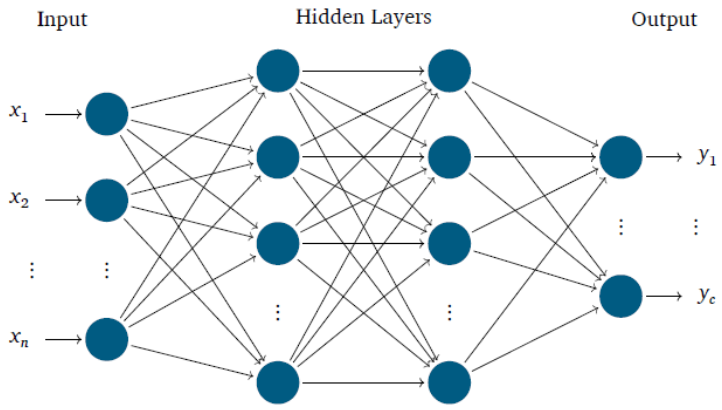
```
[train001@jrl09 scripts]$ tail mnist_out.5522445
   32/10000 [..............................] - ETA: 3s
 1504/10000 [===>..........................] - ETA: 0s
 3008/10000 [========>.....................] - ETA: 0s
 4544/10000 [============>.................] - ETA: 0s
 5952/10000 [================>.............] - ETA: 0s
 7392/10000 [====================>.........] - ETA: 0s
 8768/10000 [========================>....] - ETA: 0s
10000/10000 [==============================] - 0s 36us/step
('Test score: ', 0.2727356147527695)
('Test accuracy: ', 0.9228)
```

ANN – MNIST Dataset – Extend ANN Blueprint

- ✓ Data Preprocessing done (i.e. data normalization, reshape, etc.)
- ✓ Initial ANN topology existing
- ✓ Initial setup of model works (create, compile, fit)

- Extend the neural network topology
  - Which layers are required?
  - Think about input layer need to match the data – what data we had?
  - Maybe hidden layers?
  - How many hidden layers?
  - What activation function for which layer?
  - Think Dense layer – Keras?
  - Think about final Activation as Softmay (cf. Day One) → output probability

Exercises – Add Two Hidden Layers



ANN – MNIST Dataset – Add Two Hidden Layers

- A hidden layer in an ANN can be represented by a fully connected Dense layer in Keras by just specifying the number of hidden neurons in the hidden layer

- The non-linear Activation function 'relu' represents a so-called Rectified Linear Unit (ReLU) that only recently became very popular because it generates good experimental results in ANNs and more recent deep learning models – it just returns 0 for negative values and grows linearly for only positive values

```
# simple ANN model
model = Sequential()
model.add(Dense(N_HIDDEN, input_shape=(RESHAPED,)))
model.add(Activation('relu'))
model.add(Dense(N_HIDDEN))
model.add(Activation('relu'))
model.add(Dense(NB_CLASSES))
model.add(Activation('softmax'))
model.summary()

# compilation
model.compile(loss='categorical_crossentropy', optimizer=OPTIMIZER, metrics=['accuracy'])

# fit the model
history = model.fit(X_train, Y_train, batch_size=BATCH_SIZE, epochs=NB_EPOCH, verbose=VERBOSE)

# evaluation
score = model.evaluate(X_test, Y_test, verbose=VERBOSE)
print('Test score: ', score[0])
print('Test accuracy: ', score[1])
```

ANN 2 Hidden – MNIST Dataset – Job Script

```
[train001@jrl09 scripts]$ cp submit_train_ann_mnist.sh ~
```

```
#!/bin/bash -x
#SBATCH--nodes=1
#SBATCH--ntasks=1
#SBATCH--output=mnist_out.%j
#SBATCH--error=mnist_err.%j
#SBATCH--time=01:00:00
#SBATCH--mail-user=m.riedel@fz-juelich.de
#SBATCH--mail-type=ALL
#SBATCH--job-name=ANN_mnist_2hidden

#SBATCH--partition=gpus
#SBATCH--gres=gpu:1

#SBATCH--reservation=deep_learning

### location executable
MNIST=/homea/hpclab/train001/tools/mnist/mnist-ann-2hidden.py

module restore dl_tutorial_2

### submit
python $MNIST
```

ANN – MNIST Dataset – Job Submit & Check Output

```
[train001@jrl06 scripts]$ sbatch submit_train_ann_2hidden_mnist.sh
Submitted batch job 5522545
```

```
[train001@jrl06 scripts]$ more ann-2hidden-mnist_out.5522545
(60000, 'train samples')
(10000, 'test samples')
(784, 'input pixel values per train samples')
(784, 'input pixel values per test samples')
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 128)               100480
_________________________________________________________________
activation_1 (Activation)    (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               16512
_________________________________________________________________
activation_2 (Activation)    (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
_________________________________________________________________
activation_3 (Activation)    (None, 10)                0
=================================================================
Total params: 118,282
Trainable params: 118,282
Non-trainable params: 0
_________________________________________________________________
Epoch 1/200

  128/60000 [..............................] - ETA: 6:02 - loss: 2.3471 - acc: 0.0781
 2560/60000 [>.............................] - ETA: 18s - loss: 2.3044 - acc: 0.0902
 4992/60000 [=>............................] - ETA: 9s - loss: 2.2580 - acc: 0.1332
 7552/60000 [==>...........................] - ETA: 6s - loss: 2.2119 - acc: 0.2080
```

ANN 2 Hidden – MNIST Dataset – Output

```
 5632/10000 [===============>..............] - ETA: 0s
 7040/10000 [====================>.........] - ETA: 0s
 8448/10000 [=========================>.....] - ETA: 0s
 9824/10000 [============================>.] - ETA: 0s
10000/10000 [==============================] - 0s 37us/step
('Test score: ', 0.07480177137681167)
('Test accuracy: ', 0.9777)
```

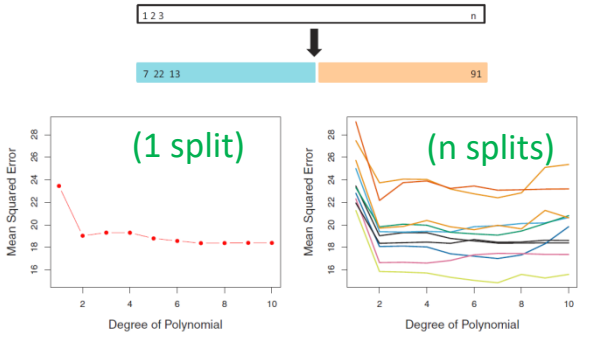
Validation & Model Selection – Terminology

- **The ‘Validation technique‘ should be used in all machine learning or data mining approaches**
- **Model assessment is the process of evaluating a models performance**
- **Model selection is the process of selecting the proper level of flexibility for a model**

modified from [6] ‘An Introduction to Statistical Learning’

- ‘Training error‘
  - Calculated when learning from data (i.e. dedicated training set)
- ‘Test error’
  - Average error resulting from using the model with ‘new/unseen data‘
  - ‘new/unseen data‘ was not used in training (i.e. dedicated test set)
  - In many practical situations, a dedicated test set is not really available
- ‘Validation Set‘
  - Split data into training & validation set
- ‘Variance‘ & ‘Variability‘
  - Result in different random splits (right)

(split creates a two subsets of comparable size)



Validation Technique – Formalization & Goal

- **Validation is a very important technique to estimate the out-of-sample performance of a model**
- **Main utility of regularization & validation is to control or avoid overfitting via model selection**

- Regularization & Validation
  - Approach: introduce a 'overfit penalty' that relates to model complexity
  - Problem: Not accurate values: 'better smooth functions'

(regularization uses a term that captures the overfit penalty)

$$E_{out}(h) = E_{in}(h) + \mathbf{overfit\ penalty}$$

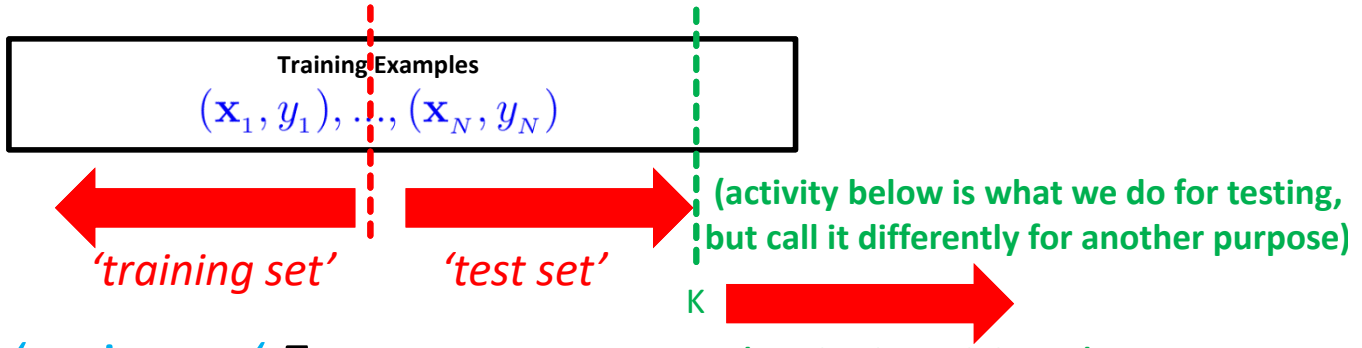
(minimize both to be better proxy for E_{out})

(validation estimates this quantity)

(regularization estimates this quantity)

- Validation (measuring E_{out} is not possible as this is an unknown quantity, another quantity is needed that is measurable that at least estimates it)
  - Goal 'estimate the out-of-sample error' (establish a quantity known as validation error)
  - Distinct activity from training and testing (testing also tries to estimate the E_{out})

Validation Technique – Pick one point & Estimate E_{out}



- Understanding 'estimate' E_{out}
  - On one out-of-sample point $(\mathbf{x}, y)$ the error is $e(h(\mathbf{x}), y)$
  - E.g. use squared error: $e(h(\mathbf{x}), f(\mathbf{x})) = (h(\mathbf{x}) - f(\mathbf{x}))^2$
    $e(h(\mathbf{x}), y) = (h(\mathbf{x}) - y)^2$
  - Use this quantity as estimate for E_{out} **(poor estimate)**
  - Term 'expected value' to formalize (probability theory)

(Taking into account the theory of Lecture 1 with probability distribution on X etc.)

Probability Distribution
P on X

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

(aka 'random variable')

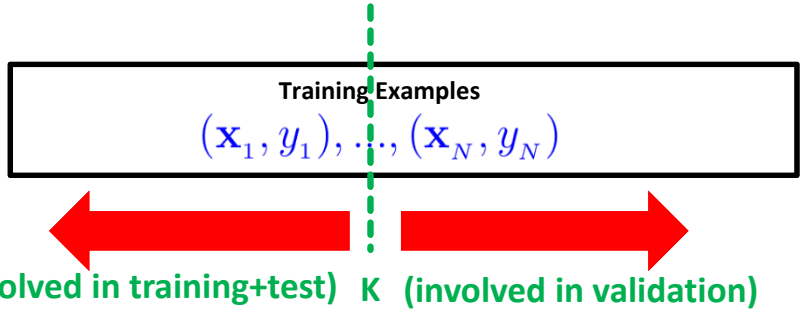
$\mathbb{E}[e(h(\mathbf{x}), y)] = E_{out}(h)$ **(aka the long-run average value of repetitions of the experiment)**

(one point as unbiased estimate of E_{out} that can have a high variance leads to bad generalization)

Validation Technique – Validation Set

- **Validation set consists of data that has been not used in training to estimate true out-of-sample**
- **Rule of thumb from practice is to take 20% (1/5) for validation of the learning model**

- Solution for high variance in expected values $\mathbb{E}[e(h(\mathbf{x}), y)] = E_{out}(h)$
  - Take a 'whole set' instead of just one point $(\mathbf{x}, y)$ for validation



(we need points not used in training to estimate the out-of-sample performance)

(we do the same approach with the testing set, but here different purpose)

  - Idea: K data points for validation

$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_K, y_K)$ (validation set)

$$E_{val}(h) = \frac{1}{K} \sum_{k=1}^{K} e(h(\mathbf{x})_k, y_k)$$ (validation error)

  - Expected value to 'measure' the out-of-sample error
  - 'Reliable estimate' if K is large

(expected values averaged over set)

$$\mathbb{E}[E_{val}(h)] = \frac{1}{K} \sum_{k=1}^{K} \mathbb{E}[e(h(\mathbf{x})_k, y_k)] = E_{out}$$

(on rarely used validation set, otherwise data gets contaminated)

(this gives a much better (lower) variance than on a single point given K is large)

Validation Technique – Model Selection Process

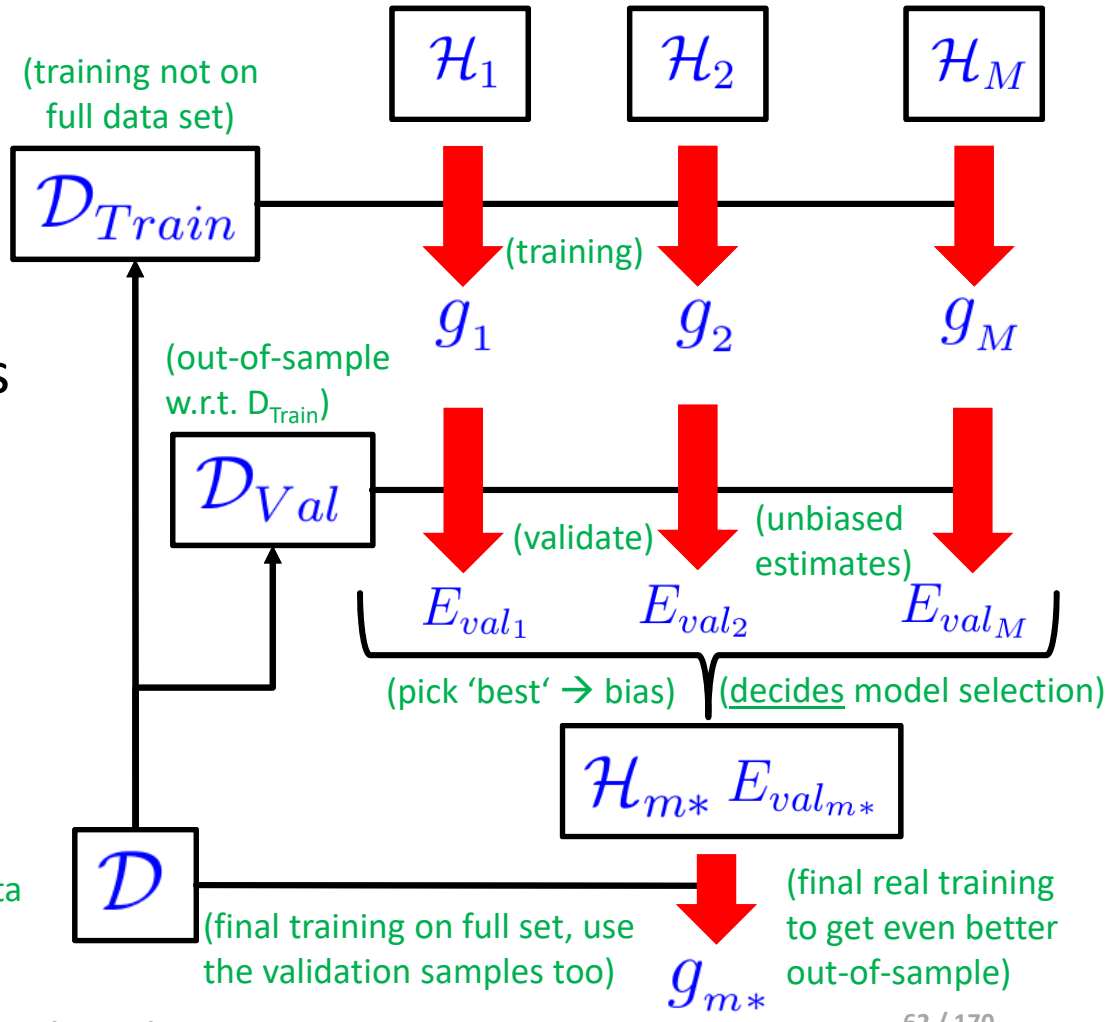
- **Model selection is choosing (a) different types of models or (b) parameter values inside models**
- **Model selection takes advantage of the validation error in order to decide → 'pick the best'**

Hypothesis Set

$$\mathcal{H} = \{h\};\ g \in \mathcal{H}$$

(set of candidate formulas across models)

- Many different models
  Use validation error to perform select decisions
- Careful consideration:
  - 'Picked means decided' hypothesis has already bias (→ contamination)
  - Using $\mathcal{D}_{Val}$ M times



Final Hypothesis

$$g_{m*} \approx f$$

(test this on unseen data good, but depends on availability in practice)

ANN 2 Hidden 1/5 Validation – MNIST Dataset

- If there is enough data available one rule of thumb is to take 1/5 (0.2) 20% of the datasets for validation only
- Validation data is used to perform model selection (i.e. parameter / topology decisions)

```
import numpy as np
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.utils import np_utils

# ANN parameters
NB_CLASSES = 10
NB_EPOCH = 200
BATCH_SIZE = 128
VERBOSE = 1
N_HIDDEN = 128
OPTIMIZER = 'SGD'
VALIDATION_SPLIT = 0.2 # 1/5 for validation
```

- The validation split parameter enables an easy validation approach during the model training (aka fit)
- Expectations should be a higher accuracy for unseen data since training data is less biased when using validation for model decisions (check statistical learning theory)
- VALIDATION_SPLIT: Float between 0 and 1
- Fraction of the training data to be used as validation data
- The model fit process will set apart this fraction of the training data and will not train on it
- Intead it will evaluate the loss and any model metrics on the validation data at the end of each epoch.

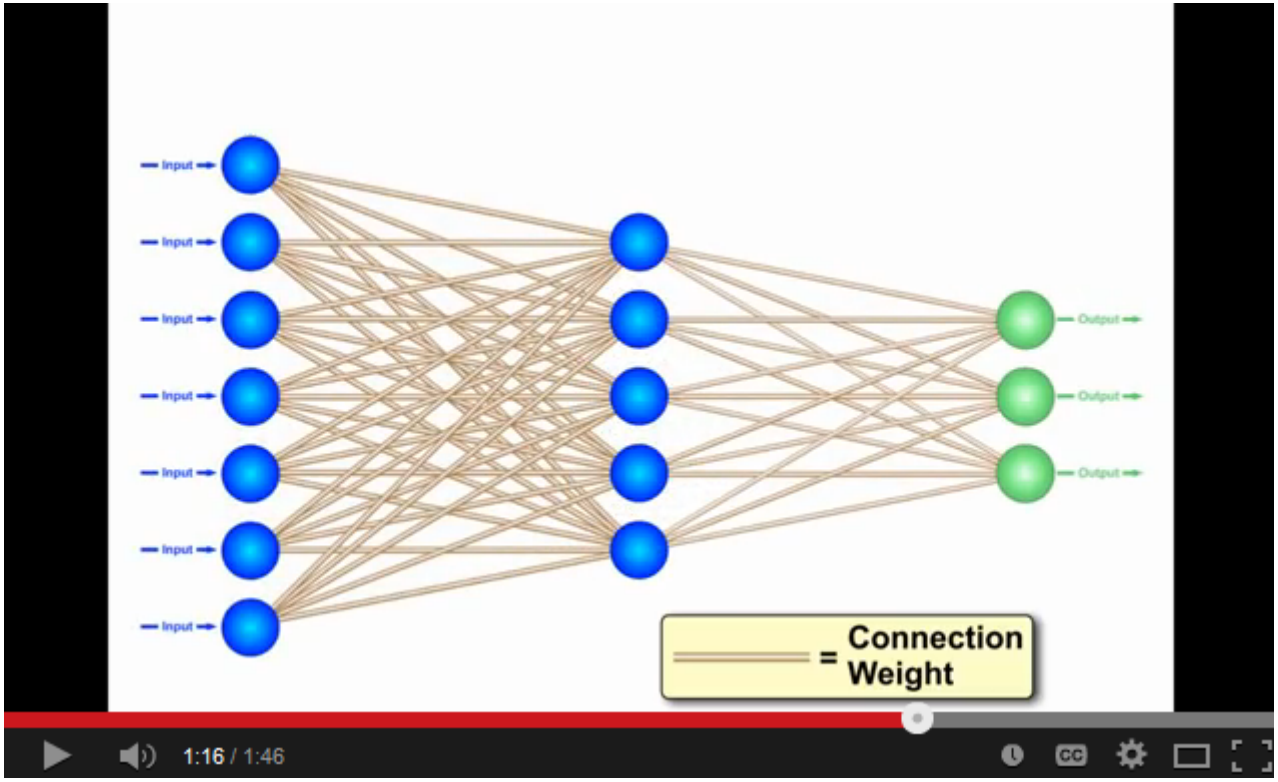
```
# compilation
model.compile(loss='categorical_crossentropy', optimizer=OPTIMIZER, metrics=['accuracy'])

# fit the model
history = model.fit(X_train, Y_train, batch_size=BATCH_SIZE, epochs=NB_EPOCH, verbose=VERBOSE, validation_split = VALIDATION_SPLIT)
```

ANN 2 Hidden – 1/5 Validation – MNIST Dataset – Output

```
 6784/10000 [===================>.........] - ETA: 0s
 8192/10000 [=======================>......] - ETA: 0s
 9568/10000 [============================>..] - ETA: 0s
10000/10000 [==============================] - 0s 37us/step
('Test score: ', 0.07833538340910454)
('Test accuracy: ', 0.9772)
```

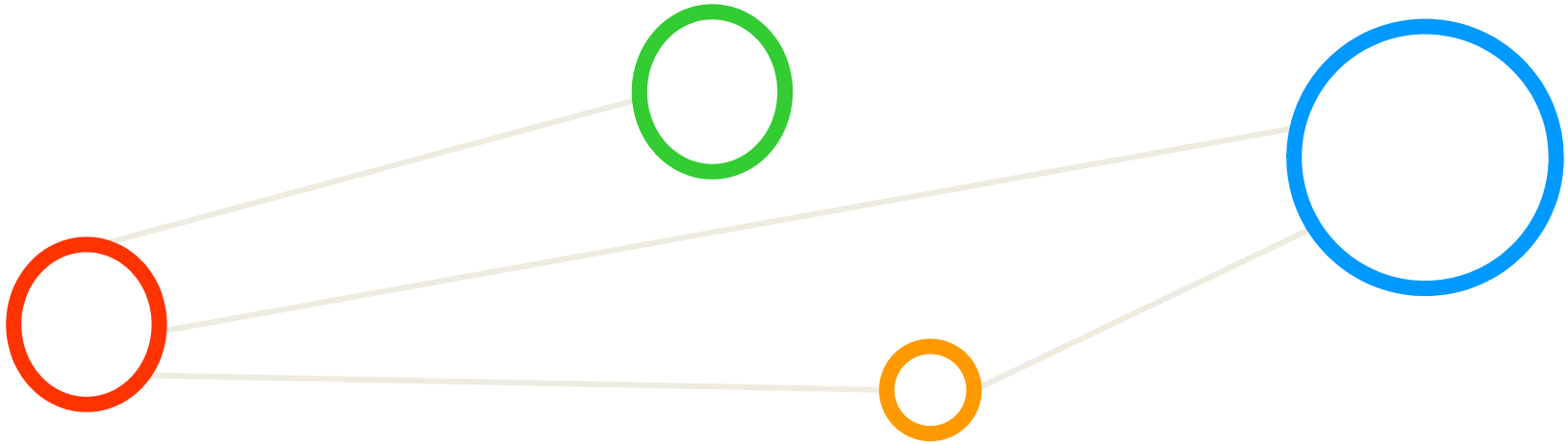
Exercises – Add Four/Six Hidden Layers → not Deep Learning

[Video] Towards Multi-Layer Perceptrons



[18] YouTube Video, Neural Networks – A Simple Explanation

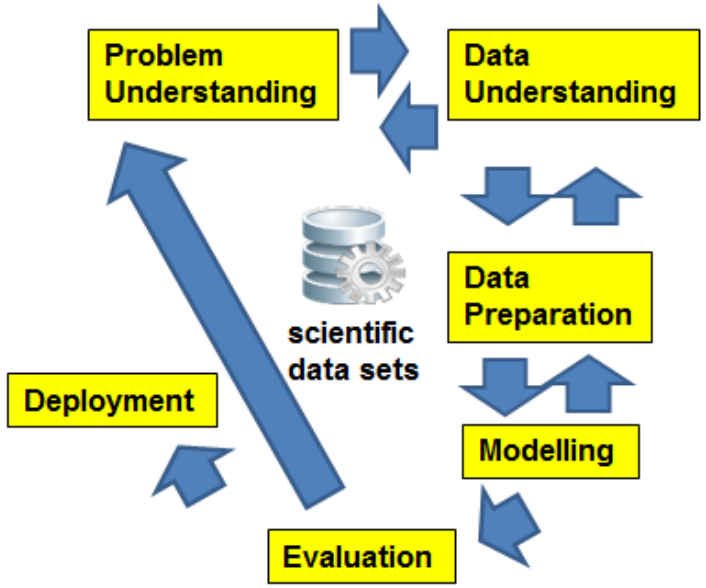
Appendix A: CRISP-DM Process



Summary: Systematic Process

- Systematic data analysis guided by a ‘standard process‘
  - Cross-Industry Standard Process for Data Mining (CRISP-DM)

- **A data mining project is guided by these six phases:**
  (1) Problem Understanding;
  (2) Data Understanding;
  (3) Data Preparation;
  (4) Modeling;
  (5) Evaluation;
  (6) Deployment



- Lessons Learned from Practice
  - Go back and forth between the different six phases

[10] C. Shearer, CRISP-DM model, Journal Data Warehousing, 5:13

1 – Problem (Business) Understanding

- **The Business Understanding phase consists of four distinct tasks: (A) Determine Business Objectives; (B) Situation Assessment; (C) Determine Data Mining Goal; (D) Produce Project Plan**

[11] CRISP-DM User Guide

- Task A – Determine Business Objectives
  - Background, Business Objectives, Business Success Criteria
- Task B – Situation Assessment
  - Inventory of Resources, Requirements, Assumptions, and Contraints
  - Risks and Contingencies, Terminology, Costs & Benefits
- Task C – Determine Data Mining Goal
  - Data Mining Goals and Success Criteria
- Task D – Produce Project Plan
  - Project Plan
  - Initial Assessment of Tools & Techniques

2 – Data Understanding

- **The Data Understanding phase consists of four distinct tasks:**
  (A) Collect Initial Data; (B) Describe Data; (C) Explore Data; (D) Verify Data Quality

[11] CRISP-DM User Guide

- Task A – Collect Initial Data
  - Initial Data Collection Report
- Task B – Describe Data
  - Data Description Report
- Task C – Explore Data
  - Data Exploration Report
- Task D – Verify Data Quality
  - Data Quality Report

3 – Data Preparation

- **The Data Preparation phase consists of six distinct tasks: (A) Data Set; (B) Select Data; (C) Clean Data; (D) Construct Data; (E) Integrate Data; (F) Format Data**

[11] CRISP-DM User Guide

- Task A – Data Set
  - Data set description
- Task B – Select Data
  - Rationale for inclusion / exclusion
- Task C – Clean Data
  - Data cleaning report
- Task D – Construct Data
  - Derived attributes, generated records
- Task E – Integrate Data
  - Merged data
- Task F – Format Data
  - Reformatted data

4 – Modeling

- **The Data Preparation phase consists of four distinct tasks: (A) Select Modeling Technique; (B) Generate Test Design; (C) Build Model; (D) Assess Model;**

[11] CRISP-DM User Guide

- Task A – Select Modeling Technique
  - Modeling assumption, modeling technique
- Task B – Generate Test Design
  - Test design
- Task C – Build Model
  - Parameter settings, models, model description
- Task D – Assess Model
  - Model assessment, revised parameter settings

5 – Evaluation

- **The Data Preparation phase consists of three distinct tasks: (A) Evaluate Results; (B) Review Process; (C) Determine Next Steps**

[11] CRISP-DM User Guide

- Task A – Evaluate Results
  - Assessment of data mining results w.r.t. business success criteria
  - List approved models
- Task B – Review Process
  - Review of Process
- Task C – Determine Next Steps
  - List of possible actions, decision

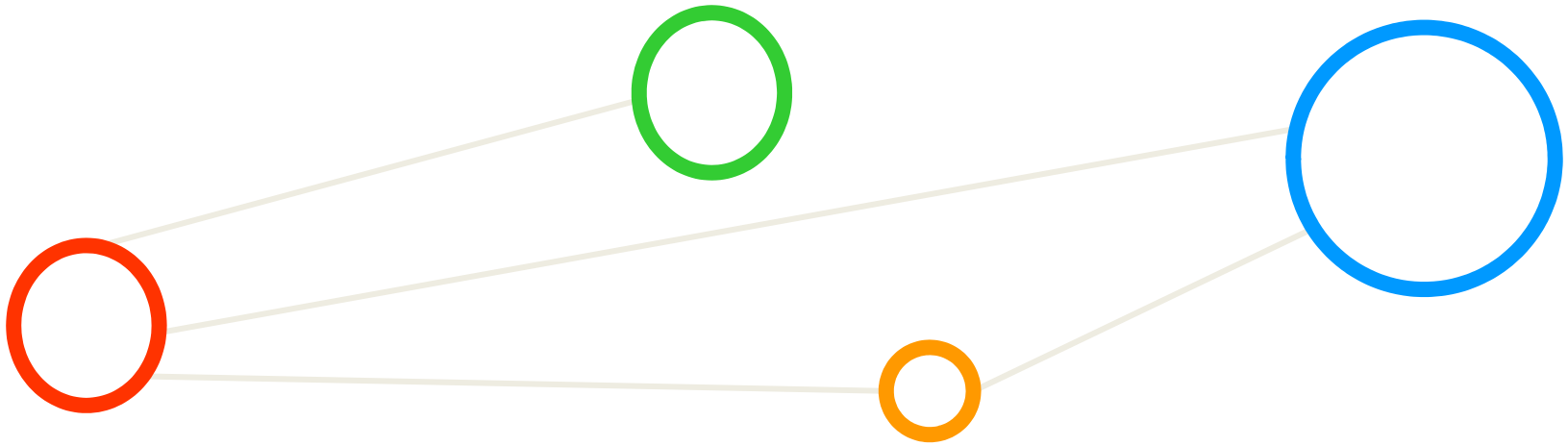
6 – Deployment

- **The Data Preparation phase consists of three distinct tasks: (A) Plan Deployment; (B) Plan Monitoring and Maintenance; (C) Produce Final Report; (D) Review Project**

[11] CRISP-DM User Guide

- Task A – Plan Deployment
  - Establish a deployment plan
- Task B – Plan Monitoring and Maintenance
  - Create a monitoring and maintenance plan
- Task C – Product Final Report
  - Create final report and provide final presentation
- Task D – Review Project
  - Document experience, provide documentation

Appendix B: Learning Theory Basics



Learning Approaches – Supervised Learning – Formalization

- Each observation of the predictor measurement(s) has an associated response measurement:
  - Input $\mathbf{x} = x_1, ..., x_d$
  - Output $y_i, i = 1, .., n$
  - Data $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)$

Training Examples

$$(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)$$

(historical records, groundtruth data, examples)

- Goal: Fit a model that relates the response to the predictors
  - Prediction: Aims of accurately predicting the response for future observations
  - Inference: Aims to better understanding the relationship between the response and the predictors

- **Supervised learning approaches fits a model that related the response to the predictors**
- **Supervised learning approaches are used in classification algorithms such as SVMs**
- **Supervised learning works with data = [input, correct output]**

[6] An Introduction to Statistical Learning

Feasibility of Learning

- **Statistical Learning Theory deals with the problem of finding a predictive function based on data**

[13] Wikipedia on 'statistical learning theory'

- Theoretical framework underlying practical learning algorithms
  - E.g. Support Vector Machines (SVMs)
  - Best understood for 'Supervised Learning'

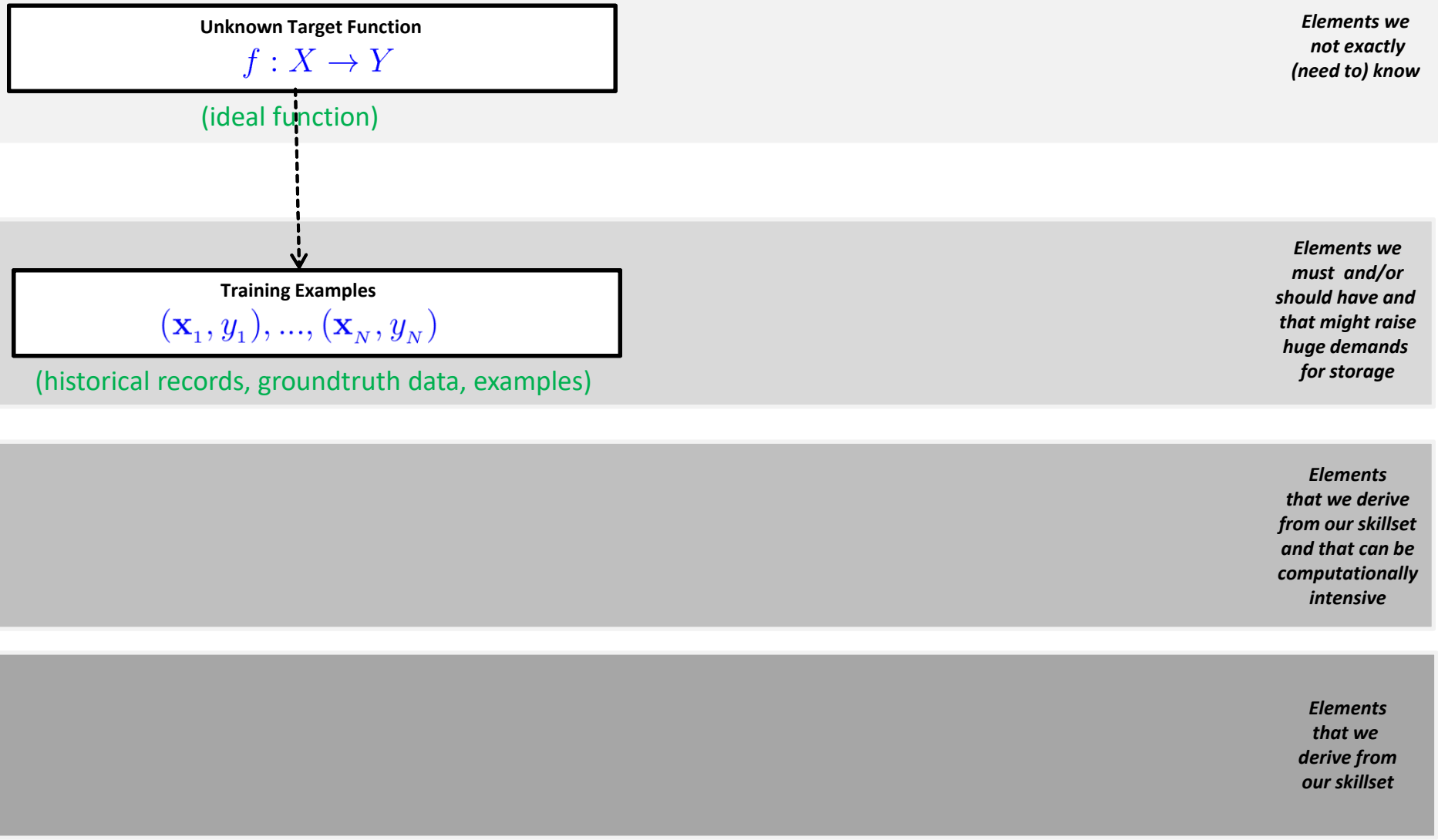
- Theoretical background used to solve 'A learning problem'
  - Inferring one 'target function' that maps between input and output
  - Learned function can be used to predict output from future input (fitting existing data is not enough)

Unknown Target Function

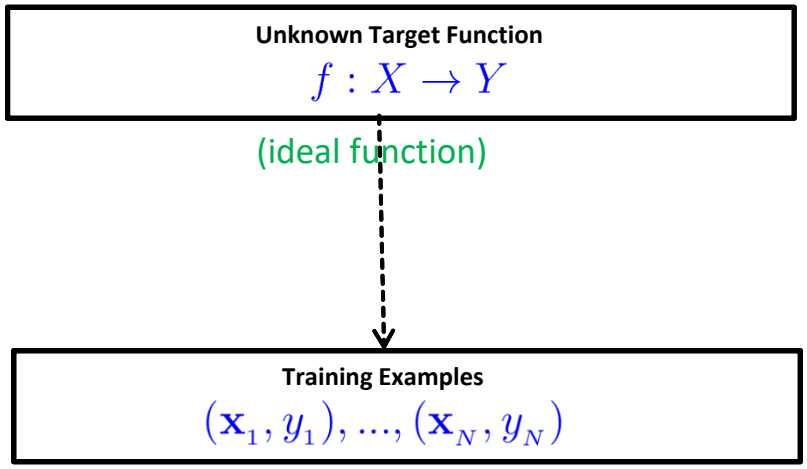
$$f : X \rightarrow Y$$

(ideal function)

Mathematical Building Blocks (1)

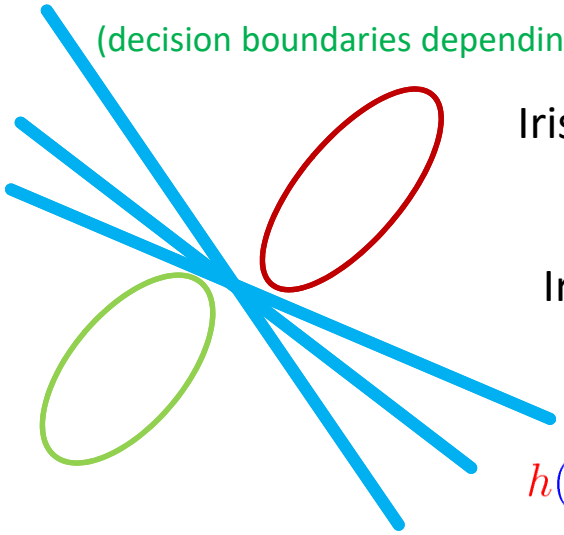


Mathematical Building Blocks (1) – Our Linear Example



1. Some pattern exists
2. No exact mathematical formula (i.e. target function)
3. Data exists

(if we would know the exact target function we dont need machine learning, it would not make sense)



Iris-virginica if $\sum_{i=1}^{d} w_i x_i > threshold$

Iris-setosa if $\sum_{i=1}^{d} w_i x_i < threshold$

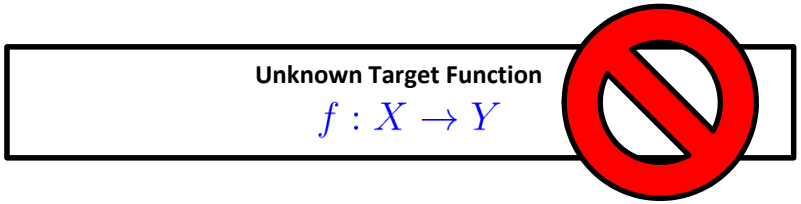
(w_i and threshold are still unknown to us)

$$h(\mathbf{x}) = sign\left(\left(\sum_{i=1}^{d} w_i x_i\right) - threshold\right); h \in \mathcal{H}$$

(we search a function similiar like a target function)

Feasibility of Learning – Hypothesis Set & Final Hypothesis

- The ‘ideal function‘ will remain unknown in learning

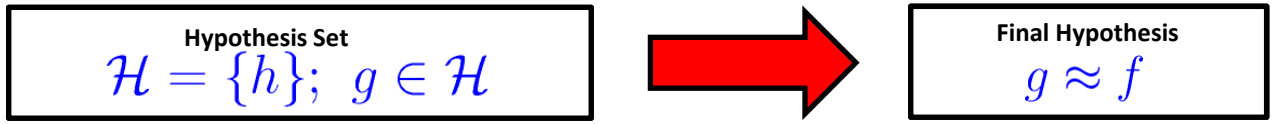


  - Impossible to know and learn from data
  - If known a straightforward implementation would be better than learning
  - E.g. hidden features/attributes of data not known or not part of data

- But ‘(function) approximation‘ of the target function is possible
  - Use training examples to learn and approximate it
  - Hypothesis set $\mathcal{H}$ consists of m different hypothesis (candidate functions)

$$\mathcal{H} = \{h_1, \ldots, h_m\};$$

‘select one function‘ that best approximates $g : X \rightarrow Y$



Feasibility of Learning – Understanding the Hypothesis Set

Hypothesis Set

$$\mathcal{H} = \{h\};\ g \in \mathcal{H}$$

$$\mathcal{H} = \{h_1, \ldots, h_m\};$$

(all candidate functions derived from models and their parameters)

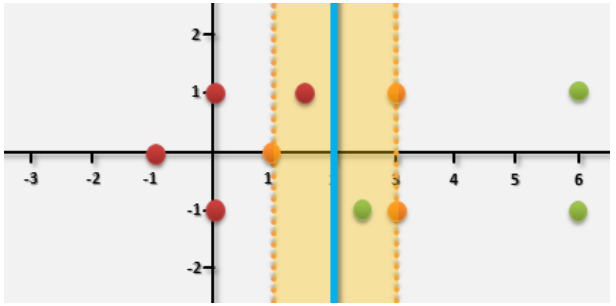
- **Already a change in model paramters of h_1, …, h_m means a completey different model**

'select one function' that best approximates

Final Hypothesis

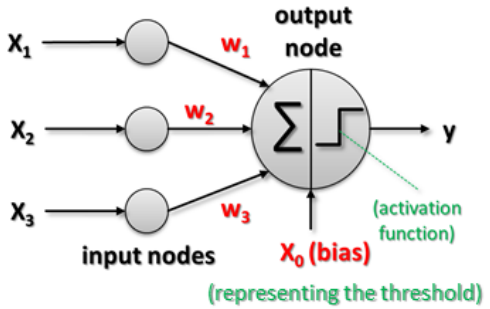
$$g \approx f$$

h_1



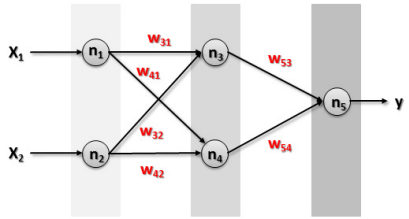
(e.g. support vector machine model)

h_2



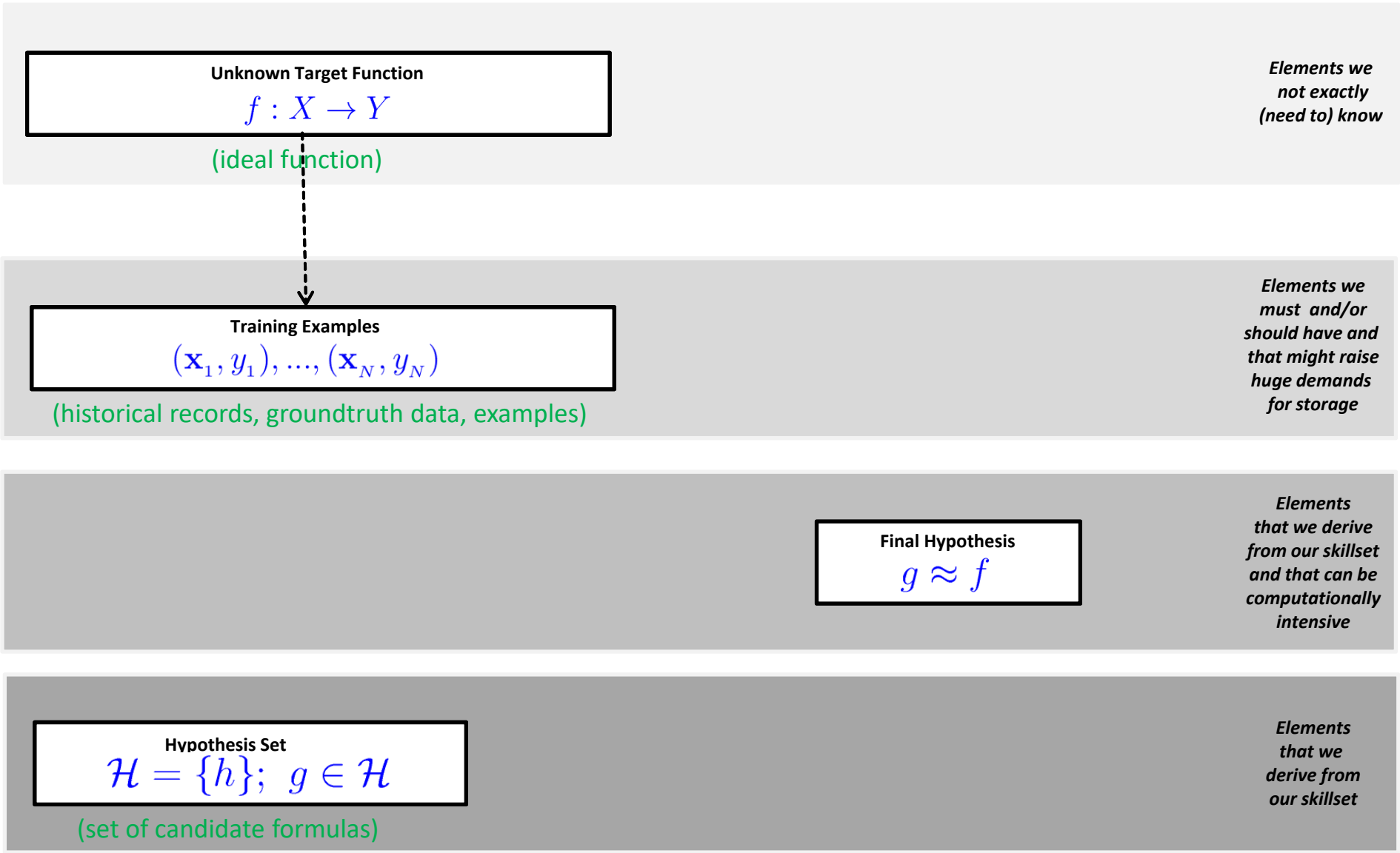
(e.g. linear perceptron model)

h_m



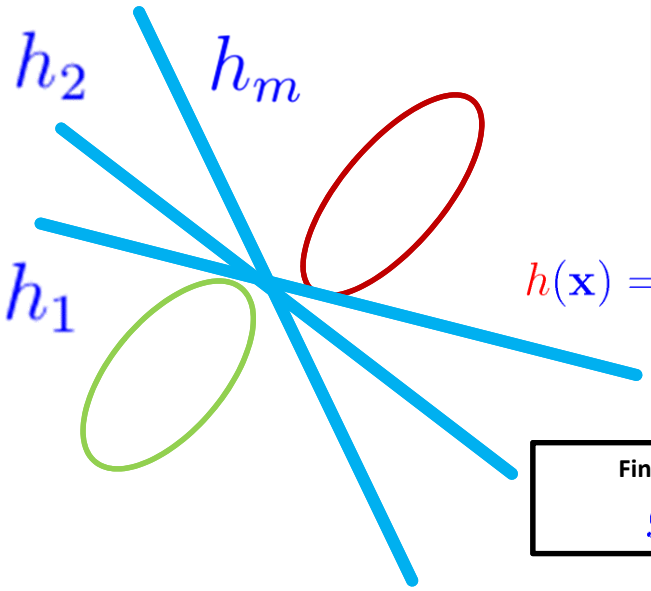
(e.g. artificial neural network model)

Mathematical Building Blocks (2)



Mathematical Building Blocks (2) – Our Linear Example

(decision boundaries depending on f)



- **Already a change in model paramters of h_1, …, h_m means a completey different model**

$$\mathcal{H} = \{h_1, \ldots, h_m\};$$

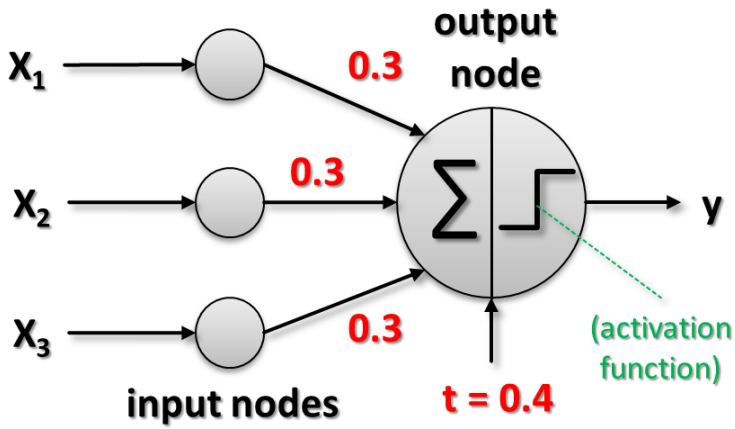
(we search a function similiar like a target function)

$$h(\mathbf{x}) = sign\left(\left(\sum_{i=1}^{d} w_i x_i\right) - threshold\right); h \in \mathcal{H}$$

Final Hypothesis

$$g \approx f$$

$$h(\mathbf{x}) = sign\left(\left(\sum_{i=1}^{d} w_i x_i\right) - threshold\right); h \in \mathcal{H}$$



(trained perceptron model and our selected final hypothesis)

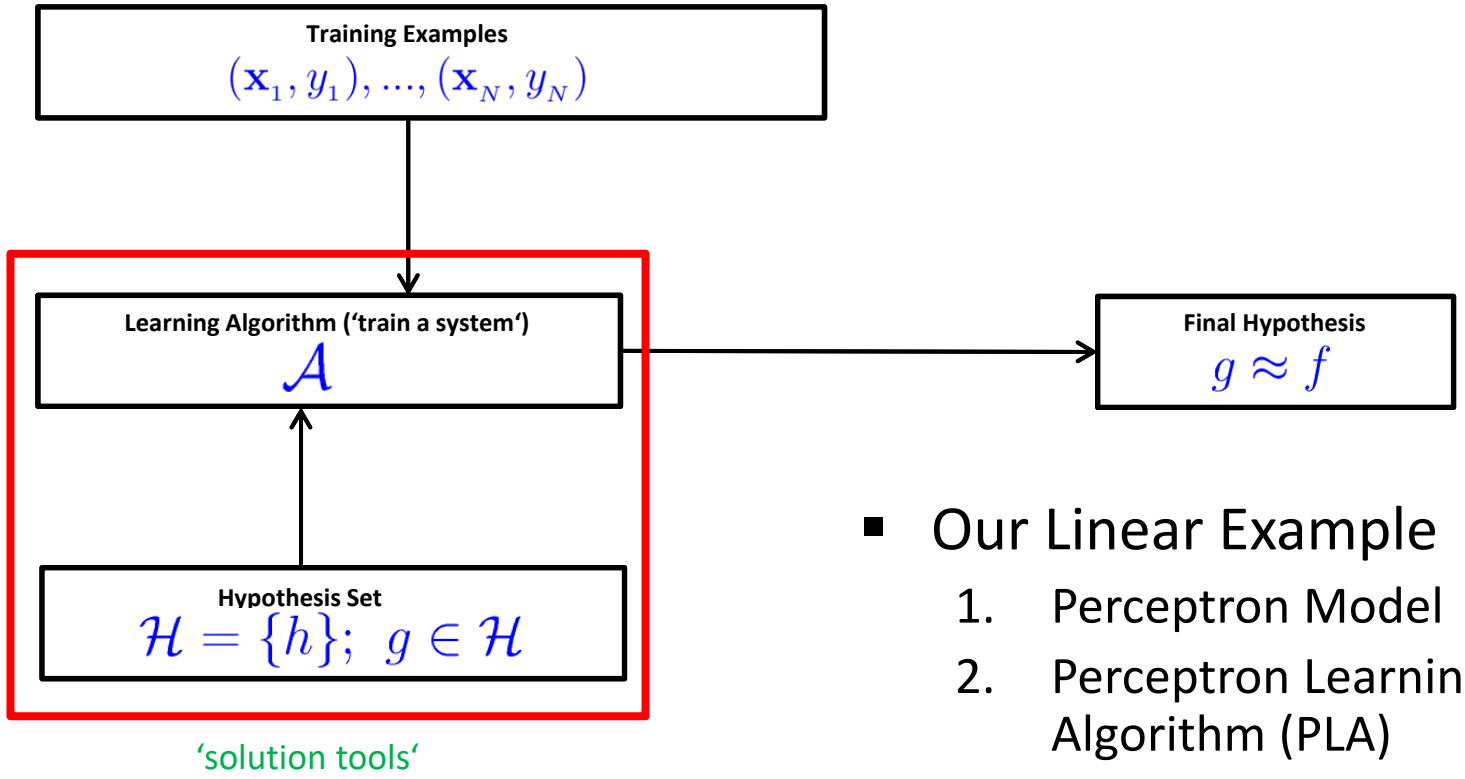
Hypothesis Set

$$\mathcal{H} = \{h\};\ g \in \mathcal{H}$$

(Perceptron model – linear model)

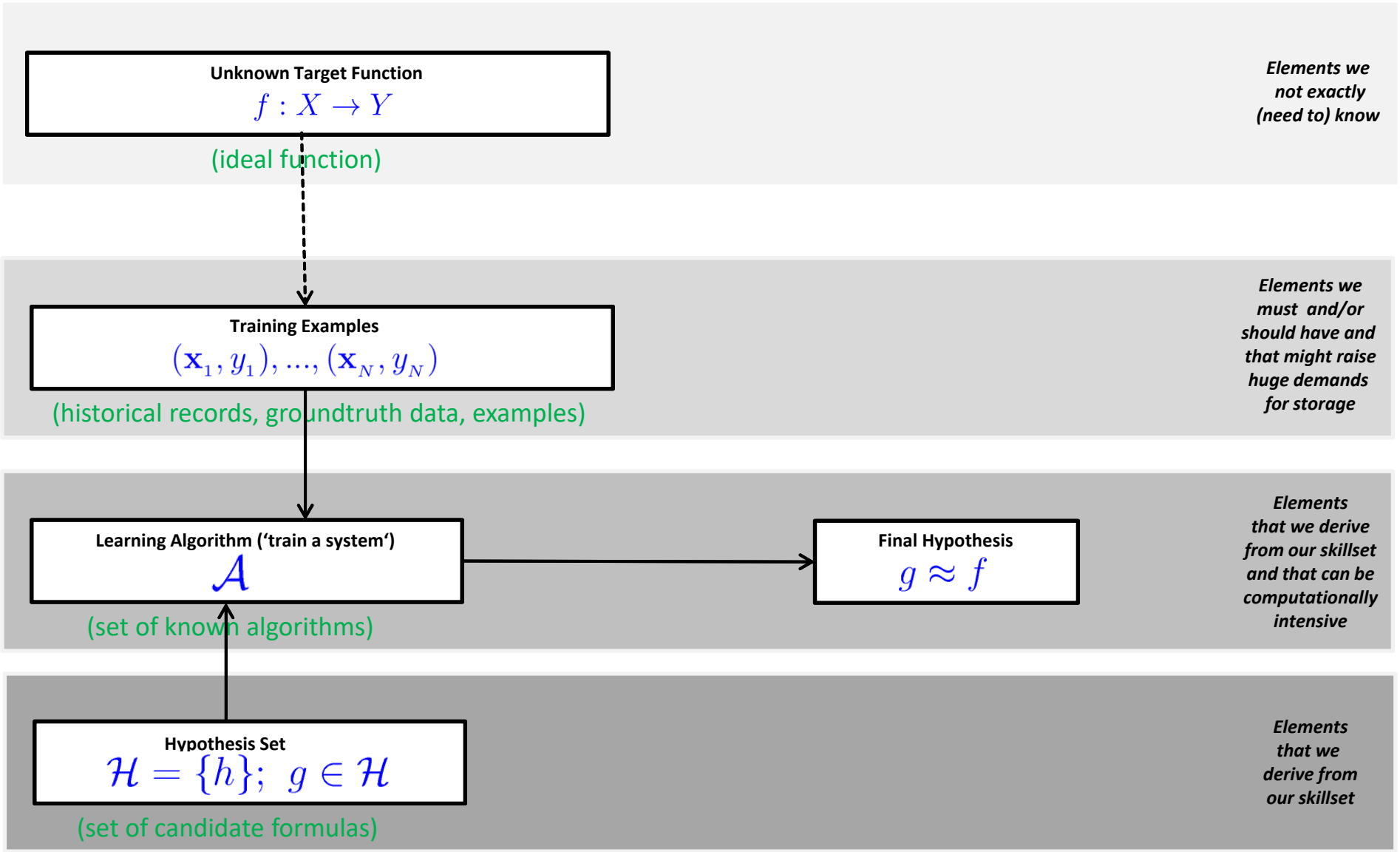
The Learning Model: Hypothesis Set & Learning Algorithm

- The solution tools – the learning model:
  1. Hypothesis set $\mathcal{H}$ - a set of candidate formulas /models
  2. Learning Algorithm $\mathcal{A}$ - 'train a system' with known algorithms

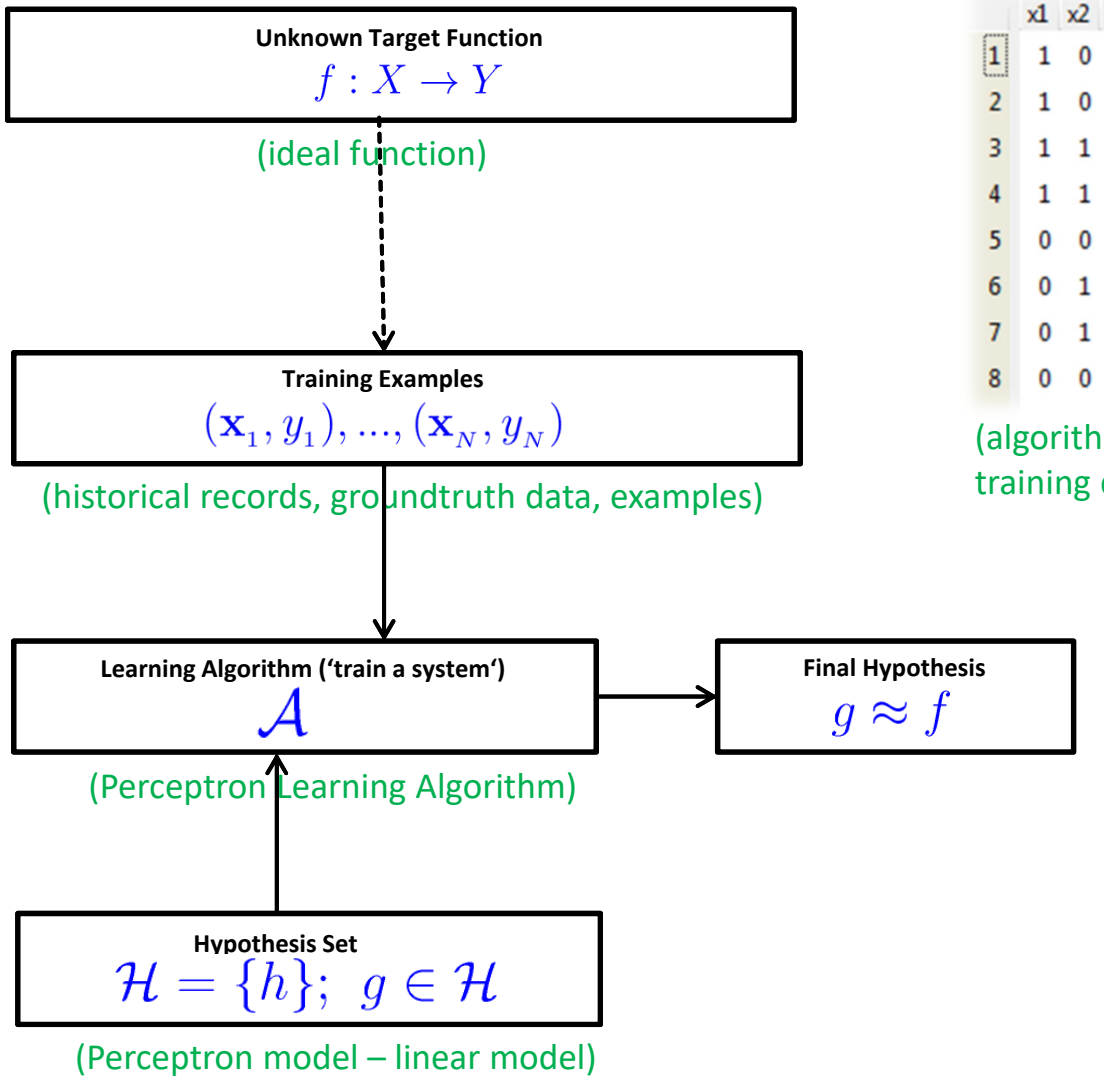


- Our Linear Example
  1. Perceptron Model
  2. Perceptron Learning Algorithm (PLA)

Mathematical Building Blocks (3)



Mathematical Building Blocks (3) – Our Linear Example



	x1	x2	x3	y
1	1	0	0	-1
2	1	0	1	1
3	1	1	0	1
4	1	1	1	1
5	0	0	1	-1
6	0	1	0	-1
7	0	1	1	1
8	0	0	0	-1

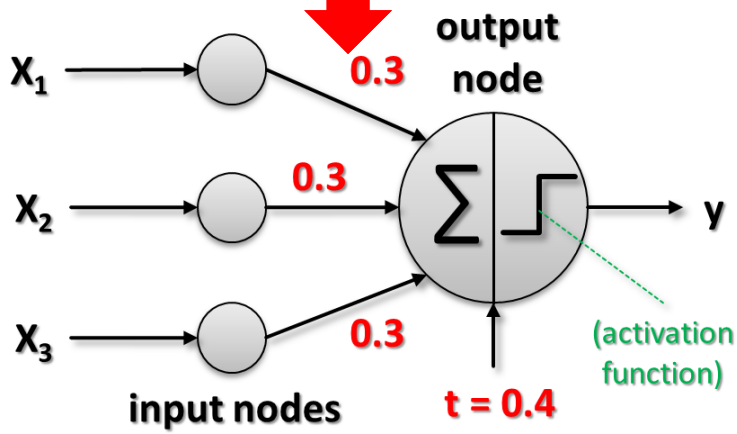
(algorithm uses training dataset)

$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$

(training data)

$$sign\left(\left(\sum_{i=1}^{d} w_i x_i\right) - threshold\right)$$

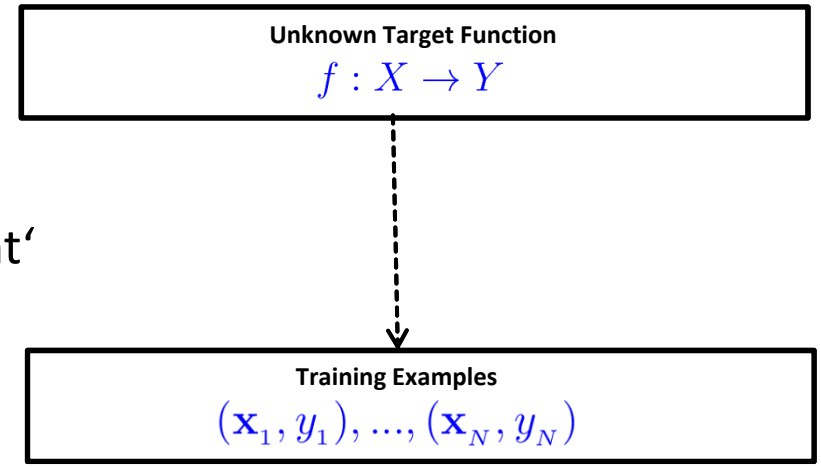
(training phase; Find w_i and threshold that fit the data)



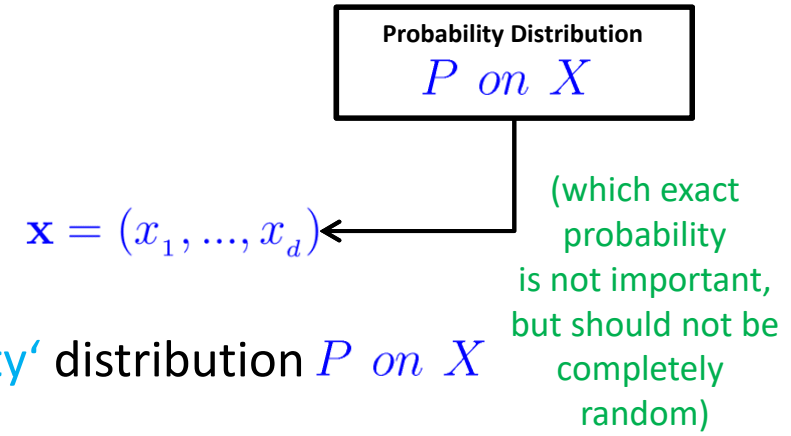
(trained perceptron model and our selected final hypothesis)

Feasibility of Learning – Probability Distribution

- Predict output from future input
  (fitting existing data is not enough)
  - In-sample ‘1000 points‘ fit well
  - Possible: Out-of-sample >= ‘1001 point‘ doesn‘t fit very well
  - Learning ‘any target function‘ is not feasible (can be anything)



- Assumptions about ‘future input‘
  - Statement is possible to define about the data outside the in-sample data $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)$
  - All samples (also future ones) are derived from same ‘unknown probability‘ distribution $P \ on \ X$

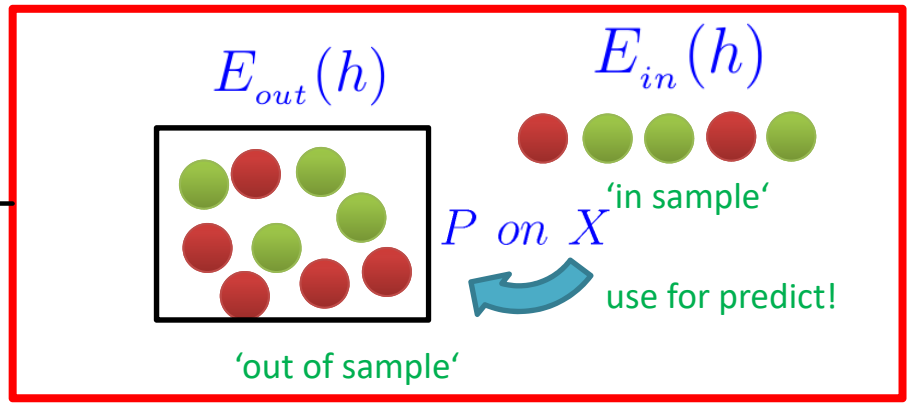


- **Statistical Learning Theory assumes an unknown probability distribution over the input space X**

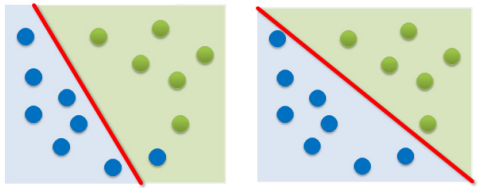
Feasibility of Learning – In Sample vs. Out of Sample

- Given ‘unknown‘ probability $P \ on \ X$
  - Given large sample N for $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)$
  - There is a probability of ‘picking one point or another‘ (i.e. from statistics)
  - ‘Error on in sample‘ is known quantity (using labelled data): $E_{in}(h)$
  - ‘Error on out of sample‘ is unknown quantity: $E_{out}(h)$
  - In-sample frequency is likely close to out-of-sample frequency E_{in} tracks E_{out}

depend on which hypothesis h out of m different ones



$\mathcal{H} = \{h_1, ..., h_m\};$



$$E_{in}(h) \approx E_{out}(h)$$

use $E_{in}(h)$ as a proxy – thus the other way around in learning

$$E_{out}(h) \approx E_{in}(h)$$

- **Statistical Learning Theory part that enables that learning is feasible in a probabilistic sense (P on X)**

Feasibility of Learning – Union Bound & Factor M

- **The union bound means that (for any countable set of m 'events') the probability that at least one of the events happens is not greater that the sum of the probabilities of the m individual 'events'**

- Assuming no overlaps in hypothesis set
  - Apply mathematical rule 'union bound' (i.e. poor bound)
  - Characterizes the number of data samples N needed

Final Hypothesis

$$g \approx f$$

Think if E_{in} deviates from E_{out} with more than tolerance Є it is a 'bad event' in order to apply union bound

$$\Pr\ [\ |\ E_{in}(g) - E_{out}(g)\ |\ > \epsilon\ ]\ <= \Pr\ [\ |\ E_{in}(h_1) - E_{out}(h_1)\ |\ > \epsilon$$

$$\text{or}\ \ |\ E_{in}(h_2) - E_{out}(h_2)\ |\ > \epsilon\ \ \ldots$$

$$\text{or}\ \ |\ E_{in}(h_M) - E_{out}(h_M)\ |\ > \epsilon\ \ ]$$

'visiting M different hypothesis'

$$\Pr\ [\ |\ E_{in}(g) - E_{out}(g)\ |\ > \epsilon\ ]\ <= \sum_{m=1}^{M} \Pr\ [\ |\ E_{in}(h_m) - E_{out}(h_m)\ |\ > \epsilon\ ]$$

sum of Pr is 'worst case' bound

$$\Pr\ [\ |\ E_{in}(g) - E_{out}(g)\ |\ > \epsilon\ ]\ <= \sum_{m=1}^{M} 2e^{-2\epsilon^2 N}$$

fixed quantity for each hypothesis obtained from Hoeffdings Inequality

$$\Pr\ [\ |\ E_{in}(g) - E_{out}(g)\ |\ > \epsilon\ ]\ <= 2Me^{-2\epsilon^2 N}$$

problematic: if M is too big we loose the link between the in-sample and out-of-sample

Feasibility of Learning – Modified Hoeffding's Inequality

- Errors in-sample $E_{in}(g)$ track errors out-of-sample $E_{out}(g)$
  - Statement is made being 'Probably Approximately Correct (PAC)'
  - Given M as number of hypothesis of hypothesis set $\mathcal{H}$
  - 'Tolerance parameter' in learning ϵ
  - Mathematically established via 'modified Hoeffdings Inequality':
    (original Hoeffdings Inequality doesn't apply to multiple hypothesis)

[14] Valiant, 'A Theory of the Learnable', 1984

'Approximately' 'Probably'

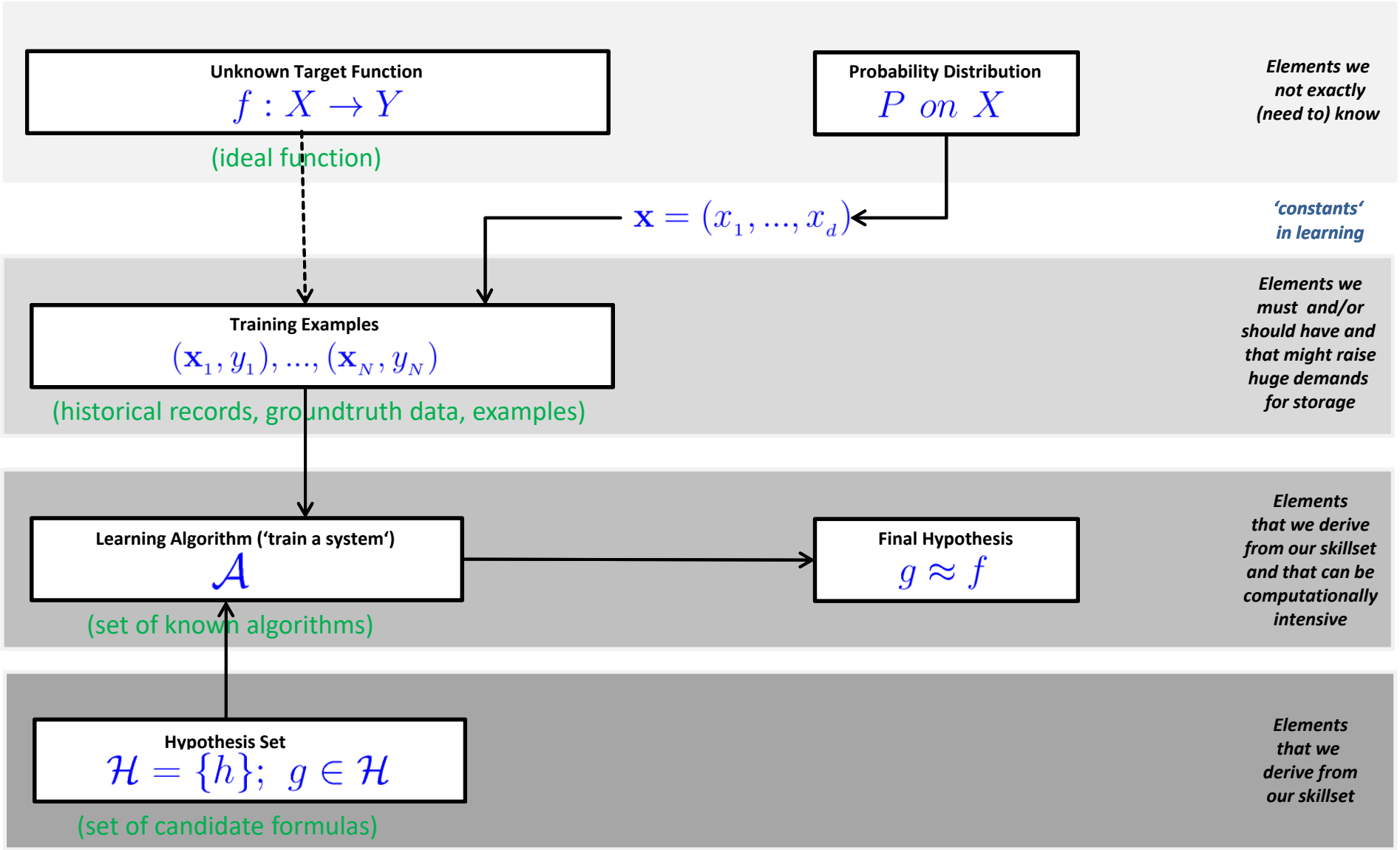
$$\Pr\ [\ |\ E_{in}(g) - E_{out}(g)\ |\ > \epsilon\ ]\ <=\ 2Me^{-2\epsilon^2 N}$$

'Probability that E_{in} deviates from E_{out} by more than the tolerance Є is a small quantity depending on M and N'

- Theoretical 'Big Data' Impact → more N → better learning
  - The more samples N the more reliable will track $E_{in}(g)\ E_{out}(g)$ well
  - (But: the 'quality of samples' also matter, not only the number of samples)

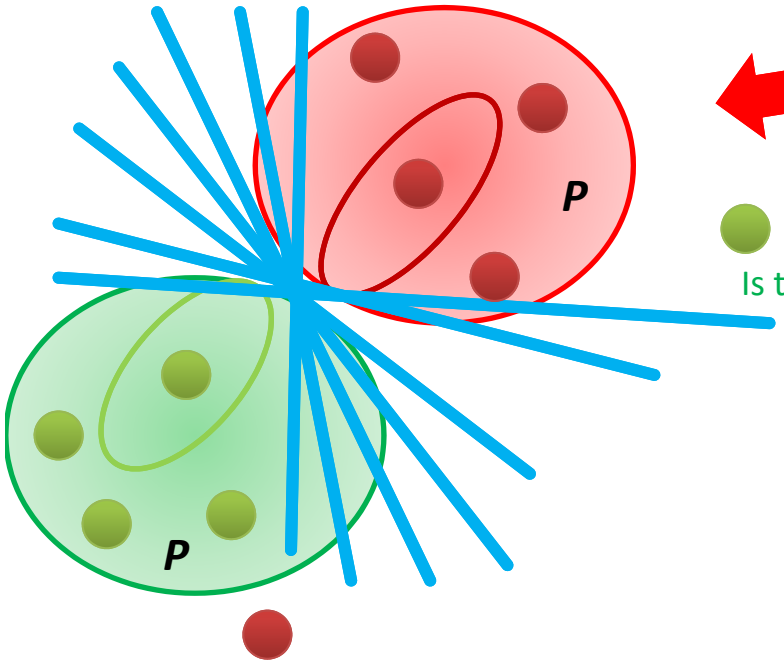
- **Statistical Learning Theory part describing the Probably Approximately Correct (PAC) learning**

Mathematical Building Blocks (4)



Mathematical Building Blocks (4) – Our Linear Example

(infinite M decision boundaries depending on f)



Probability Distribution

$$P \ on \ X$$

$$\mathbf{x} = (x_1, ..., x_d)$$

Is this point very likely from the same distribution or just noise?

We assume future points are taken from the same probability distribution as those that we have in our training examples

Training Examples

$$(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)$$

Is this point very likely from the same distribution or just noise?

(we help here with the assumption for the samples)

(we do not solve the M problem here)

$$\Pr\ [\ |\ E_{in}(g) - E_{out}(g)\ |\ > \epsilon\ ]\ <=\ 2Me^{-2\epsilon^2 N}$$

(counter example would be for instance a random number generator, impossible to learn this!)

Statistical Learning Theory – Error Measure & Noisy Targets

- Question: How can we learn a function from (noisy) data?
- ‘Error measures‘ to quantify our progress, the goal is: $h \approx f$
  - Often user-defined, if not often ‘squared error‘:

$$e(h(\mathbf{x}), f(\mathbf{x})) = (h(\mathbf{x}) - f(\mathbf{x}))^2$$

Error Measure α

  - E.g. ‘point-wise error measure‘ (e.g. think movie rated now and in 10 years from now)
- ‘(Noisy) Target function‘ is not a (deterministic) function
  - Getting with ‘same x in‘ the ‘same y out‘ is not always given in practice
  - Problem: ‘Noise‘ in the data that hinders us from learning
  - Idea: Use a ‘target distribution‘ instead of ‘target function‘
  - E.g. credit approval (yes/no)

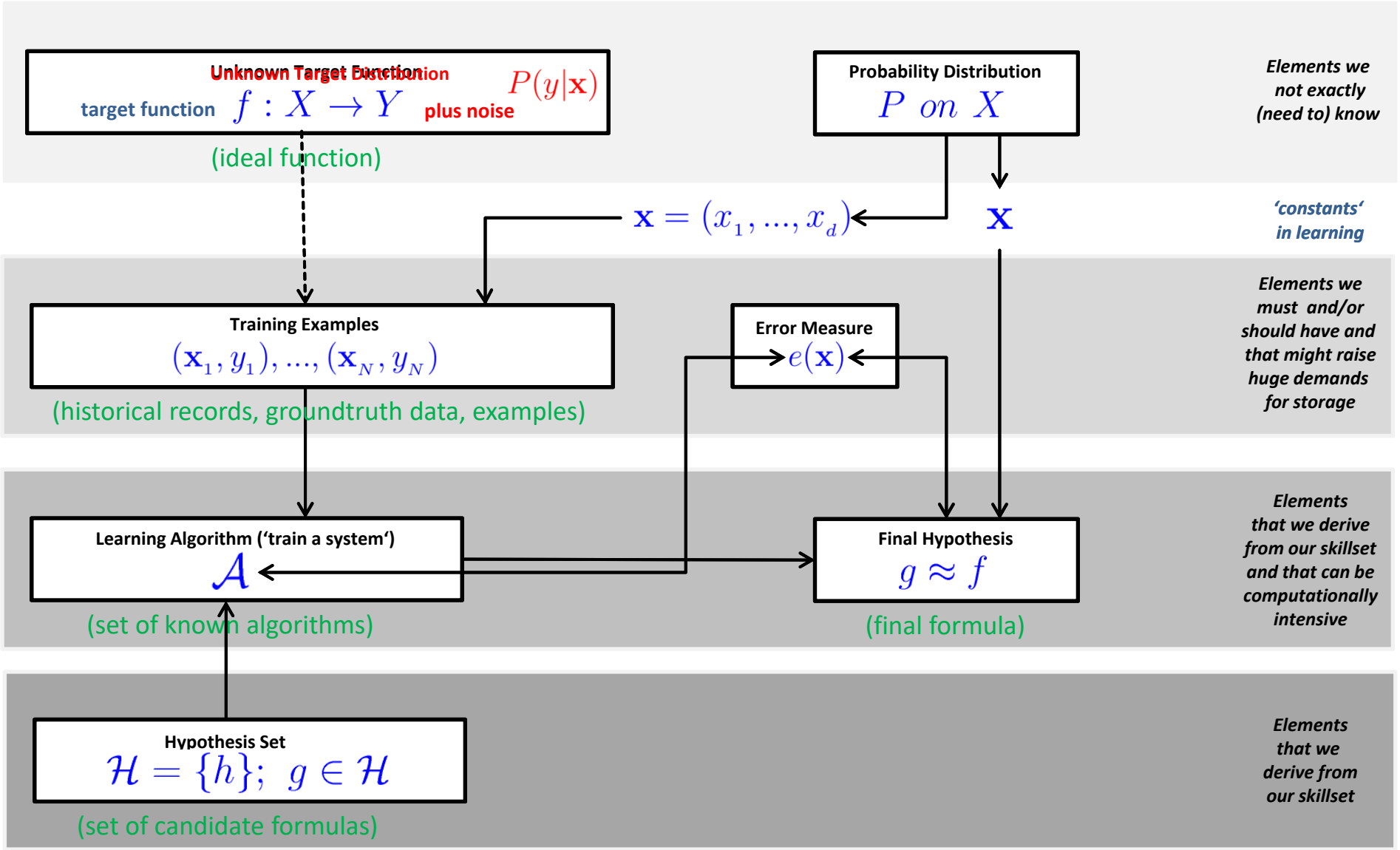
Unknown Target Distribution $P(y|\mathbf{x})$

target function $f : X \rightarrow Y$ plus noise

(ideal function)

- **Statistical Learning Theory refines the learning problem of learning an unknown target distribution**

Mathematical Building Blocks (5)



Mathematical Building Blocks (5) – Our Linear Example

- Iterative Method using (labelled) training data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$

(one point at a time is picked)

1. Pick one misclassified training point where:

$$sign(\mathbf{w}^T \mathbf{x}_n) \neq y_n$$

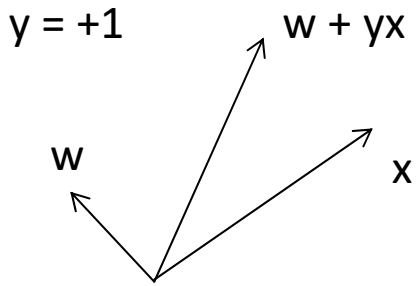
Error Measure α

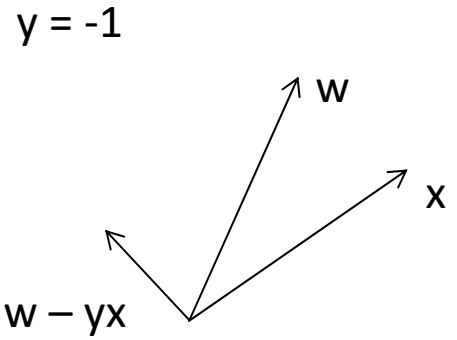
2. Update the weight vector:

$$\mathbf{w} \leftarrow \mathbf{w} + y_n \mathbf{x}_n$$

(y_n is either +1 or -1)

(a) adding a vector or
(b) subtracting a vector





- Terminates when there are no misclassified points

Error Measure α

(converges only with linearly seperable data)

Training and Testing – Influence on Learning

- Mathematical notations
  - Testing follows: (hypothesis clear)
    $$\Pr\ [\ |\ E_{in}(g) - E_{out}(g)\ |\ >\epsilon\ ]\ <=\ 2\quad e^{-2\epsilon^2 N}$$
  - Training follows: (hypothesis search)
    $$\Pr\ [\ |\ E_{in}(g) - E_{out}(g)\ |\ >\epsilon\ ]\ <=\ 2Me^{-2\epsilon^2 N}$$
    (e.g. student exam training on examples to get E_{in} ‚down', then test via exam)
- Practice on 'training examples'
  - Create two disjoint datasets
  - One used for training only (aka training set)
  - Another used for testing only (aka test set)

Training Examples

$$(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)$$

(historical records, groundtruth data, examples)

- Training & Testing are different phases in the learning process
  - Concrete number of samples in each set often influences learning

Theory of Generalization – Initial Generalization & Limits

- Learning is feasible in a probabilistic sense
  - Reported final hypothesis – using a 'generalization window' on $E_{out}(g)$
  - Expecting 'out of sample performance' tracks 'in sample performance'
  - Approach: $E_{in}(g)$ acts as a 'proxy' for $E_{out}(g)$

$$E_{out}(g) \approx E_{in}(g)$$

This is not full learning – rather 'good generalization' since the quantity $E_{out}(g)$ is an unknown quantity

- Reasoning
  - Above condition is not the final hypothesis condition:
  - More similiar like $E_{out}(g)$ approximates 0
    (out of sample error is close to 0 if approximating f)
  - $E_{out}(g)$ measures how far away the value is from the 'target function'
  - Problematic because $E_{out}(g)$ is an unknown quantity (cannot be used…)
  - The learning process thus requires 'two general core building blocks'

Final Hypothesis

$$g \approx f$$

Theory of Generalization – Learning Process Reviewed

- ‘Learning Well‘
  - Two core building blocks that achieve $E_{out}(g)$ approximates 0
- First core building block
  - Theoretical result using Hoeffdings Inequality $E_{out}(g) \approx E_{in}(g)$
  - Using $E_{out}(g)$ directly is not possible – it is an unknown quantity
- Second core building block (try to get the ‘in-sample‘ error lower)
  - Practical result using tools & techniques to get $E_{in}(g) \approx 0$
  - e.g. linear models with the Perceptron Learning Algorithm (PLA)
  - Using $E_{in}(g)$ is possible – it is a known quantity – ‘so lets get it small‘
  - Lessons learned from practice: in many situations ‘close to 0‘ impossible
  - E.g. remote sensing images use case of land cover classification

- **Full learning means that we can make sure that $E_{out}(g)$ is close enough to $E_{in}(g)$ [from theory]**
- **Full learning means that we can make sure that $E_{in}(g)$ is small enough [from practical techniques]**

Complexity of the Hypothesis Set – Infinite Spaces Problem

$$\Pr\ [\ |\ E_{in}(g) - E_{out}(g)\ |\ > \epsilon\ ]\ <=\ 2Me^{-2\epsilon^2 N}$$

theory helps to find a way to deal with infinite M hypothesis spaces

- Tradeoff & Review
  - Tradeoff between Є, M, and the 'complexity of the hypothesis space H'
  - Contribution of detailed learning theory is to 'understand factor M'
- M Elements of the hypothesis set $\mathcal{H}$ M elements in H here
  - Ok if N gets big, but problematic if M gets big → bound gets meaningless
  - E.g. classification models like perceptron, support vector machines, etc.
  - Challenge: those classification models have continous parameters
  - Consequence: those classification models have infinite hypothesis spaces
  - Aproach: despite their size, the models still have limited expressive power

- **Many elements of the hypothesis set H have continous parameter with infinite M hypothesis spaces**

Factor M from the Union Bound & Hypothesis Overlaps

$$\Pr\ [\ |\ E_{in}(g) - E_{out}(g)\ |\ > \epsilon\ ]\ <= \Pr\ [\ |\ E_{in}(h_1) - E_{out}(h_1)\ |\ > \epsilon$$
$$\text{or}\ \ |\ E_{in}(h_2) - E_{out}(h_2)\ |\ > \epsilon\ \ldots$$
$$\text{or}\ \ |\ E_{in}(h_M) - E_{out}(h_M)\ |\ > \epsilon\ ]$$

assumes no overlaps, all probabilities happen disjointly

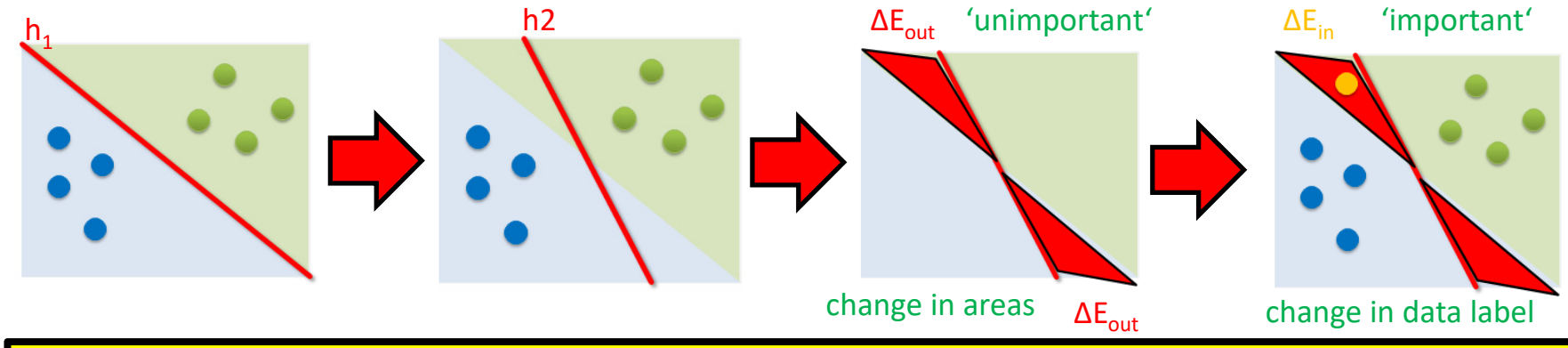
$$\Pr\ [\ |\ E_{in}(g) - E_{out}(g)\ |\ > \epsilon\ ]\ <=\ 2Me^{-2\epsilon^2 N}$$

takes no overlaps of M hypothesis into account

- Union bound is a 'poor bound', ignores correlation between h
  - Overlaps are common: the interest is shifted to data points changing label

$$|\ E_{in}(h_1) - E_{out}(h_1)\ | \approx\ \ |\ E_{in}(h_2) - E_{out}(h_2)\ |$$

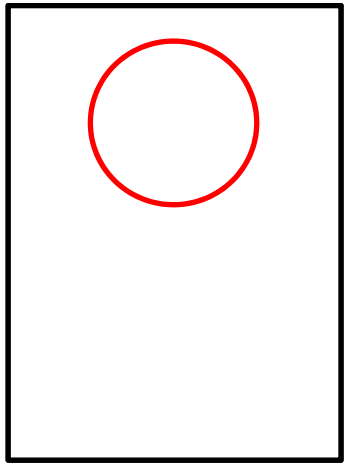
(at least very often, indicator to reduce M)



- **Statistical Learning Theory provides a quantity able to characterize the overlaps for a better bound**

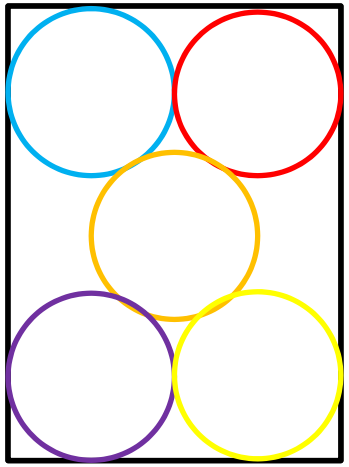
Replacing M & Large Overlaps

(Hoeffding Inequality)



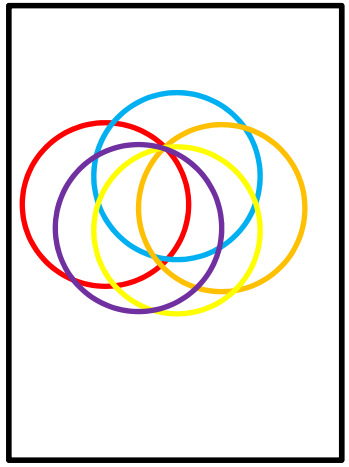
(valid for 1 hypothesis)

(Union Bound)



(valid for M hypothesis, worst case)

(towards Vapnik Chervonenkis Bound)



(valid for m (N) as growth function)

- Characterizing the overlaps is the idea of a 'growth function'
  - Number of dichotomies: $\mathbf{m}_{\mathcal{H}}(N) = \max_{\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_N} |\mathcal{H}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)|$
    Number of hypothesis but on finite number N of points
  - Much redundancy: Many hypothesis will reports the same dichotomies

- **The mathematical proofs that $m_H(N)$ can replace M is a key part of the theory of generalization**

Complexity of the Hypothesis Set – VC Inequality

$$\Pr\ [\ |\ E_{in}(g) - E_{out}(g)\ |\ >\ \epsilon\ ]\ <=\ 2Me^{-2\epsilon^2 N}$$

$$\mathbf{m}_{\mathcal{H}}(N) = \max_{\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_N} |\mathcal{H}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)|$$

- Vapnik-Chervonenkis (VC) Inequality
  - Result of mathematical proof when replacing M with growth function m
  - 2N of growth function to have another sample (2 x $E_{in}(h)$, no $E_{out}(h)$)

$$\Pr\ [\ |\ E_{in}(g) - E_{out}(g)\ |\ >\ \epsilon\ ]\ <=\ 4m_{\mathcal{H}}(2N)e^{-1/8\epsilon^2 N}$$

(characterization of generalization)

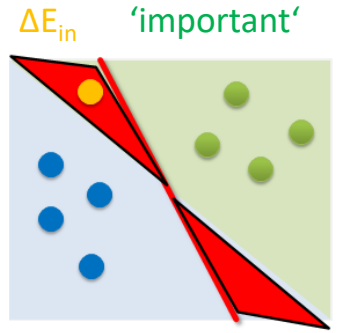
Important for bound: $m_h(N)$ is polynomial in N

  - In Short – finally : We are able to learn and can generalize ‘ouf-of-sample‘

- **The Vapnik-Chervonenkis Inequality is the most important result in machine learning theory**
- **The mathematial proof brings us that M can be replaced by growth function (no infinity anymore)**
- **The bound changes thus from ‘infinity with M‘ to a realistic bound that we can work with: max 2^N**

‘Growth Function‘ – Perceptron Example

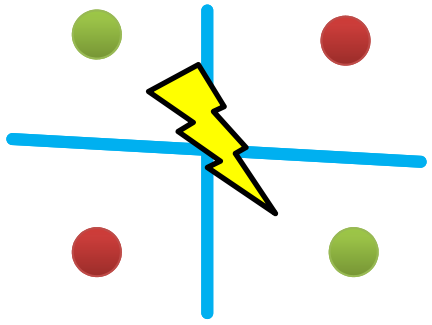
- Dichotomies
  - Hypothesis set seperates data, but only change important
  - Set of ‘mini-hyposthesis‘ is restricted to finite data points N
  - Number of mini-hypothesis = number of dichotomies



- ‘Growth Function‘
  - Based on the number of dichotomies (cardiality)

$$\mathbf{m}_{\mathcal{H}}(N) = \max_{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N} |\mathcal{H}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)|$$

  - Pick $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ wisely to maximise the dichotomies (# at most 2^N)
- 2D Perceptron
  - Practice: restriction on dichotomies means less mini-hypothesis possible (less than 2^N)
  - E.g. for N = 4 points, there is always a pattern that can not be realized



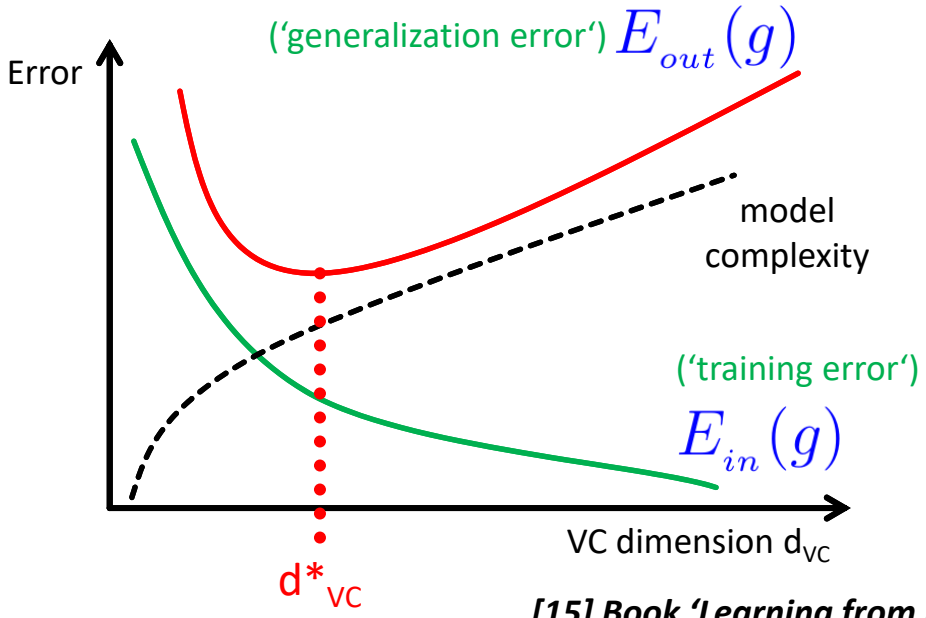
(breakpoint k = 4)

[15] Book ‘Learning from Data’

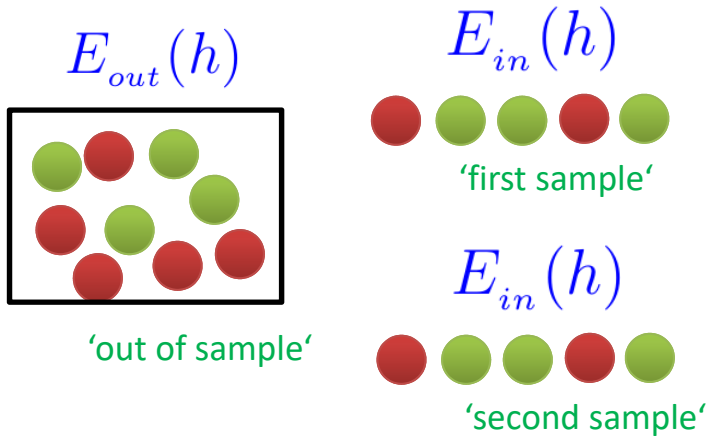
Towards Complexity of the Hypothesis Set – VC Dimension

- Vapnik-Chervonenkis (VC) Dimension over instance space X
  - VC dimension gets a 'generalization bound' on all possible target functions
  - Practice: think how much model parameters ('degrees of freedom')

Issue: unknown to 'compute' – VC solved this using the growth function on different samples



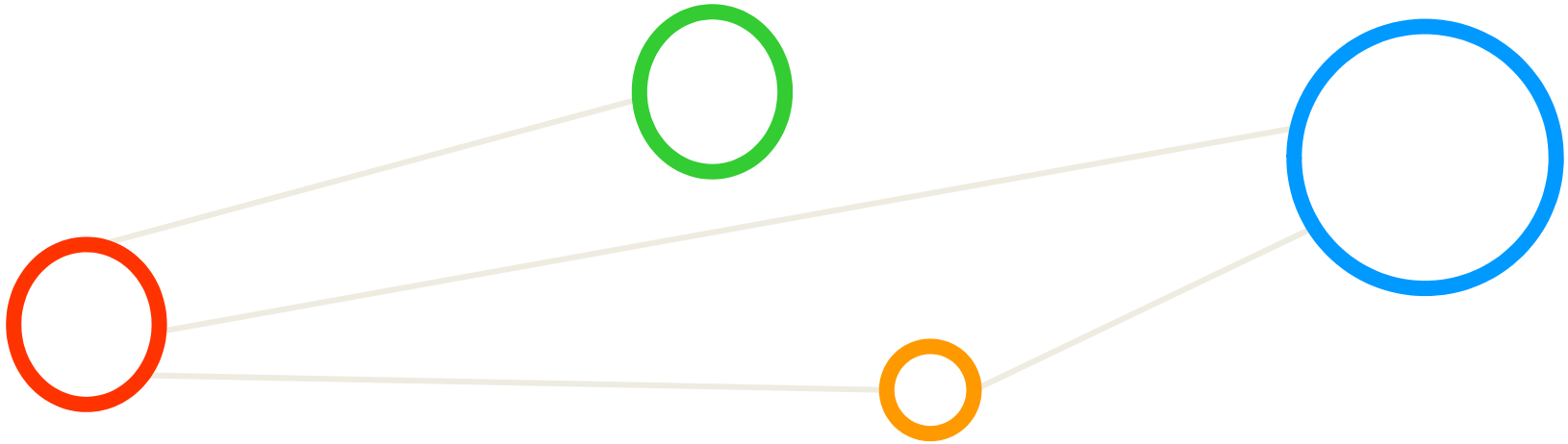
[15] Book 'Learning from Data'



idea: 'first sample' frequency close to 'second sample' frequency

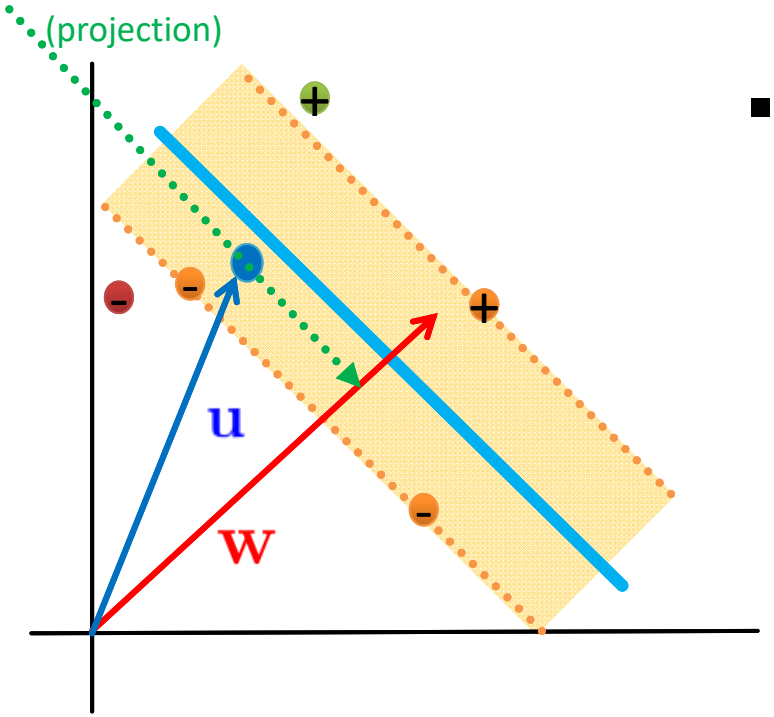
- **Complexity of Hypothesis set H can be measured by the Vapnik-Chervonenkis (VC) Dimension d_{VC}**
- **Ignoring the model complexity d_{VC} leads to situations where $E_{in}(g)$ gets down and $E_{out}(g)$ gets up**

Appendix C: Support Vector Machines (SVMs)



Geometric SVM Interpretation and Setup (1)

- Think ‘simplified coordinate system‘ and use ‘Linear Algebra‘
  - Many other samples are removed (red and green not SVs)
  - Vector $\mathbf{w}$ of ‘any length‘ perpendicular to the decision boundary
  - Vector $\mathbf{u}$ points to an unknown quantity (e.g. new sample to classify)
  - Is $\mathbf{u}$ on the left or right side of the decision boundary?



- Dot product $\mathbf{w} \cdot \mathbf{u} \geq C; C = -b$
  - With $\mathbf{u}$ takes the projection on the $\mathbf{w}$
  - Depending on where projection is it is left or right from the decision boundary
  - Simple transformation brings decison rule:

(1) $\mathbf{w} \cdot \mathbf{u} + b \geq 0$ → means +

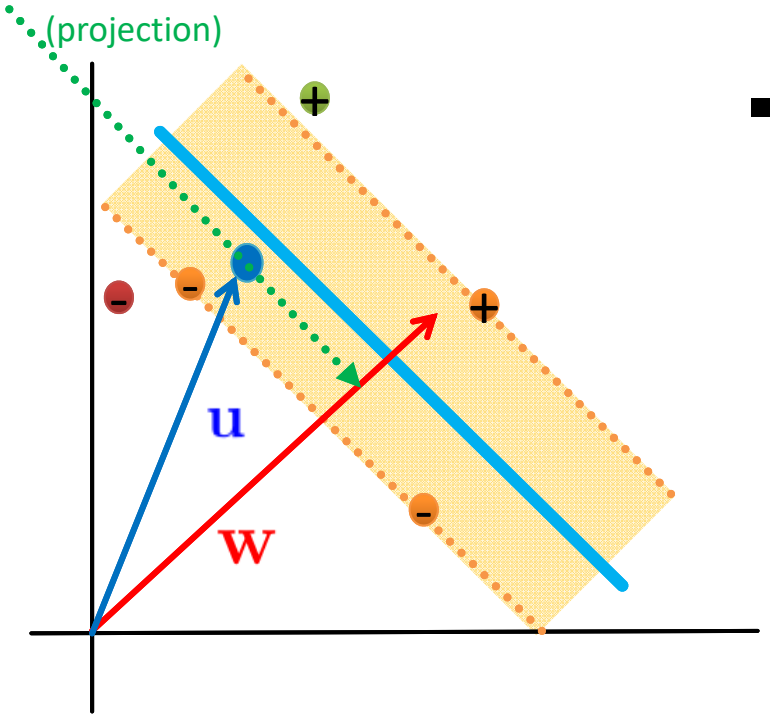
  - (given that b and $\mathbf{w}$ are unknown to us)

(constraints are not enough to fix particular b or w, need more constraints to calculate b or w)

Geometric SVM Interpretation and Setup (2)

- Creating our constraints to get b or $\mathbf{w}$ computed
  - First constraint set for positive samples ⊕ $\mathbf{w} \cdot \mathbf{x}_+ + b \geq 1$
  - Second constraint set for negative samples ⊖ $\mathbf{w} \cdot \mathbf{x}_- + b \leq 1$
  - For mathematical convenience introduce variables (i.e. labelled samples)

$y_i = +$ for ⊕ and $y_i = -$ for ⊖



- Multiply equations by y_i
  - Positive samples: $y_i(\mathbf{x}_i \cdot \mathbf{w} + b) \geq 1$
  - Negative samples: $y_i(\mathbf{x}_i \cdot \mathbf{w} + b) \geq 1$
  - Both same due to $y_i = +$ and $y_i = -$

(brings us mathematical convenience often quoted)

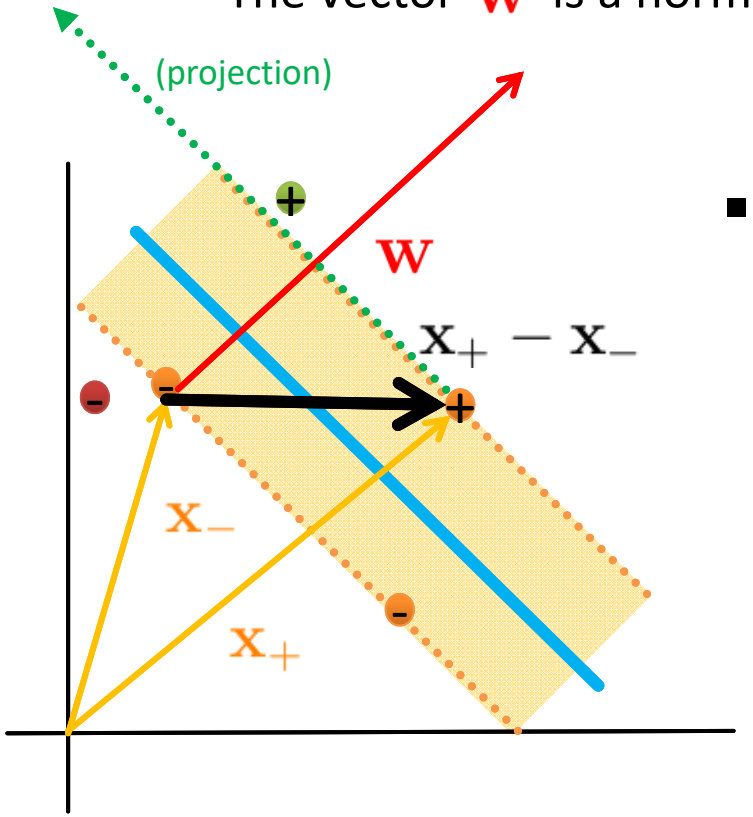
$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \geq 0$$

(additional constraints just for support vectors itself helps)

(2) $y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 = 0$

Geometric SVM Interpretation and Setup (3)

- Determine the ‘width of the margin‘
  - Difference between positive and negative SVs: $\mathbf{x}_+ - \mathbf{x}_-$
  - Projection of $\mathbf{x}_+ - \mathbf{x}_-$ onto the vector $\mathbf{w}$
  - The vector $\mathbf{w}$ is a normal vector, magnitude is $\|\mathbf{w}\|$



(Dot product of two vectors is a scalar, here the width of the margin)

- Unit vector is helpful for ‘margin width‘
  - Projection (dot product) for margin width:

$$\mathbf{x}_+ - \mathbf{x}_- \cdot \frac{\mathbf{w}}{\|\mathbf{w}\|}$$ (unit vector)

$1 - b \quad 1 + b$

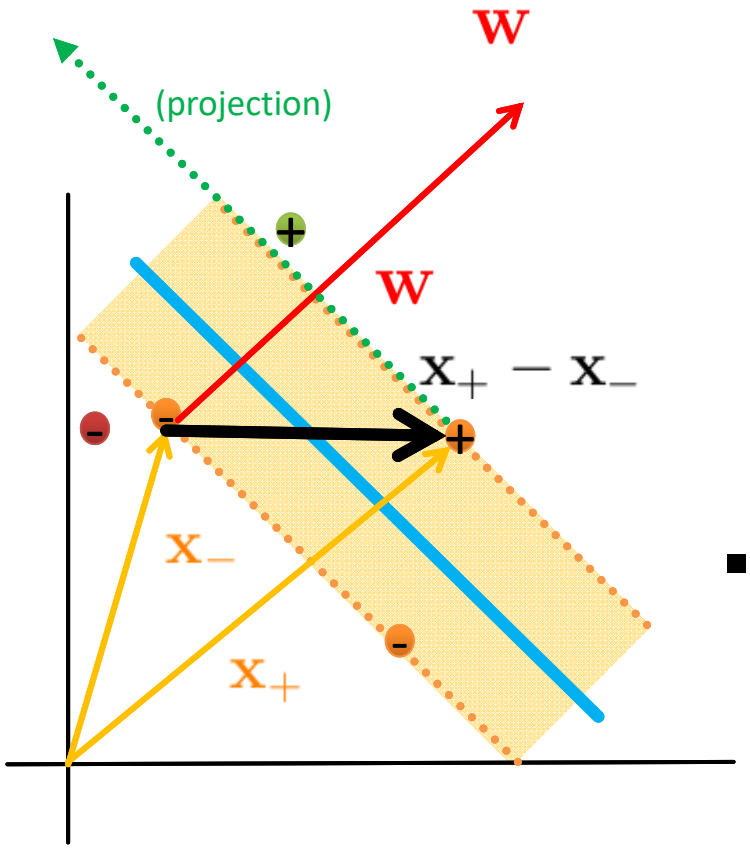
$$\frac{2}{\|\mathbf{w}\|}$$ (3)

  - When enforce constraint: $y_i = +$ $y_i = -$

(2) $y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 = 0$

Constrained Optimization Steps SVM (1)

- Use ‘constraint optimization‘ of mathematical toolkit
  - Idea is to ‘maximize the width‘ of the margin: $max \frac{2}{\|\mathbf{w}\|}$ (drop the constant 2 is possible here)



$\Rightarrow$ $max \frac{1}{\|\mathbf{w}\|}$ (equivalent)

$\Rightarrow$ $min \|\mathbf{w}\|$ (equivalent for max)

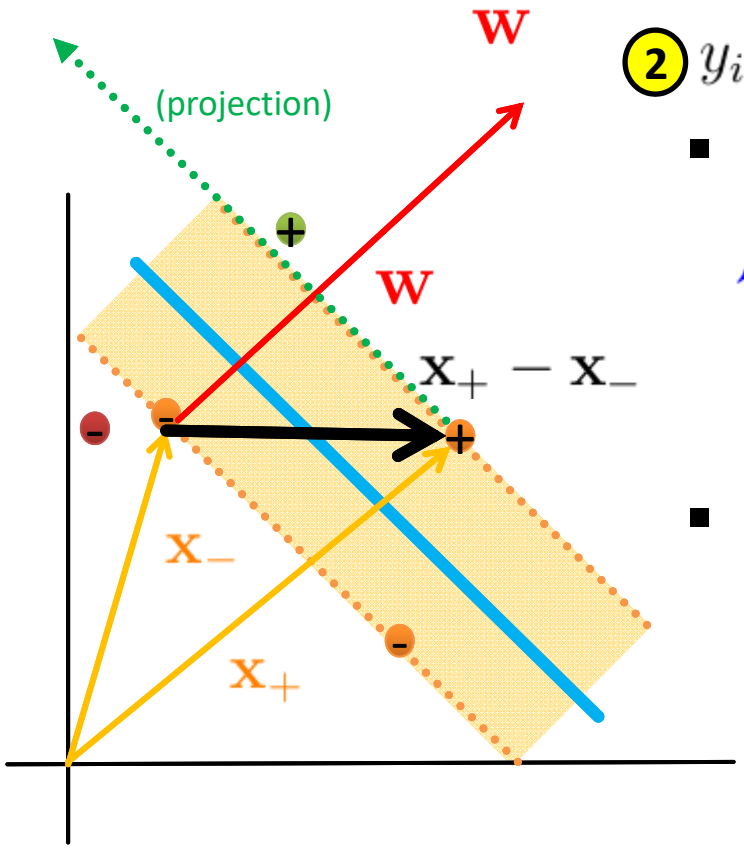
$\Rightarrow$ $min \frac{1}{2} \|\mathbf{w}\|^2$ (mathematical convenience) (3)

- Next: Find the extreme values
  - Subject to constraints

(2) $y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 = 0$

Constrained Optimization Steps SVM (2)

- Use ‘Lagrange Multipliers‘ of mathematical toolkit
  - Established tool in ‘constrained optimization‘ to find function extremum
  - ‘Get rid‘ of constraints by using Lagrange Multipliers (4)



(2) $y_i(\mathbf{x}_i \cdot \mathbf{w} + b - 1) = 0$

- Introduce a multiplier for each constraint

$$\mathcal{L}(\alpha) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum \alpha_i [y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1]$$

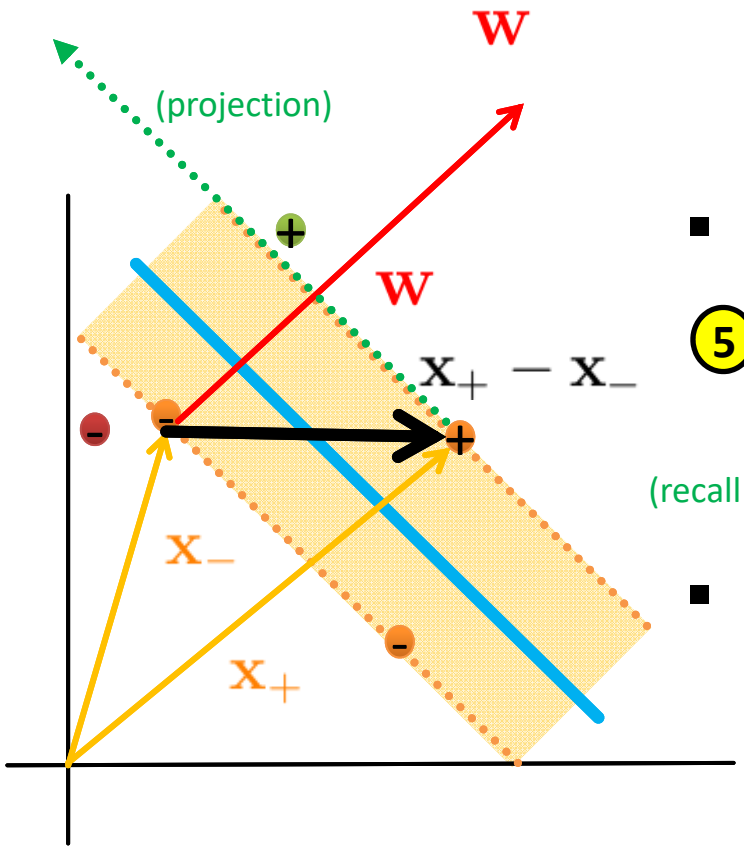
(interesting: non zero for support vectors, rest zero)

- Find derivatives for extremum & set 0
  - But two unknowns that might vary
  - First differentiate w.r.t. $\mathbf{w}$
  - Second differentiate w.r.t. b

(derivative gives the gradient, setting 0 means extremum like min)

Constrained Optimization Steps SVM (3)

- Lagrange gives: $\mathcal{L}(\alpha) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum \alpha_i [y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1]$



- First differentiate w.r.t $\mathbf{w}$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \mathbf{w} - \sum \alpha_i y_i \mathbf{x}_i = 0$$

(derivative gives the gradient, setting 0 means extremum like min)

- Simple transformation brings:

(5) $$\mathbf{w} = \sum \alpha_i y_i \mathbf{x}_i$$

(i.e. vector is linear sum of samples)

(recall: non zero for support vectors, rest zero → even less samples)

- Second differentiate w.r.t. b

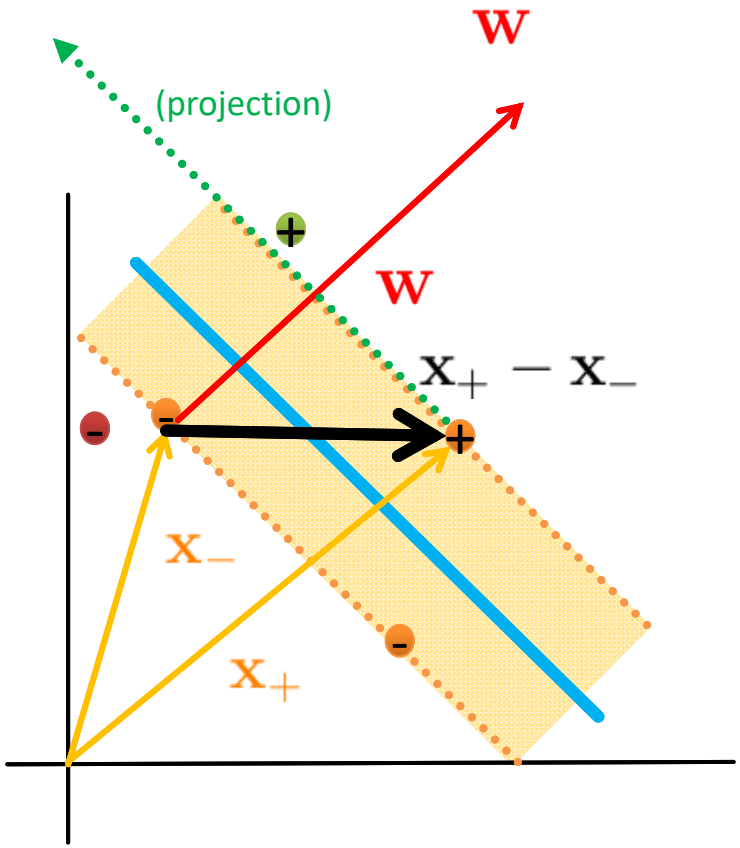
$$\frac{\partial \mathcal{L}}{\partial b} = -\sum \alpha_i y_i = 0 \Rightarrow \sum \alpha_i y_i = 0$$ (5)

Constrained Optimization Steps SVM (4)

- Lagrange gives: $\mathcal{L}(\alpha) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum \alpha_i [y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1]$

(plug into)

  - Find minimum
    - Quadratic optimization problem
    - Take advantage of (5) $\mathbf{w} = \sum \alpha_i y_i \mathbf{x}_i$



$$\mathcal{L} = \frac{1}{2}\left(\sum \alpha_i y_i \mathbf{x}_i\right) \cdot \left(\sum \alpha_j y_j \mathbf{x}_j\right)$$

$$- \sum \alpha_i y_i \mathbf{x}_i \cdot \left(\sum \alpha_j y_j \mathbf{x}_j\right)$$

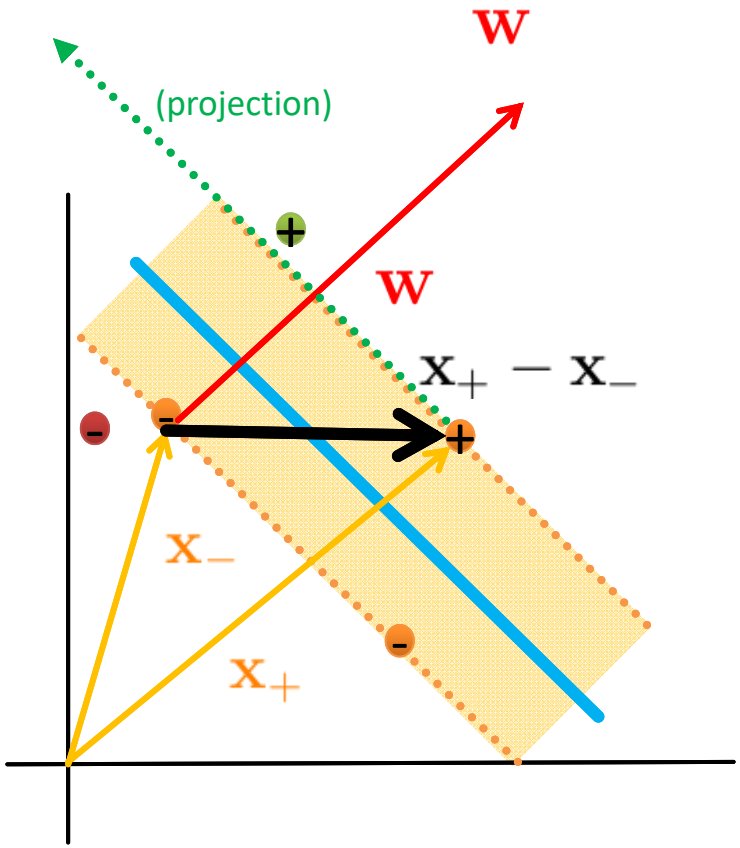
$$- \sum \alpha_i y_i b + \sum \alpha_i$$

(b constant in front sum)

(5) $$\sum \alpha_i y_i = 0$$

Constrained Optimization Steps SVM (5)

- Rewrite formula: $\mathcal{L} = \frac{1}{2}\left(\sum \alpha_i y_i \mathbf{x}_i\right) \cdot \left(\sum \alpha_j y_j \mathbf{x}_j\right) - \sum \alpha_i y_i \mathbf{x}_i \cdot \left(\sum \alpha_j y_j \mathbf{x}_j\right)$ (the same)



$$-\sum \alpha_i y_i b + \sum \alpha_i$$ (was 0)

(results in)

(optimization depends only on dot product of samples)

$$\mathcal{L} = \sum \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \quad (6)$$

- Equation to be solved by some quadratic programming package

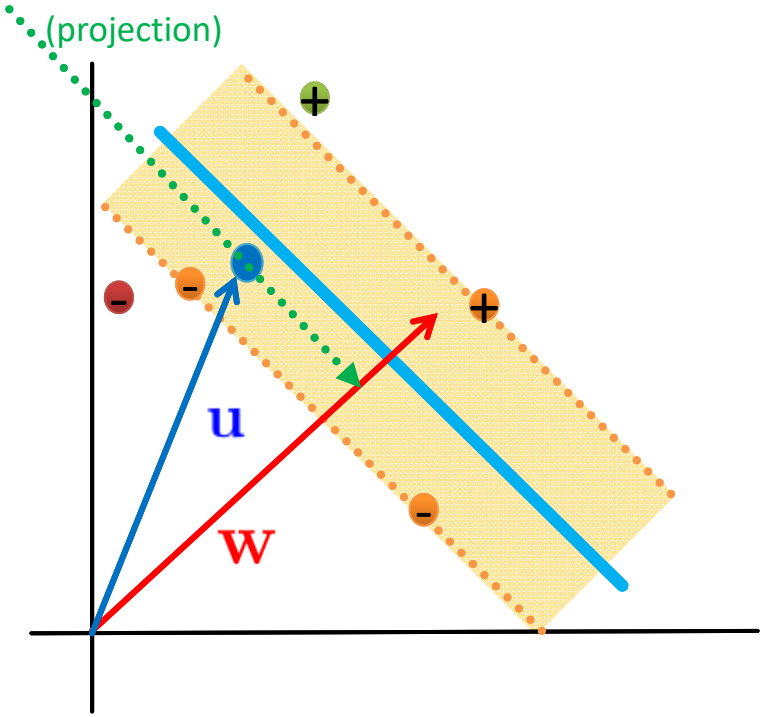
Use of SVM Classifier to Perform Classification

- Use findings for decision rule

(5) $\mathbf{w} = \sum \alpha_i y_i \mathbf{x}_i$

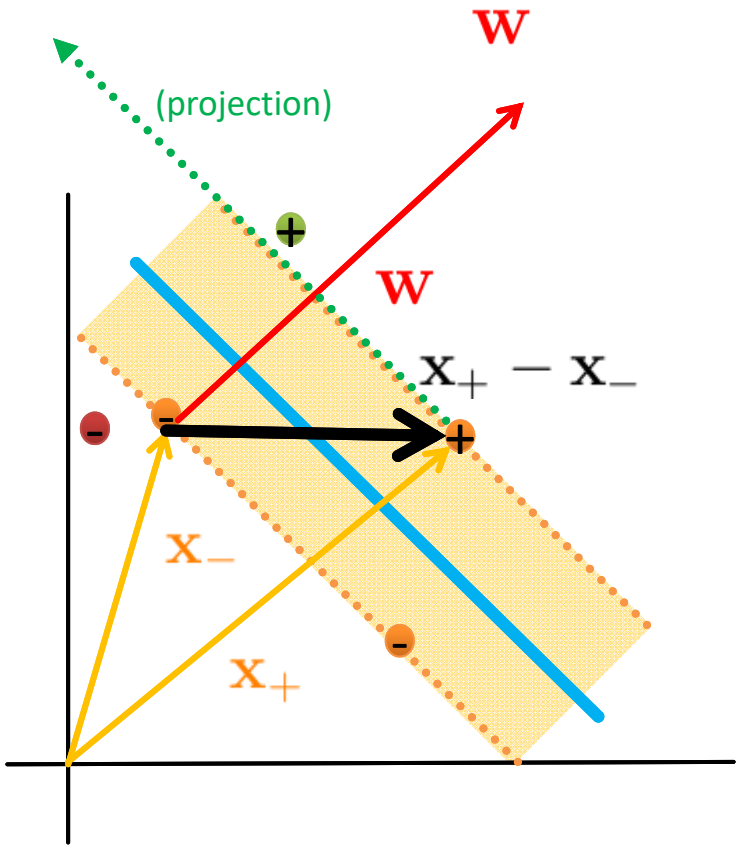
(1) $\mathbf{w} \cdot \mathbf{u} + b \geq 0$ ⊕ ⇒ $\sum \alpha_i y_i \mathbf{x}_i \cdot \mathbf{u}_i + b \geq 0$ ⊕

(decision rule also depends on dotproduct)



Constrained Optimization Steps SVM & Dot Product

- Rewrite formula: $\mathcal{L} = \frac{1}{2}\left(\sum \alpha_i y_i \mathbf{x}_i\right) \cdot \left(\sum \alpha_j y_j \mathbf{x}_j\right) - \sum \alpha_i y_i \mathbf{x}_i \cdot \left(\sum \alpha_j y_j \mathbf{x}_j\right)$ (the same)



$$-\sum \alpha_i y_i b + \sum \alpha_i$$ (was 0)

(results in)

(optimization depends only on dot product of samples)

$$\mathcal{L} = \sum \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \quad (6)$$

- Equation to be solved by some quadratic programming package

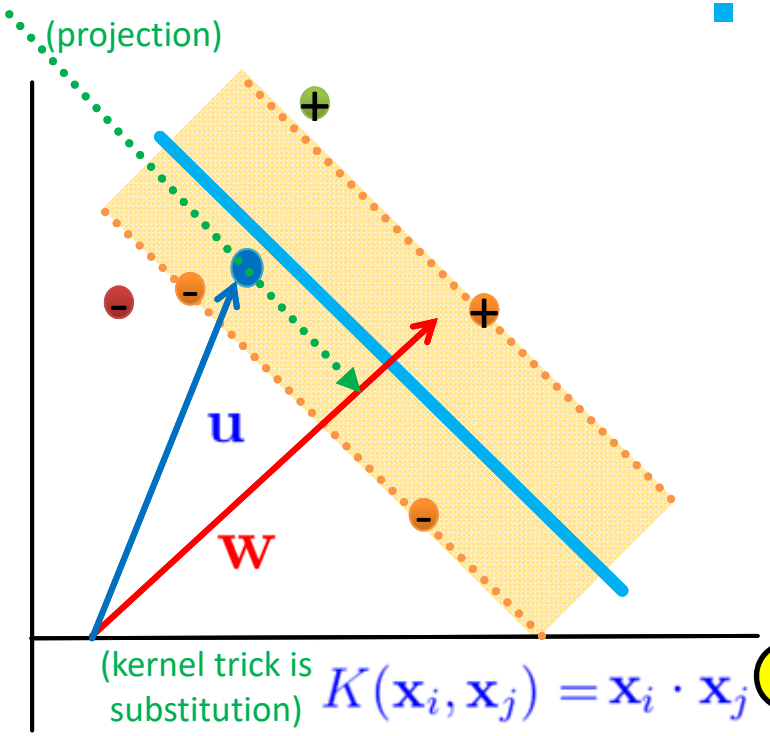
Kernel Methods & Dot Product Dependency

- Use findings for decision rule

(5) $\mathbf{w} = \sum \alpha_i y_i \mathbf{x}_i$

(1) $\mathbf{w} \cdot \mathbf{u} + b \geq 0$ ⊕ ⇒ $\sum \alpha_i y_i \mathbf{x}_i \cdot \mathbf{u}_i + b \geq 0$ ⊕

(decision rule also depends on dotproduct)



- Dotproduct enables nice more elements
  - E.g. consider non linearly seperable data
  - Perform non-linear transformation Φ of the samples into another space (work on features)

$$\mathcal{L} = \sum \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$$ (6)

(optimization depends only on dot product of samples)

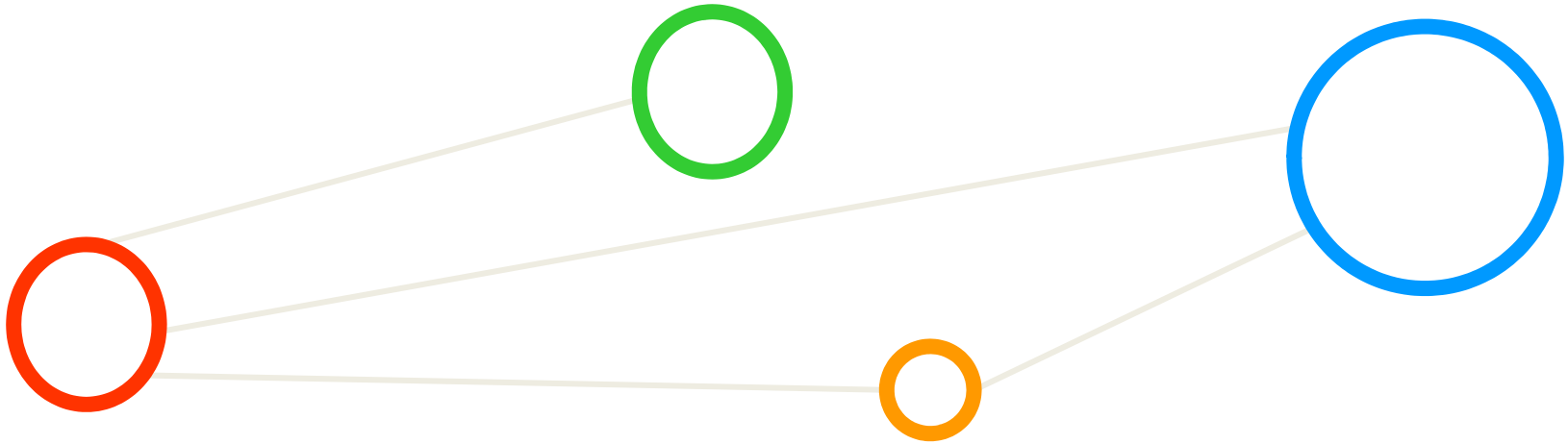
⇒ $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$ (in optimization)

⇒ $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{u}_i)$ (for decision rule above too)

(kernel trick is substitution) $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$ (7) $K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$

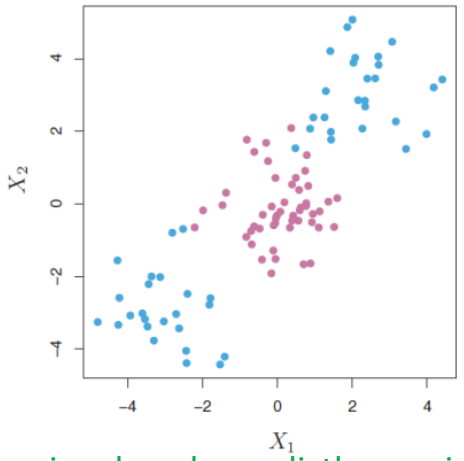
(trusted Kernel avoids to know Phi)

Appendix D: Kernel Methods



Need for Non-linear Decision Boundaries

- Lessons learned from practice
  - Scientists and engineers are often faced with non-linear class boundaries
- Non-linear transformations approach
  - Enlarge feature space (computationally intensive)
  - Use quadratic, cubic, or higher-order polynomial functions of the predictors
- Example with Support Vector Classifier



(time invest: mapping done by explictly carrying out the map into the feature space)

$X_1, X_2, \ldots, X_p$ (previously used p features)

$X_1, X_1^2, X_2, X_2^2, \ldots, X_p, X_p^2$ (new 2p features)

(decision boundary is linear in the enlarged feature space)

(decision boundary is non-linear in the original feature space with q(x) = 0 where q is a quadratic polynomial)

$$\underset{\beta_0,\beta_{11},\beta_{12}\ldots,\beta_{p1},\beta_{p2},\epsilon_1,\ldots,\epsilon_n}{\text{maximize}} \quad M$$

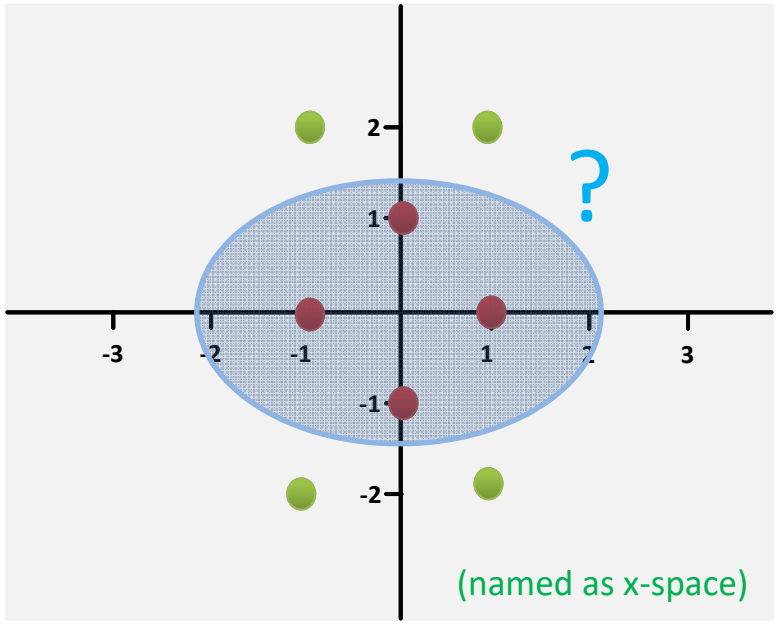
$$\text{subject to } y_i\left(\beta_0 + \sum_{j=1}^{p}\beta_{j1}x_{ij} + \sum_{j=1}^{p}\beta_{j2}x_{ij}^2\right) \geq M(1-\epsilon_i)$$

$$\sum_{i=1}^{n}\epsilon_i \leq C, \quad \epsilon_i \geq 0, \quad \sum_{j=1}^{p}\sum_{k=1}^{2}\beta_{jk}^2 = 1$$

[6] An Introduction to Statistical Learning

Understanding Non-Linear Transformations (1)

- Example: ‘Use measure of distances from the origin/centre‘
- Classification (still linear models applicable)
  - (1) new point; (2) transform to z-space; (3) classify it with e.g. perceptron

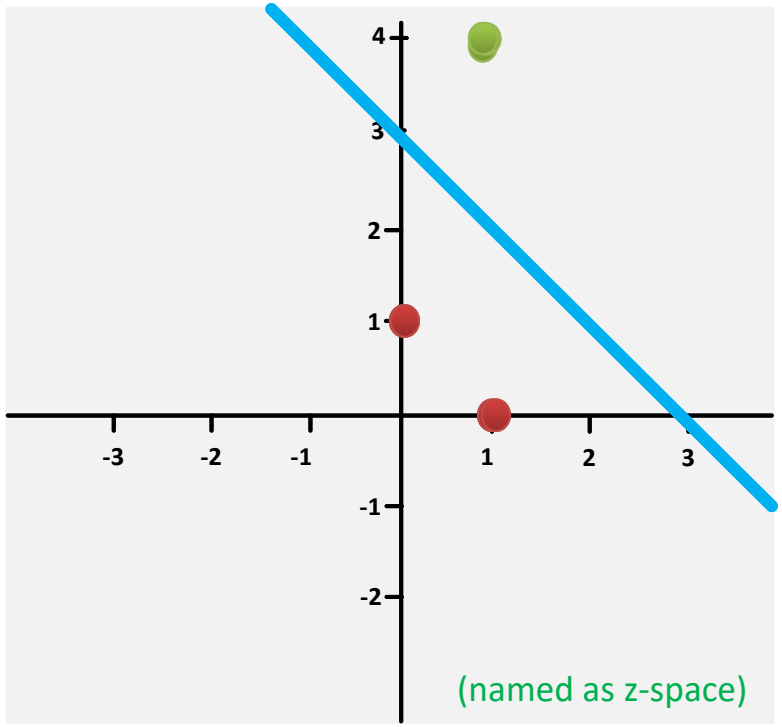


(named as x-space)

(also called input space)

Φ

(‘changing constants‘)

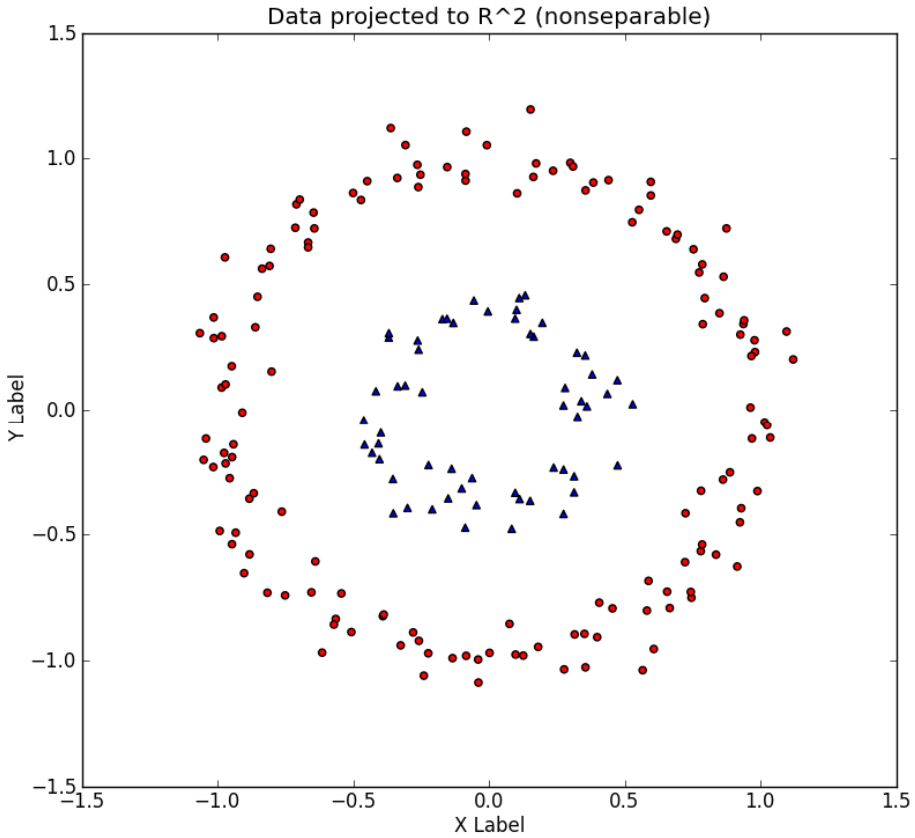


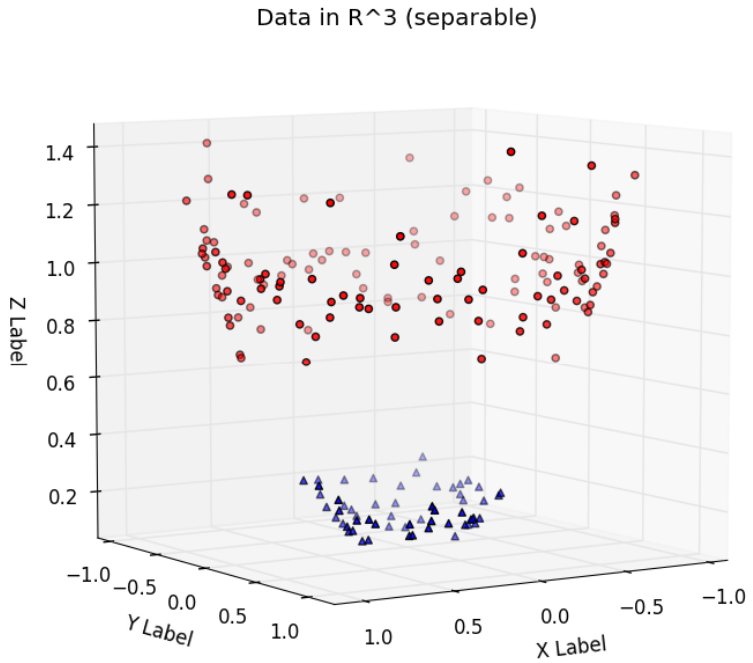
(named as z-space)

(also called feature space)

Understanding Non-Linear Transformations (2)

- Example: From 2 dimensional to 3 dimensional: $[x_1, x_2] = [x_1, x_2, {x_1}^2 + {x_2}^2]$
  - Much higher dimensional can cause memory and computing problems



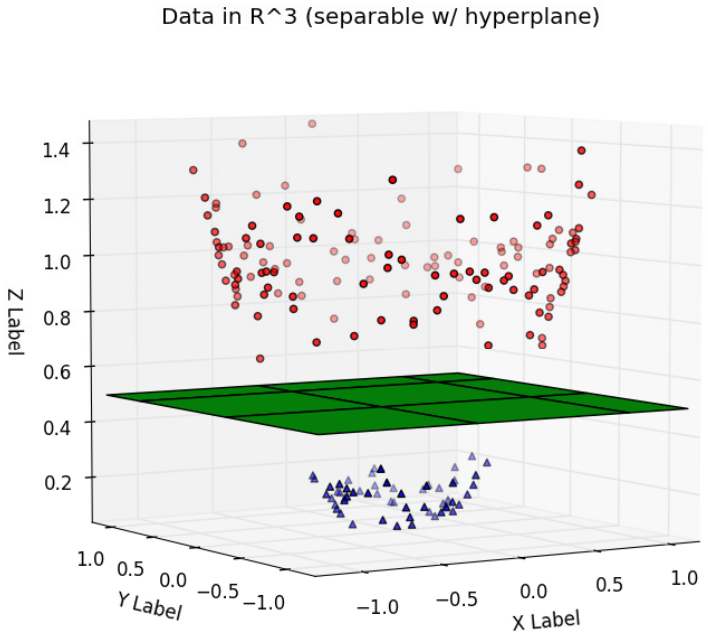


[20] E. Kim

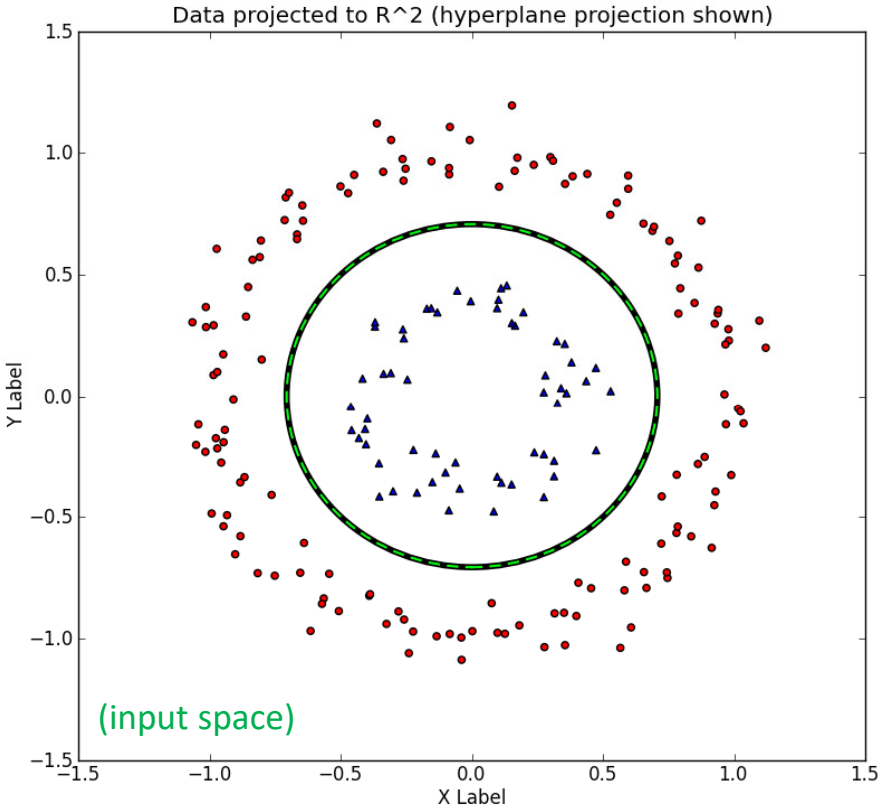
- **Problems: Not clear which type of mapping (search); optimization is computationally expensive task**

Understanding Non-linear Transformations (3)

- Example: From 2 dimensional to 3 dimensional: $[x_1, x_2] = [x_1, x_2, {x_1}^2 + {x_2}^2]$
  - Separating hyperplane can be found and ‘mapped back‘ to input space



(feature space)



[6] E. Kim

- **Problem: ‘curse of dimensionality’ – As dimensionality increases & volume of space too: sparse data!**

Term Support Vector Machines – Revisited

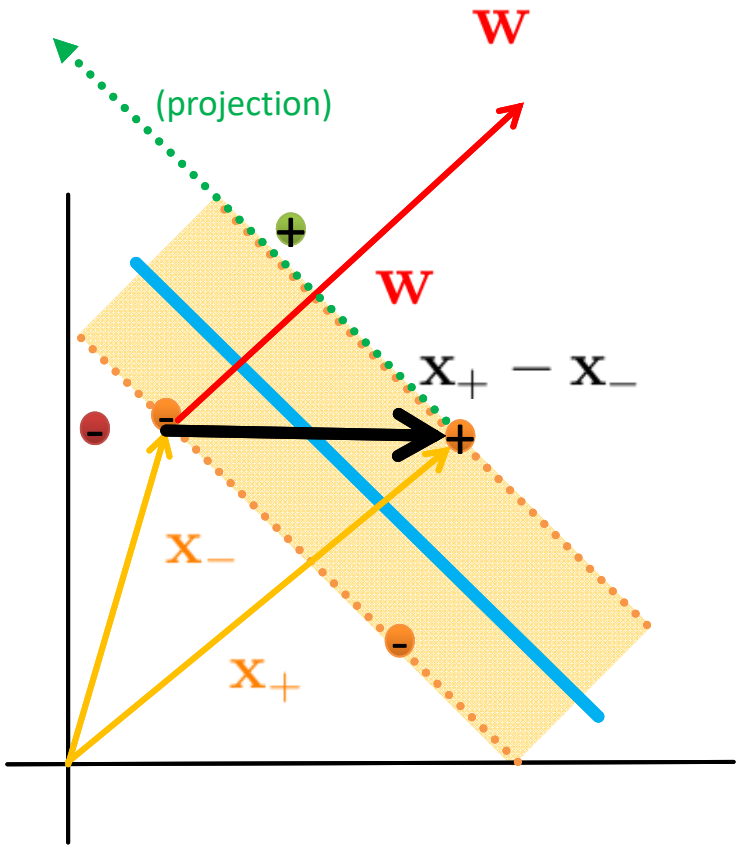
- **Support Vector Machines (SVMs) are a classification technique developed ~1990**
- **SVMs perform well in many settings & are considered as one of the best ‘out of the box classifiers‘**

[6] An Introduction to Statistical Learning

- Term detailed refinement into ‘three separate techniques’
  - Practice: applications mostly use the SVMs with kernel methods
- ‘Maximal margin classifier‘
  - A simple and intuitive classifier with a ‘best‘ linear class boundary
  - Requires that data is ‘linearly separable‘
- ‘Support Vector Classifier‘
  - Extension to the maximal margin classifier for non-linearly seperable data
  - Applied to a broader range of cases, idea of ‘allowing some error‘
- ‘Support Vector Machines‘ → Using Non-Linear Kernel Methods
  - Extension of the support vector classifier
  - Enables non-linear class boundaries & via kernels;

Constrained Optimization Steps SVM & Dot Product

- Rewrite formula: $\mathcal{L} = \frac{1}{2}\left(\sum \alpha_i y_i \mathbf{x}_i\right) \cdot \left(\sum \alpha_j y_j \mathbf{x}_j\right) - \sum \alpha_i y_i \mathbf{x}_i \cdot \left(\sum \alpha_j y_j \mathbf{x}_j\right)$ (the same)



$$-\sum \alpha_i y_i b + \sum \alpha_i$$ (was 0)

(results in)

(optimization depends only on dot product of samples)

$$\mathcal{L} = \sum \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \quad (6)$$

- Equation to be solved by some quadratic programming package

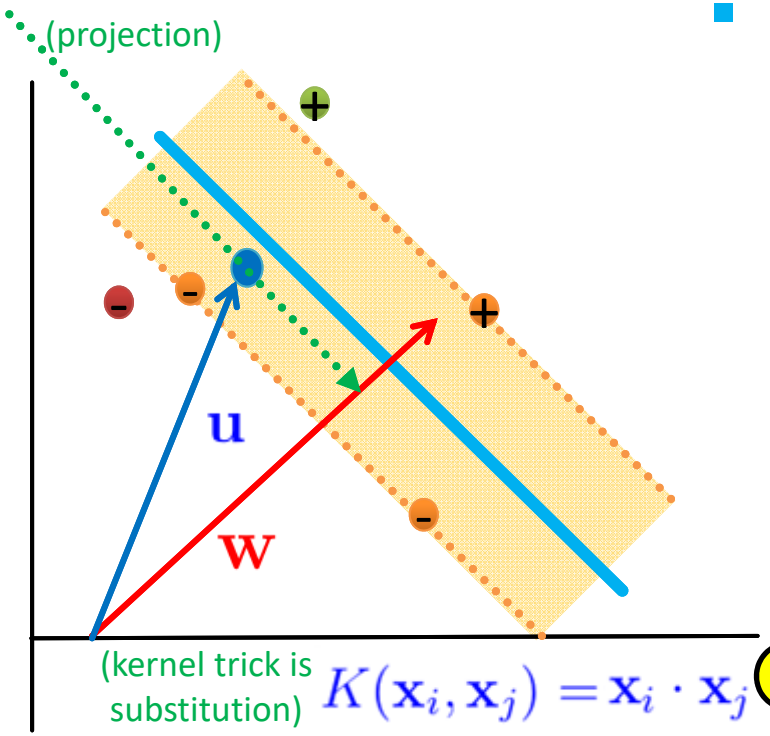
Kernel Methods & Dot Product Dependency

- Use findings for decision rule

(5) $\mathbf{w} = \sum \alpha_i y_i \mathbf{x}_i$

(1) $\mathbf{w} \cdot \mathbf{u} + b \geq 0$ ⊕ ⇒ $\sum \alpha_i y_i \mathbf{x}_i \cdot \mathbf{u}_i + b \geq 0$ ⊕

(decision rule also depends on dotproduct)



- Dotproduct enables nice more elements
  - E.g. consider non linearly seperable data
  - Perform non-linear transformation Φ of the samples into another space (work on features)

$$\mathcal{L} = \sum \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$$ (6)

(optimization depends only on dot product of samples)

$\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$ (in optimization)

$\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{u}_i)$ (for decision rule above too)

(kernel trick is substitution) $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$

(7) $K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$

(trusted Kernel avoids to know Phi)

Support Vector Machines & Kernel Methods

- **Support Vector Machines are extensions of the support vector classifier using kernel methods**
- **Support Vector Machines enable non-linear decision boundaries that can be efficiently computed**
- **Support Vector Machines avoids 'curse of dimensionality' and mapping search using a 'kernel trick'**

- Non-linear transformations

[6] An Introduction to Statistical Learning

  - Lead to high number of features → Computations become unmanageable
    (including the danger to run into 'curse of dimensionality')
- Benefits with SVMs
  - Enlarge feature space using 'kernel trick' → ensures efficient computations
  - Map training data into a higher-dimensional feature space using Φ
  - Create a seperating 'hyperplane' with maximum margin in feature space
- Solve constraint optimization problem
  - Using Lagrange multipliers & quadratic programming (cf. earlier classifiers)
  - Solution involves the inner products of the data points (dot products)
  - Inner product of two r-vectors a and b is defined as $\langle a, b \rangle = \sum_{i=1}^{r} a_i b_i$
  - Inner product of two data points: $\langle x_i, x_{i'} \rangle = \sum_{j=1}^{p} x_{ij} x_{i'j}$

Linear SV Classifier Refined & Role of SVs

- Linear support vector classifier
  - Details w.r.t. inner products
  - With n parameters $\alpha_i, \quad i = 1, \ldots, n$ (Lagrange multipliers)
  - Use training set to estimate parameters
  - Estimate $\alpha_1, \ldots, \alpha_n$ and β_0 using $\binom{n}{2}$ inner products $\langle x_i, x_{i'} \rangle$

$$\langle x_i, x_{i'} \rangle = \sum_{j=1}^{p} x_{ij} x_{i'j}$$

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle$$

(between all pairs of training data points)

n (n – 1) / 2 number of pairs

- Evaluate $f(x)$ with a new point
  - Compute the inner product between new point x and each of the training points x_i
- Identify support vectors → Quadratic programming
  - α_i is zero most of the times (identified as not support vectors)
  - α_i is nonzero several times (identified as the support vectors)

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle$$

'big data' reduction & less computing

$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \alpha_i \langle x, x_i \rangle$$

(S with indices of support vectors)

[6] An Introduction to Statistical Learning

The ('Trusted') Kernel Trick

- Summary for computation
  - All that is needed to compute coefficients are inner products
- Kernel Trick
  - Replace the inner product with a generalization of the inner product
  - K is some kernel function
- Kernel types
  - Linear kernel
  - Polynomial kernel

(sompute the hyperplane without explictly carrying out the map into the feature space)

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle$$

(inner product used before)

$$\langle x_i, x_{i'} \rangle = \sum_{j=1}^{p} x_{ij} x_{i'j}$$

(kernel ~ distance measure)

$$K(x_i, x_{i'})$$

(choosing a specific kernel type)

$$K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j}$$ (linear in features)

$$K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p} x_{ij} x_{i'j})^d$$ (polynomial of degree d)

[6] An Introduction to Statistical Learning

- **Kernel trick refers to a mechanism of using different kernel functions (e.g. polynomial)**
- **A kernel is a function that quantifies the similarity of two data points (e.g. close to each other)**

Kernel Trick – Example

- Consider again a simple two dimensional dataset
  - We found an ideal mapping Φ after long search
  - Then we need to transform the whole dataset according to Φ

$$\Phi : (x_1, x_2) \mapsto (x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, 1)$$

  - Instead, with the ‘kernel trick‘ we ‘wait‘ and ‘let the kernel do the job‘:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \Rightarrow min_\alpha \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) - \sum \alpha_i$$

(no need to compute the mapping already)

$$\Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) = (u_1^2, u_2^2, \sqrt{2}u_1, \sqrt{2}u_2, 1) \cdot (v_1^2, v_2^2, \sqrt{2}v_1, \sqrt{2}v_2, 1)$$

$$\Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) = u_1^2 v_1^2 + u_2^2 v_2^2 + 2u_1 v_1 + 2u_2 v_2 + 1$$

$$\Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) = (\mathbf{u} \cdot \mathbf{v} + 1)^2 = K(\mathbf{u}, \mathbf{v})$$

(we can save computing time by do not perform the mapping)

(in transformed space still a dot product in the original space → no mapping needed)

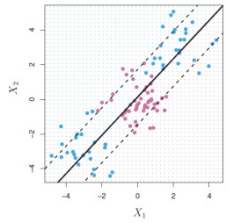
- **Example shows that the dot product in the transformed space can be expressed in terms of a similarity function in the original space (here dot product is a similiarity between two vectors)**

Linear vs. Polynomial Kernel Example

- Linear kernel
  - Enables linear decision boundaries (i.e. like linear support vector classifier)

$$K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j}$$ (linear in features)

(observed useless for non-linear data)

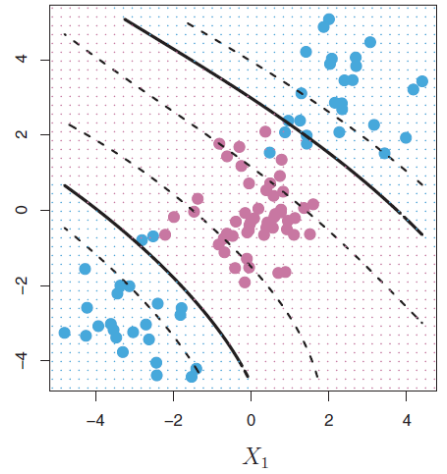


- Polynomial kernel
  - Satisfy Mercer's theorem = trusted kernel
  - Enables non-linear decision boundaries (when choosing degree d > 1)
  - Amounts to fit a support vector classifier in a higher-dimensional space
  - Using polynomials of degree d (d=1 linear support vector classifier)

$$K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p} x_{ij} x_{i'j})^d$$ (polynomial of degree d)

(SVM with polynomial kernel of degree 3)

(significantly improved decision rule due to much more flexible decision boundary)

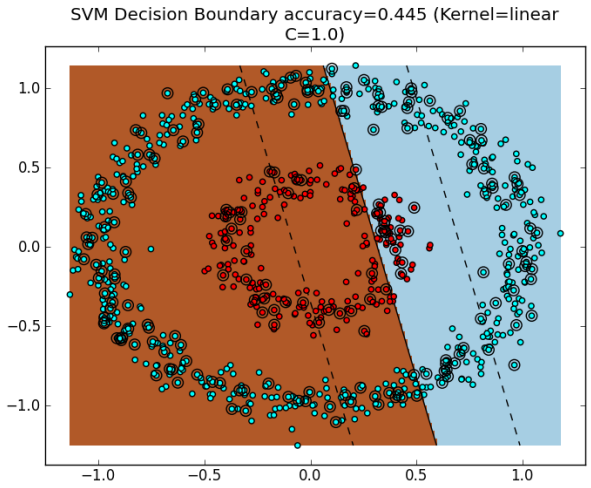


[6] An Introduction to Statistical Learning

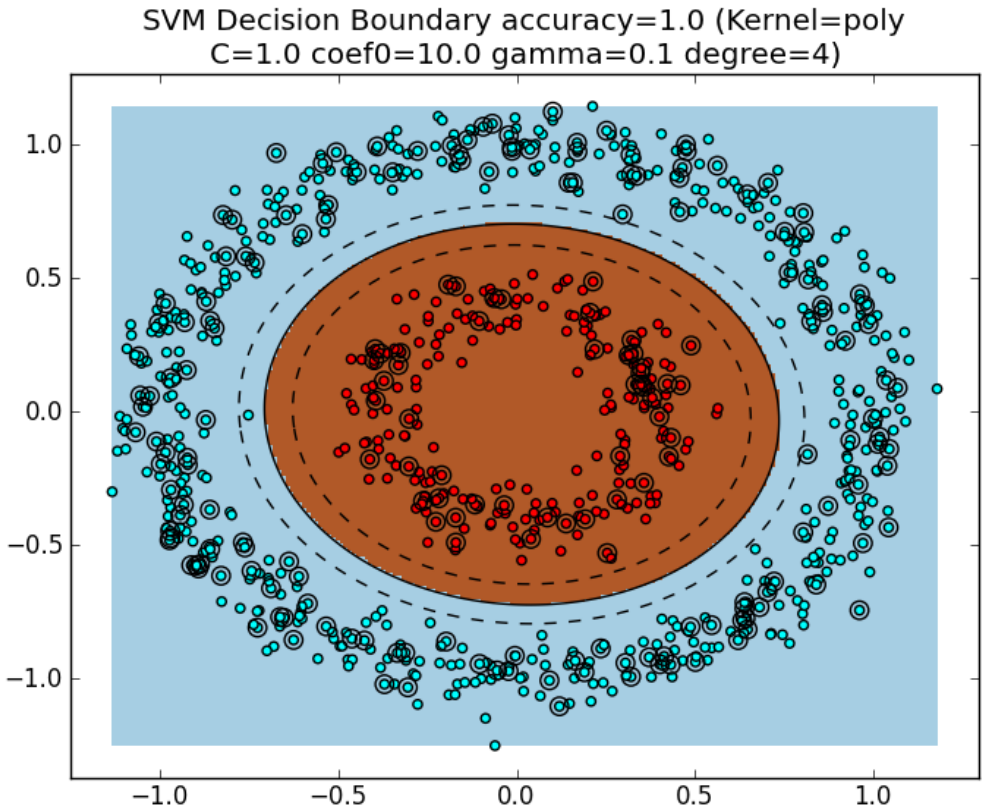
- **Polynomial kernel applied to non-linear data is an improvement over linear support vector classifiers**

Polynomial Kernel Example

- Circled data points are from the test set



$$K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p} x_{ij} x_{i'j})^d$$



[20] E. Kim

RBF Kernel

- Radial Basis Function (RBF) kernel (also known as radial kernel)
  - One of the mostly used kernel function
  - Uses parameter γ as positive constant

$$K(x_i, x_{i'}) = \exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2)$$

- 'Local Behaviour functionality'
  - Related to Euclidean distance measure (ruler distance)

$$d([x_1, x_2, \ldots, x_n],\ [y_1, y_2, \ldots, y_n]) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

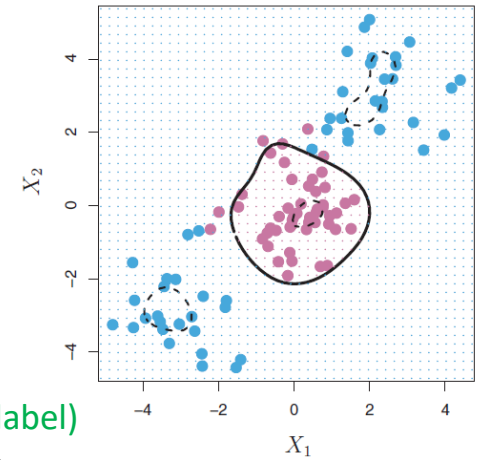
- Example
  - Use test data $x^* = (x_1^* \ldots x_p^*)^T$
  - Euclidean distance gives x^* far from x_i

$\sum_{j=1}^{p} (x_j^* - x_{ij})^2$ (large value with large distance)

$K(x^*, x_i) = \exp(-\gamma \sum_{j=1}^{p} (x_j^* - x_{ij})^2)$ (tiny value)

$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \alpha_i K(x, x_i)$ (training data x_i plays no role for x* & its class label)

(SVM with radial kernel)



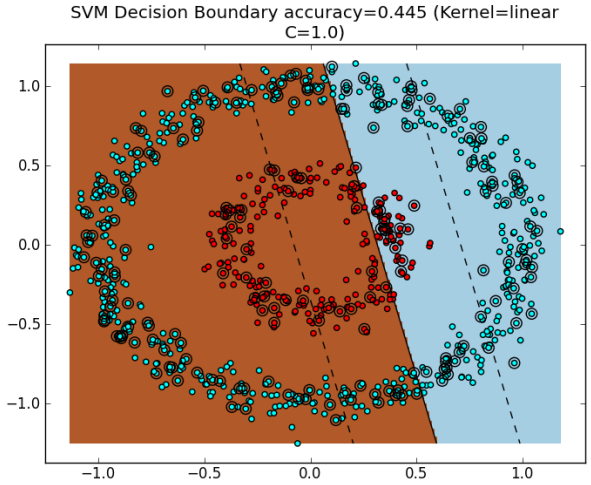
[1] An Introduction to Statistical Learning

- **RBF kernel have local behaviour (only nearby training data points have an effect on the class label**

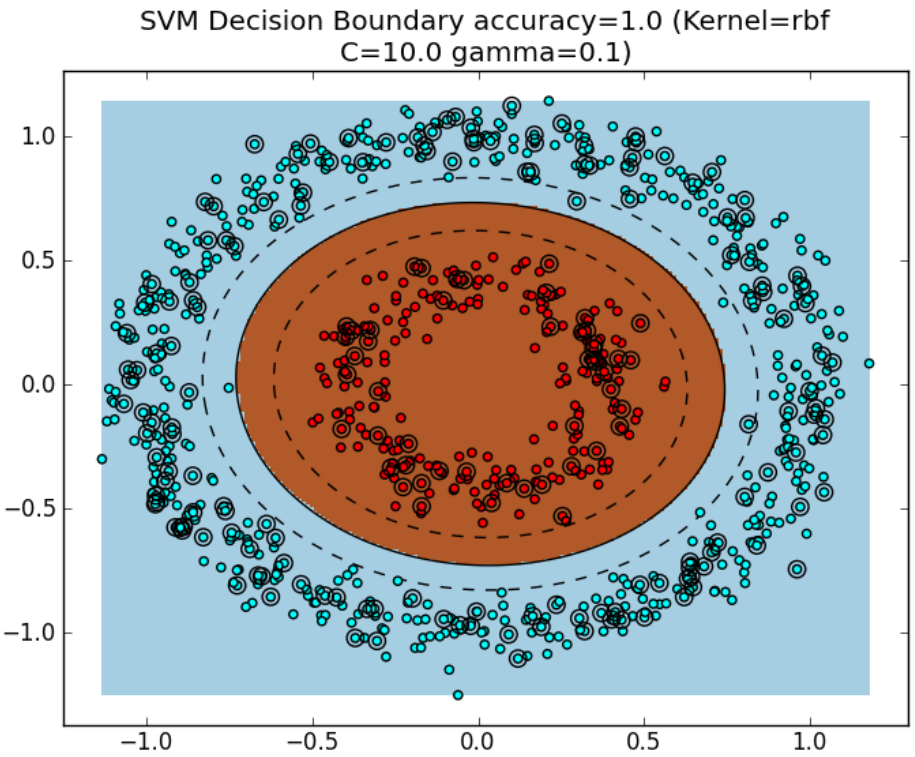
RBF Kernel Example

- Circled data points are from the test set

(similiar decision boundary as polynomial kernel)



$$K(x^*, x_i) = \exp(-\gamma \sum_{j=1}^{p} (x_j^* - x_{ij})^2)$$



[20] E. Kim

Exact SVM Definition using non-linear Kernels

- **True Support Vector Machines are Support Vector Classifiers combined with a non-linear kernel**
- **There are many non-linear kernels, but mostly known are polynomial and RBF kernels**

[6] An Introduction to Statistical Learning

- General form of SVM classifier
  - Assuming non-linear kernel function K
  - Based on 'smaller' collection S of SVs

$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \alpha_i K(x, x_i)$$

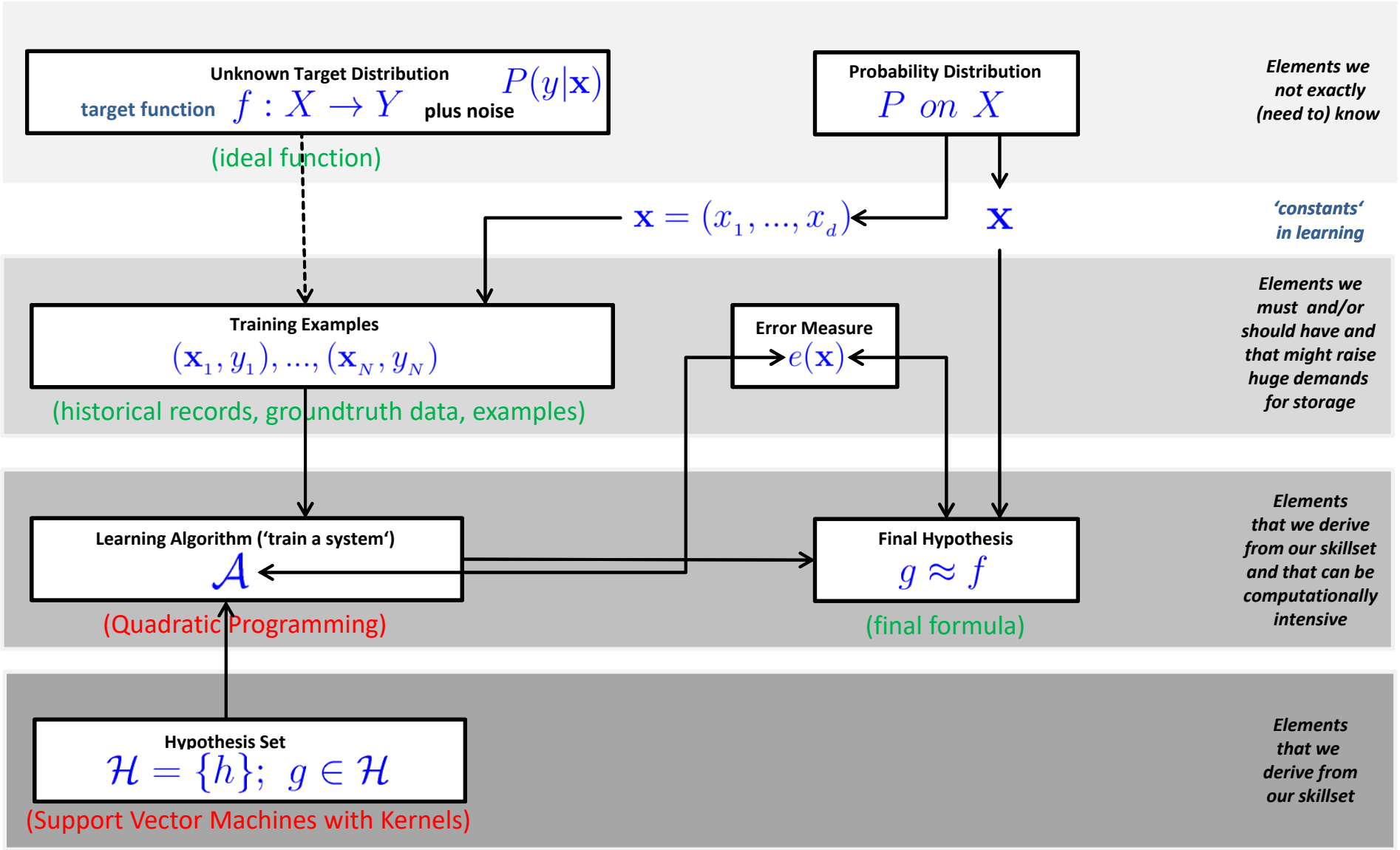
- Major benefit of Kernels: Computing done in original space

(independent from transformed space)

- Linear Kernel $K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j}$ (linear in features)
- Polynomial Kernel $K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p} x_{ij} x_{i'j})^d$ (polynomial of degree d)
- RBF Kernel $K(x_i, x_{i'}) = \exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2)$ (large distance, small impact)

(the win: kernel can compute this without ever computing the coordinates of the data in that space, next slides)

Solution Tools: Support Vector Classifier & QP Algorithm



Non-Linear Transformations with Support Vector Machines

- Same idea: work in z space instead of x space with SVMs
  - Understanding effect on solving (labels remain same)
  - SVMs will give the 'best seperator'

$$\mathcal{L}(\alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m \mathbf{x}_n^T \mathbf{x}_m$$

(replace this simply with z's obtained by Φ)

(result from this new inner product is given to quadratic programming optimization as input as before)

  - Value: inner product is done with z instead of x – the only change
  - Result after quadratic programming is hyperplane in z space using the value
- Impacts of Φ to optimization
  - From linear to 2D → probably no drastic change
  - From 2D to million-D → sounds like a drastic change but just inner product
  - Input for $K(x_i, x_i')$ remains the number of data points (nothing to do with million-D)
  - Computing longer million-D vectors is 'easy' – optimization steps 'difficult'

- **Infinite-D Z spaces are possible since the non-linear transformation does not affect the optimization**

Kernels & Infinite Z spaces

- Understanding advantage of using a kernel
  - Better than simply enlarging the feature space
  - E.g. using functions of the original features like $X_1, X_1^2, X_2, X_2^2, \ldots, X_p, X_p^2.$

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \emptyset(\boldsymbol{x}_i)^T \emptyset(\boldsymbol{x}_j)$$

(maps data to higher-dimensional feature spaces)

- Computational advantages
  - By using kernels only compute $K(x_i, x_i')$
  - Limited to just all $\binom{n}{2}$ distinct pairs $i,\ i'$

(number of 2 element sets from n element set)

  - Computing without explicitly working in the enlarged feature space
  - Important because in many applications the enlarged feature space is large

(computing would be infeasible then w/o kernels)

- Infinite-D Z spaces $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \emptyset(\boldsymbol{x}_i)^T \emptyset(\boldsymbol{x}_j)$
  - Possible since all that is needed to compute coefficients are inner products

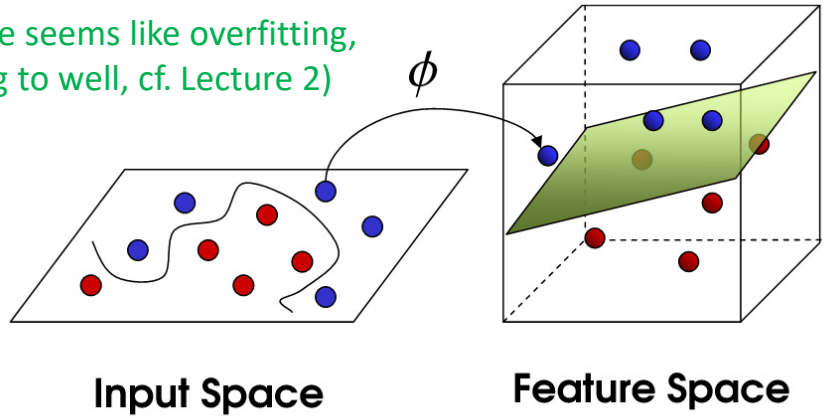
[1] An Introduction to Statistical Learning

- **Kernel methods like RBF have an implicit and infinite-dimensional features space that is not 'visited'**

Visualization of SVs

- Problem: z-Space is infinite (unknown)
  - How can the Support Vectors (from existing points) be visualized?
  - Solution: non-zero alphas have been the identified support vectors
    (solution of quadratic programming optimization will be a set of alphas we can visualize)
  - Support vectors exist in Z – space (just transformed original data points)
  - Example: million-D means a million-D vector for $\mathbf{w}$
  - But number of support vector is very low, expected E_{out} is related to #SVs
    (generalization behaviour despite million-D & snake-like overfitting)

(snake seems like overfitting, fitting to well, cf. Lecture 2)

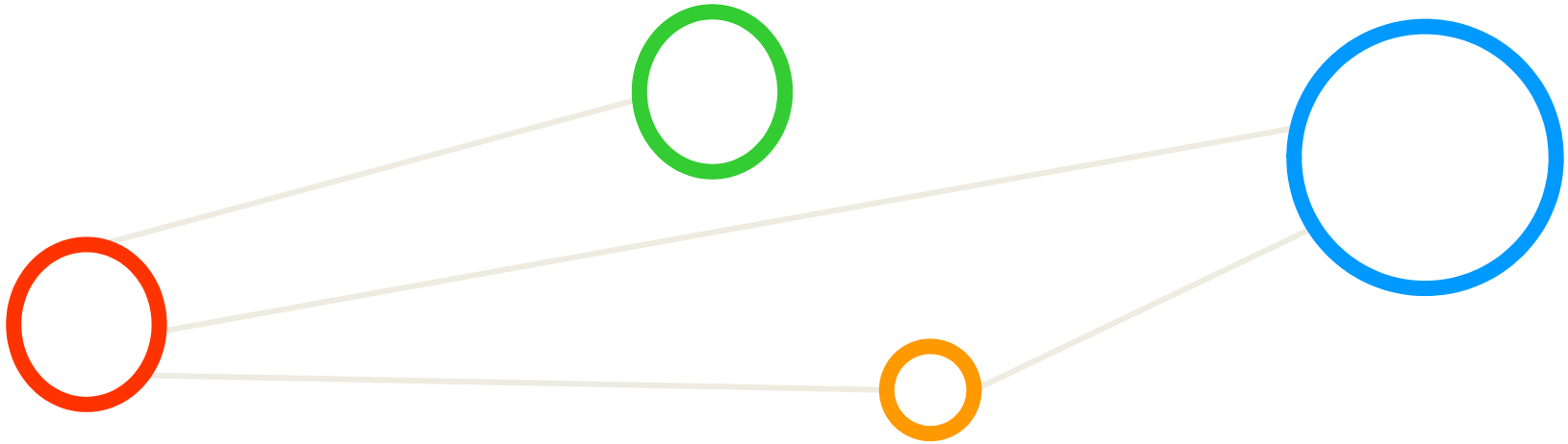


[21] Visualization of high-dimensional space

- **Counting the number of support vectors remains to be a good indicator for generalization behaviour even when performing non-linear transforms and kernel methods that can lead to infinite-D spaces**

(rule of thumb)

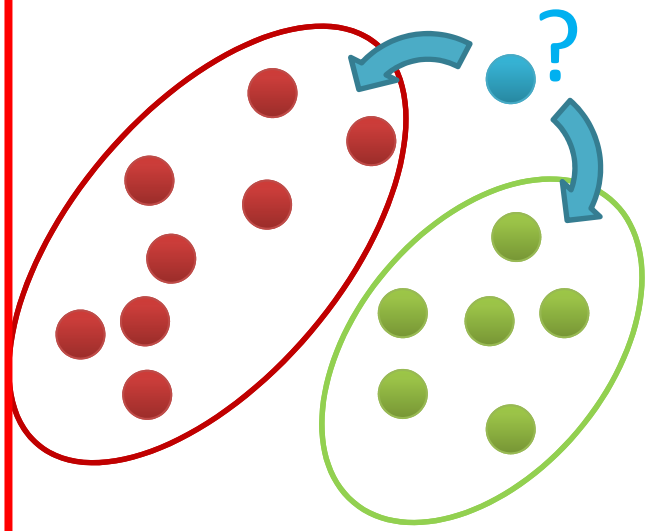
Appendix E: Simple Introduction to Machine Learning



Machine Learning Methods Overview

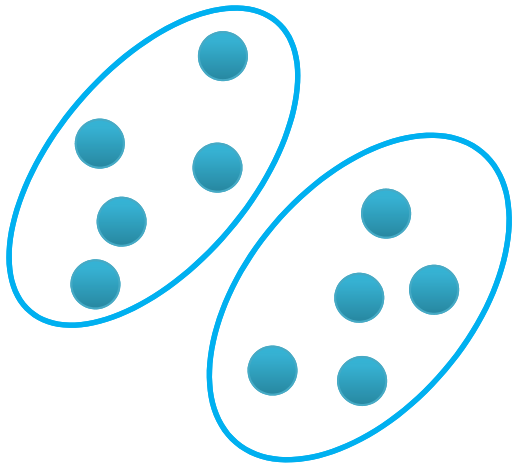
- **Machine learning methods can be roughly categorized in classification, clustering, or regression augmented with various techniques for data exploration, selection, or reduction**

Classification



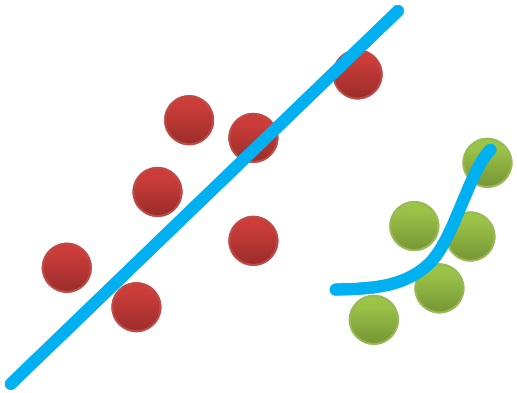
- Groups of data exist
- New data classified to existing groups

Clustering



- No groups of data exist
- Create groups from data close to each other

Regression

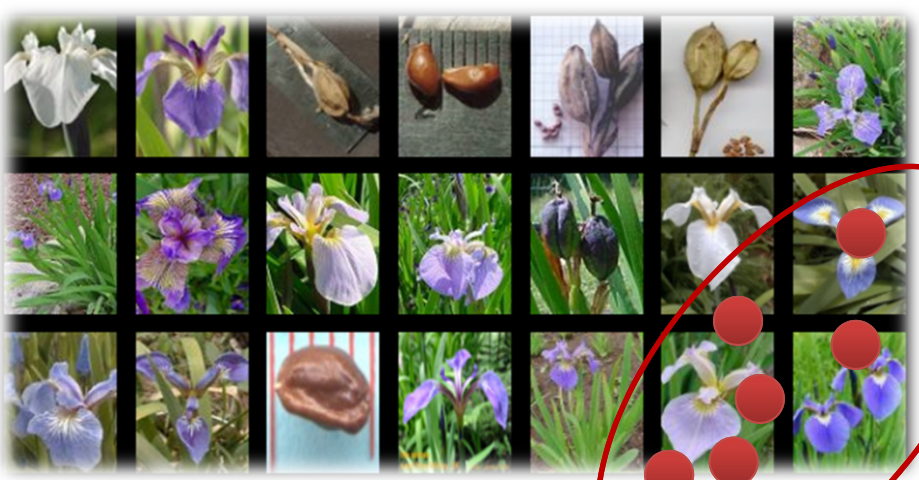


- Identify a line with a certain slope describing the data

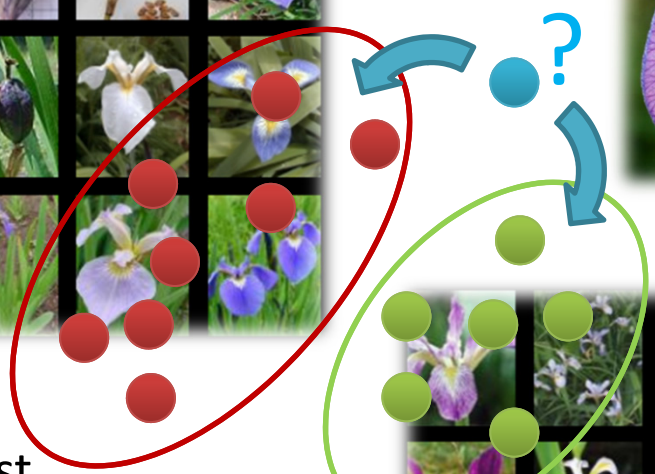
Simple Application Example: Classification of a Flower

(1) Problem Understanding Phase

(what type of flower is this?)



(flowers of type 'IRIS Setosa')



(flowers of type 'IRIS Virginica')

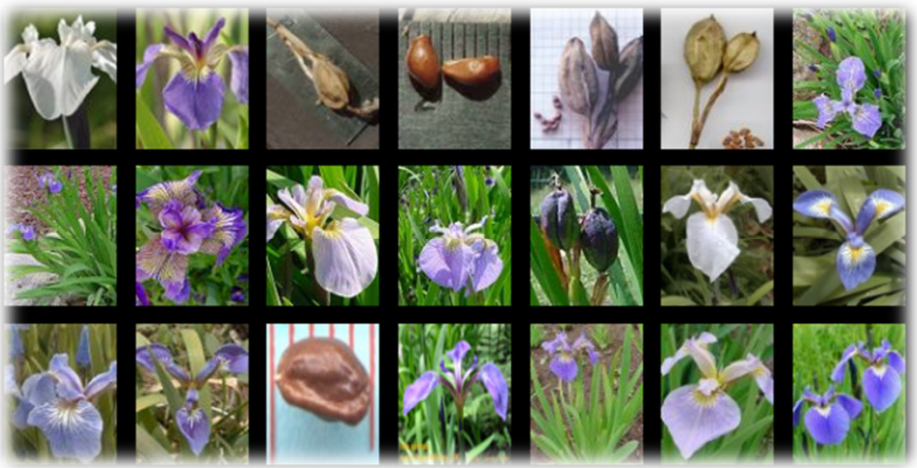
- Groups of data exist
- New data classified to existing groups

[1] Image sources: Species Iris Group of North America Database, www.signa.org

➢ **Appendix E offers a simple introduction to machine learning for a two class classification problem**

The Learning Problem in the Example

(flowers of type ‘IRIS Setosa‘)



(flowers of type ‘IRIS Virginica‘)

[1] Image sources: Species Iris Group of North America Database, www.signa.org

Learning problem: A prediction task

- Determine whether a new Iris flower sample is a “Setosa” or “Virginica”
- Binary (two class) classification problem
- What attributes about the data help?

(what type of flower is this?)

Feasibility of Machine Learning in this Example

1. Some pattern exists:
   - Believe in a ‘pattern with ‘petal length‘ & ‘petal width‘ somehow influence the type
2. No exact mathematical formula
   - To the best of our knowledge there is no precise formula for this problem
3. Data exists
   - Data collection from UCI Dataset „Iris“
   - 150 labelled samples (aka ‘data points‘)
   - Balanced: 50 samples / class

[2] Image source: Wikipedia, Sepal

(2) Data Understanding Phase

[3] UCI Machine Learning Repository Iris Dataset

[5] Iris Dataset

(four data attributes for each sample in the dataset)

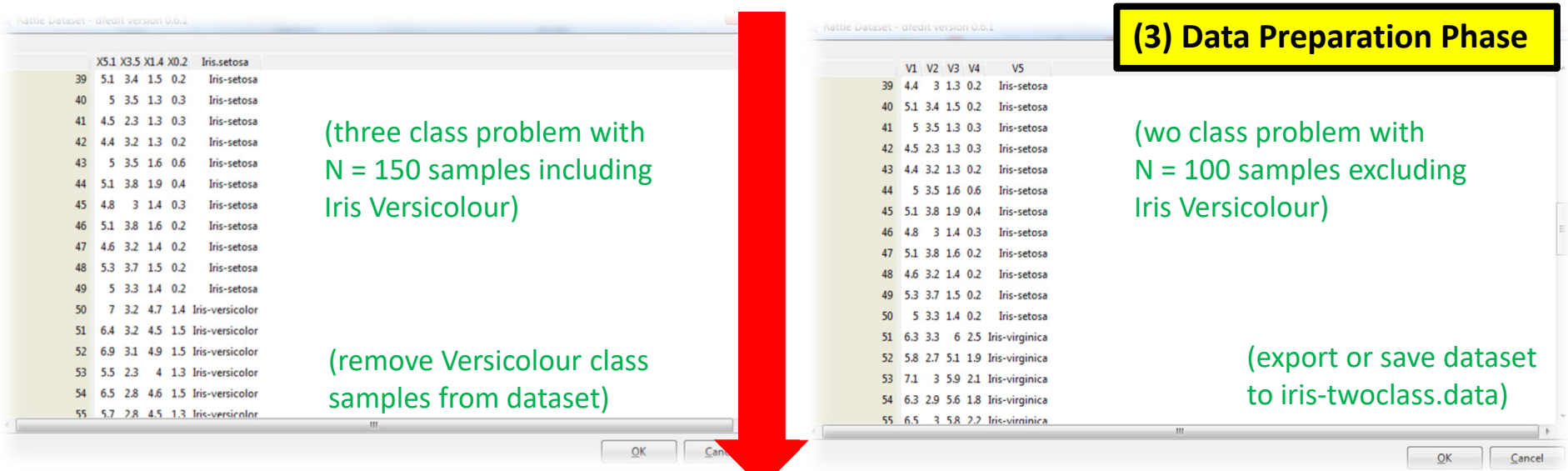
- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm

(one class label for each sample in the dataset)

- class: Iris Setosa, or Iris Versicolour, or Iris Virginica

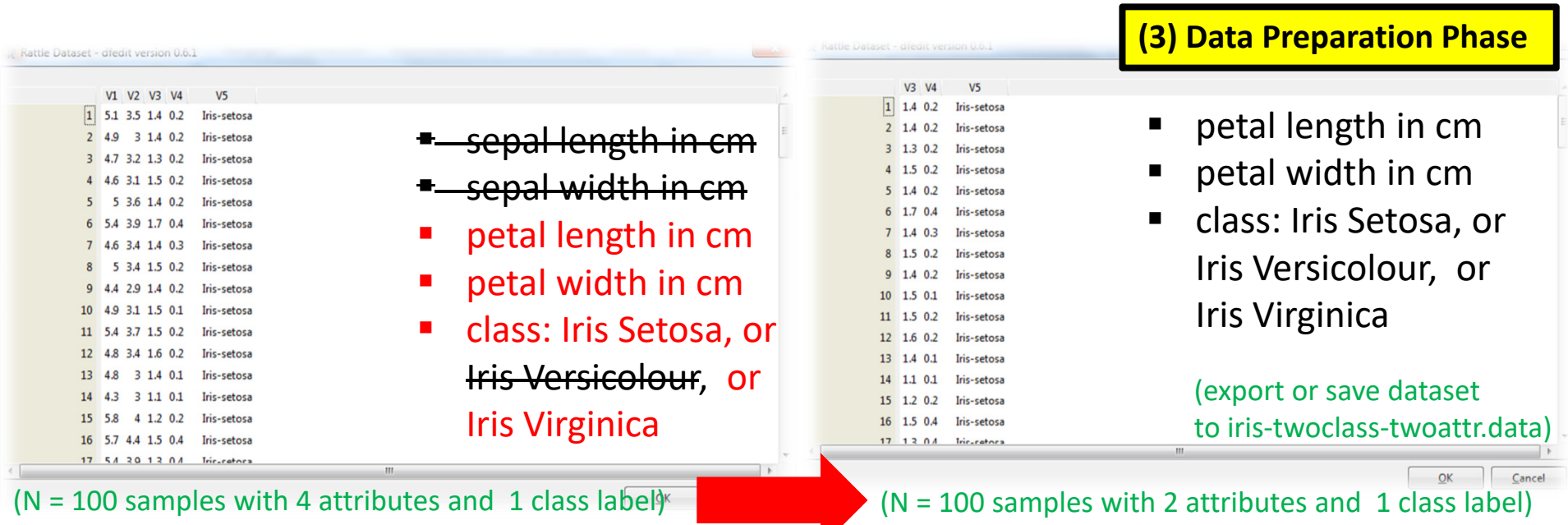
Sampling as Data Preparation – Remove Third Class Samples

- Data preparation means to prepare our data for our problem
  - In practice the whole dataset is rarely needed to solve one problem
  - E.g. apply several sampling strategies (but be aware of class balance)
- Recall: Our learning problem
  - Determine whether a new Iris flower sample is a “Setosa” or “Virginica”
  - Binary (two class) classification problem : ‘Setosa‘ or ‘Virginica‘

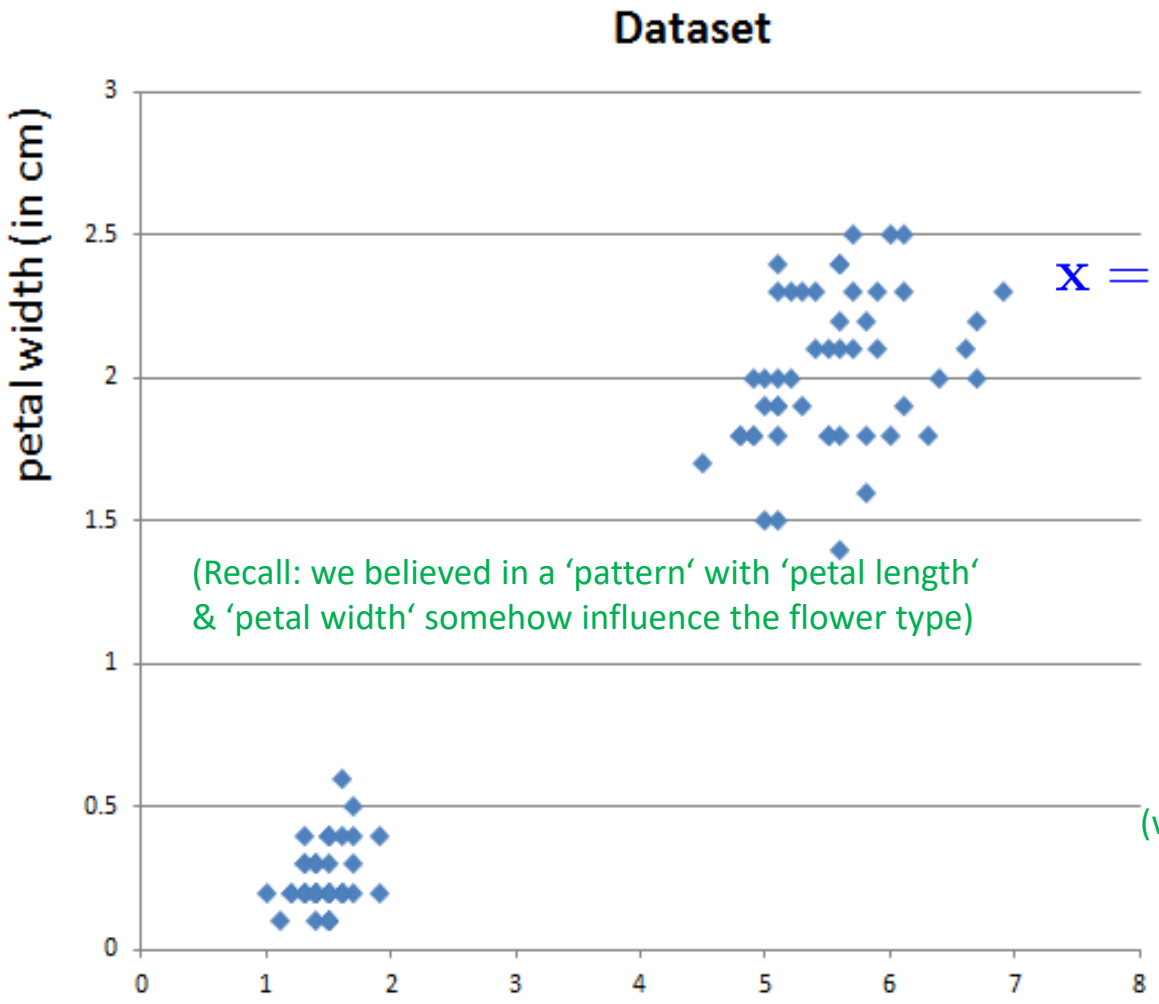


Feature Selection as Data Preparation – Select Attributes

- Data preparation means to prepare our data for our problem
  - In practice the whole dataset is rarely needed to solve one problem
  - E.g. perform feature selection (aka remove not needed attributes)
- Recall: Our believed pattern in the data
  - A ‘pattern with ‘petal length‘ & ‘petal width‘ somehow influence the type



Check Preparation Phase: Plotting the Data



(attributes with d=2)

$\mathbf{x} = x_1, \ldots, x_d$

(x1 is petal length,
x2 is petal width)

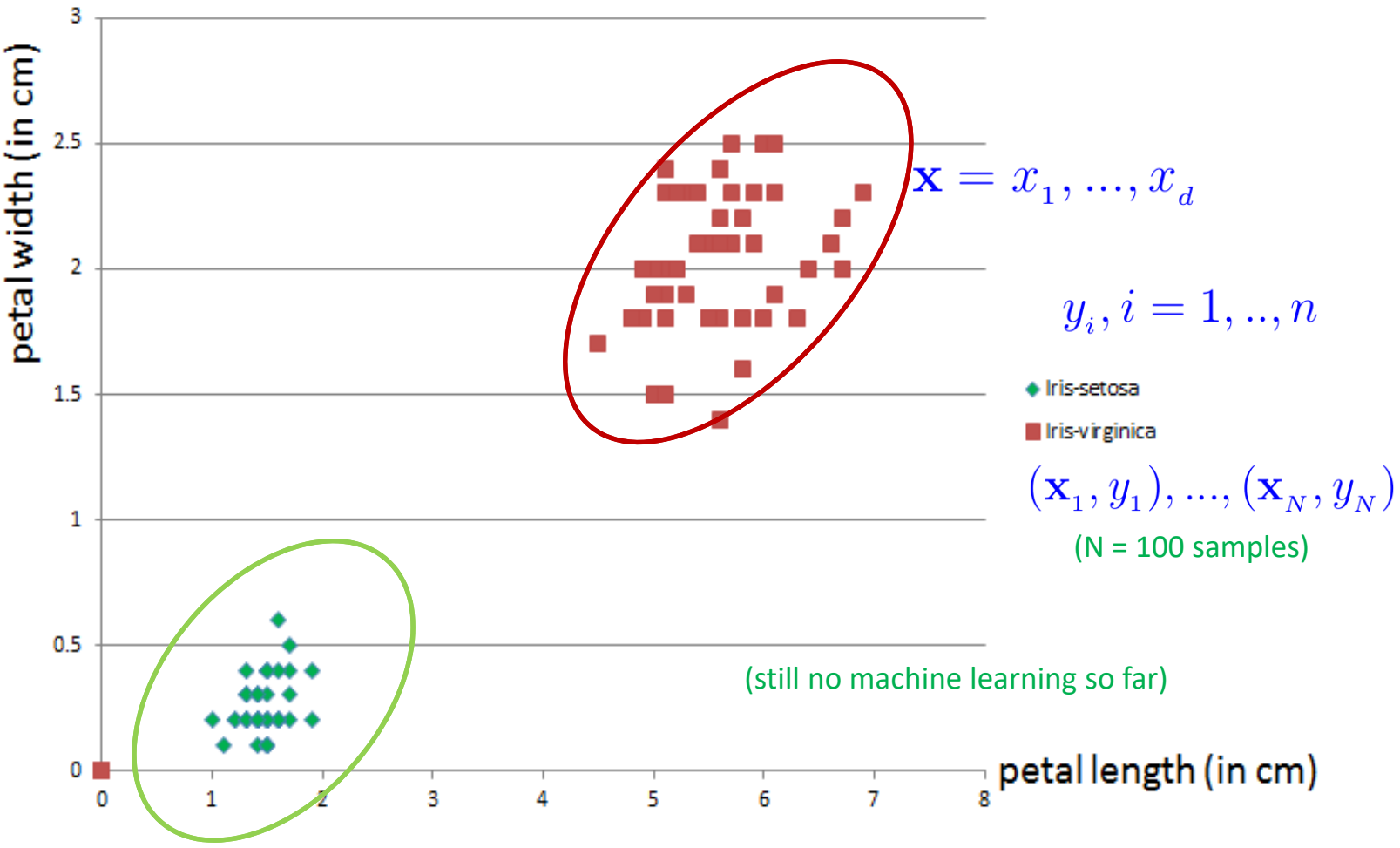
$(\mathbf{x}_1), \ldots, (\mathbf{x}_N)$

(N = 100 samples)

(Recall: we believed in a ‘pattern‘ with ‘petal length‘
& ‘petal width‘ somehow influence the flower type)

(what about the class labels?)

Check Preparation Phase: Class Labels



$\mathbf{x} = x_1, \ldots, x_d$

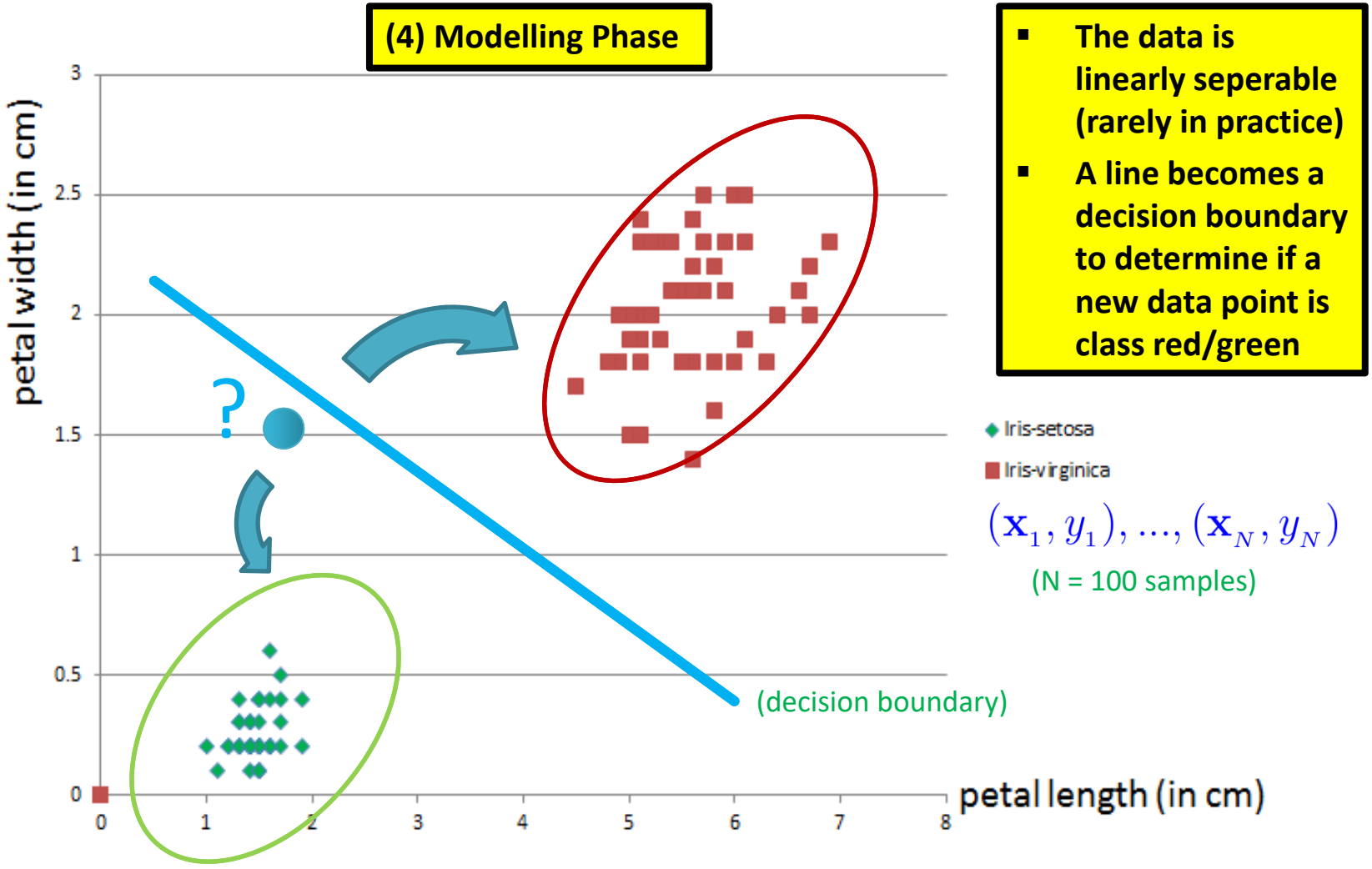
$y_i, i = 1, .., n$

$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$

(N = 100 samples)

(still no machine learning so far)

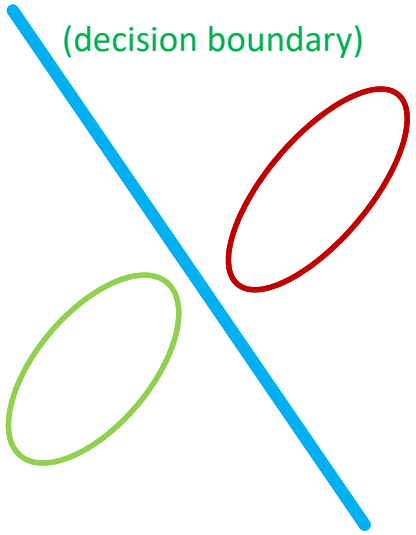
Linearly Seperable Data & Linear Decision Boundary



- The data is linearly seperable (rarely in practice)
- A line becomes a decision boundary to determine if a new data point is class red/green

Separating Line & Mathematical Notation

- Data exploration results
  - A line can be crafted between the classes since linearly seperable data
  - All the data points representing Iris-setosa will be below the line
  - All the data points representing Iris-virginica will be above the line
- More formal mathematical notation
  - Input: $\mathbf{x} = x_1, \ldots, x_d$ (attributes of flowers)
  - Output: class +1 (Iris-virginica) or class -1 (Iris-setosa)



Iris-virginica if $\sum_{i=1}^{d} w_i x_i > threshold$

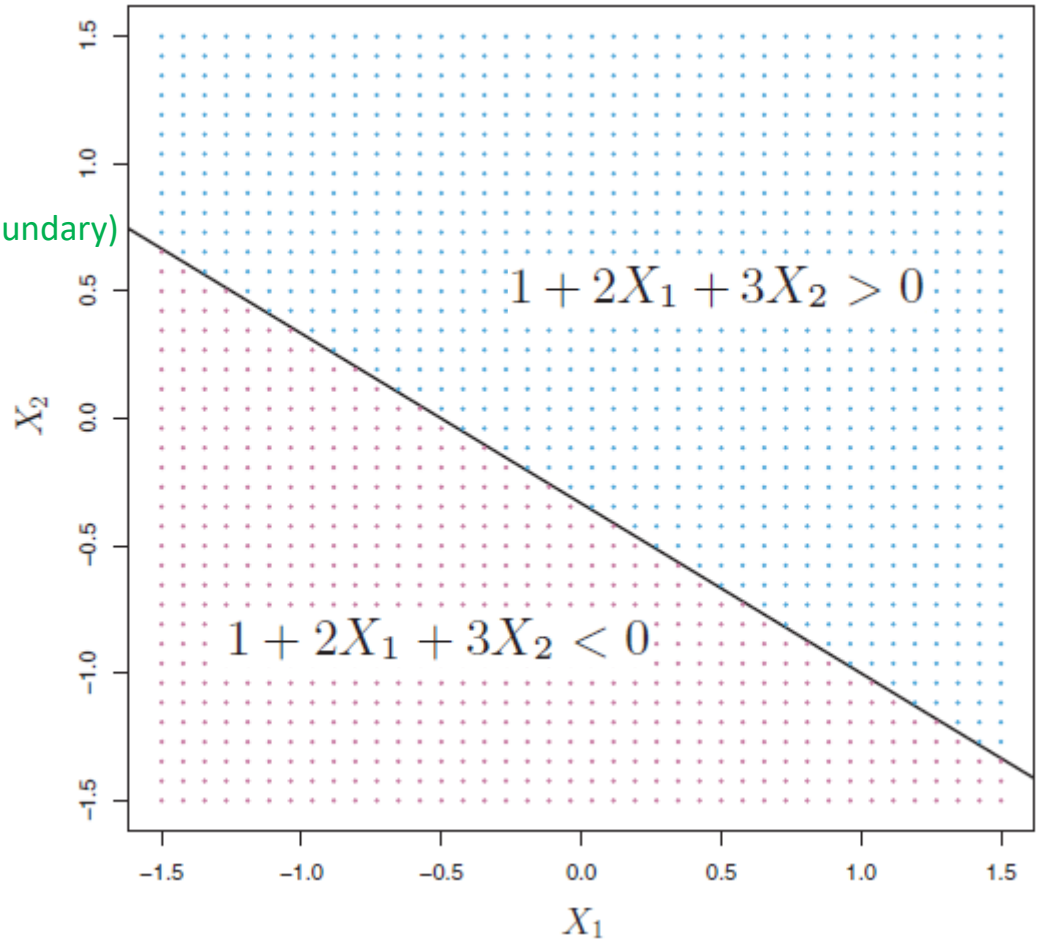
Iris-setosa if $\sum_{i=1}^{d} w_i x_i < threshold$

(w_i and threshold are still unknown to us)

$$sign\left(\left(\sum_{i=1}^{d} w_i x_i\right) - threshold\right)$$ (compact notation)

Separating Line & ‘Decision Space‘ Example

(decision boundary)



(equation of a line)

$$1 + 2X_1 + 3X_2 = 0$$

(all points X_i on this line have to satisfy this equation)

modified from [6] An Introduction to Statistical Learning

A Simple Linear Learning Model – The Perceptron

- Human analogy in learning *[7] F. Rosenblatt, 1957*
  - Human brain consists of nerve cells called neurons
  - Human brain learns by changing the strength of neuron connections (w_i) upon repeated stimulation by the same impulse (aka a 'training phase')
  - Training a perceptron model means adapting the weights w_i
  - Done until they fit input-output relationships of the given 'training data'

$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$

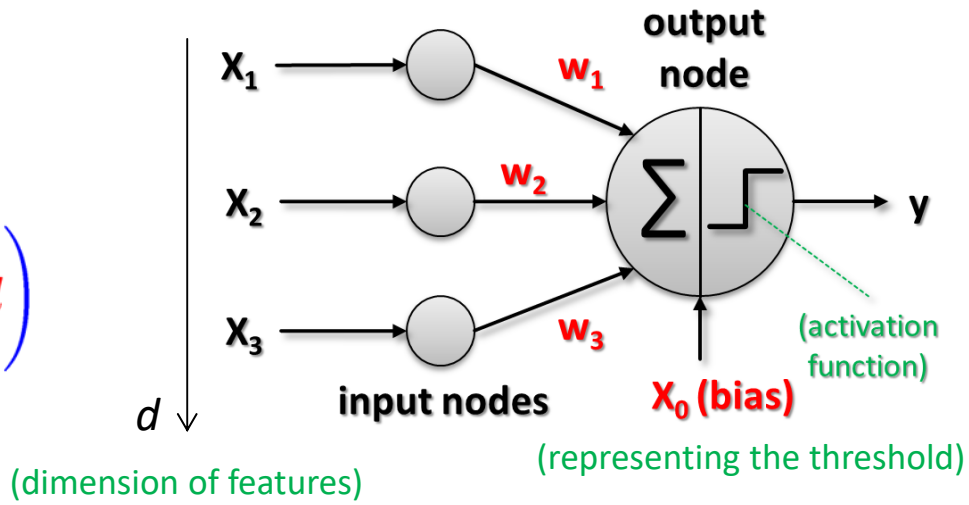
(training data)

$$sign\left(\left(\sum_{i=1}^{d} w_i x_i\right) - threshold\right)$$

(activation function, +1 or -1)

(the signal)

(modelled as bias term)



Perceptron – Example of a Boolean Function

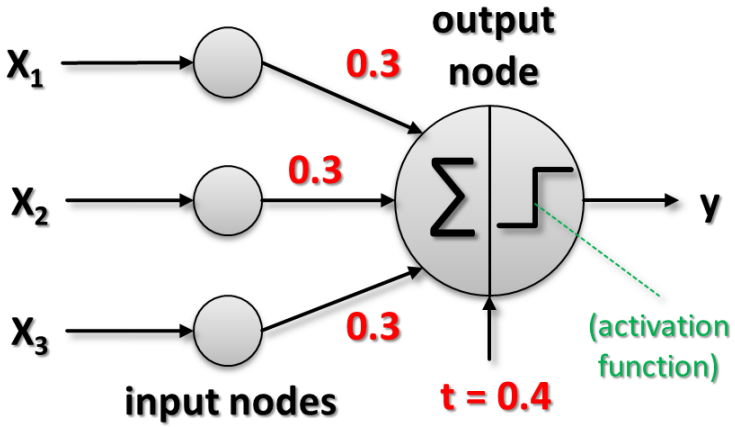
	x1	x2	x3	y
1	1	0	0	-1
2	1	0	1	1
3	1	1	0	1
4	1	1	1	1
5	0	0	1	-1
6	0	1	0	-1
7	0	1	1	1
8	0	0	0	-1

$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$

(training data)

(training phase)

$$sign\left(\left(\sum_{i=1}^{d} w_i x_i\right) - threshold\right)$$



(trained perceptron model)

- Output node interpretation
  - More than just the weighted sum of the inputs – threshold (aka bias)
  - Activation function sign (weighted sum): takes sign of the resulting sum

$$y = 1, \textbf{if } 0.3x_1 + 0.3x_2 + 0.3x_3 - 0.4 > 0$$

(e.g. consider sample #3, sum is positive (0.2) → +1)

$$y = -1, \textbf{if } 0.3x_1 + 0.3x_2 + 0.3x_3 - 0.4 < 0$$

(e.g. consider sample #6, sum is negative (-0.1) → -1)

Summary Perceptron & Hypothesis Set h(x)

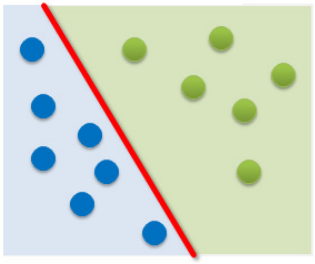
- When: Solving a linear classification problem *[7] F. Rosenblatt, 1957*
  - Goal: learn a simple value (+1/-1) above/below a certain threshold
  - Class label renamed: Iris-setosa = -1 and Iris-virginica = +1
- Input: $\mathbf{x} = x_1, \ldots, x_d$ (attributes in one dataset)

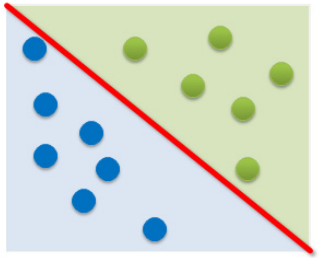
- Linear formula (take attributes and give them different weights – think of 'impact of the attribute')
  - All learned formulas are different hypothesis for the given problem

$$h(\mathbf{x}) = sign\left(\left(\sum_{i=1}^{d} w_i x_i\right) - threshold\right); h \in \mathcal{H}$$

(parameters that define one hypothesis vs. another)

(each green space and blue space are regions of the same class label determined by sign function)





(red parameters correspond to the redline in graphics)

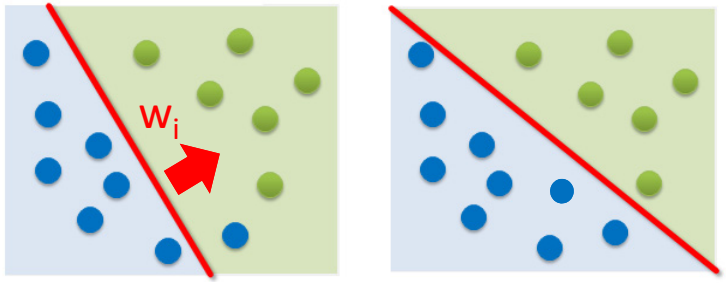
(but question remains: how do we actually learn w_i and threshold?)

Perceptron Learning Algorithm – Understanding Vector W

- When: If we believe there is a linear pattern to be detected
  - Assumption: Linearly seperable data (lets the algorithm converge)
  - Decision boundary: perpendicular vector w_i fixes orientation of the line

$$\mathbf{w}^T\mathbf{x} = 0$$
$$\mathbf{w} \cdot \mathbf{x} = 0$$

(points on the decision boundary satisfy this equation)



$$h(\mathbf{x}) = sign(\mathbf{w}^T\mathbf{x})$$

(vector notation, using T = transpose)

$$\mathbf{w}_i = (w_{i1}, w_{i2}, \ldots, w_d)$$

$$\mathbf{w}_i^T = \begin{bmatrix} w_{i1} \\ w_{i2} \\ \ldots \\ w_{id} \end{bmatrix}$$

- Possible via simplifications since we also need to learn the threshold:

$$h(\mathbf{x}) = sign\left(\left(\sum_{i=1}^{d} w_i x_i\right) + w_0\right); w_0 = -threshold$$

$$\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_d)$$

$$h(\mathbf{x}) = sign\left(\left(\sum_{i=0}^{d} w_i x_i\right)\right); x_0 = 1$$

$$h(\mathbf{x}) = sign(\mathbf{w} \cdot \mathbf{x})$$

(equivalent dotproduct notation)

(all notations are equivalent and result is a scalar from which we derive the sign)

[8] Rosenblatt, 1958

Understanding the Dot Product – Example & Interpretation

- ‘Dot product‘
  - Given two vectors
  - Multiplying corresponding components of the vector
  - Then adding the resulting products
  - Simple example: $(2,3)\cdot(4,1) = 2*4+3*1 = \boxed{11}$ (a scalar!)
  - Interesting: Dot product of two vectors is a scalar

$$\mathbf{u}\cdot\mathbf{v} = \sum_{i=1}^{n} u_i v_i$$

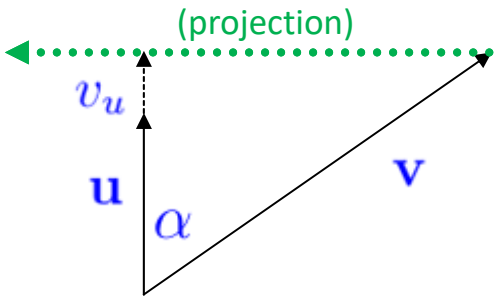
$$h(\mathbf{x}) = sign\left(\left(\sum_{i=0}^{d} w_i x_i\right)\right); x_0 = 1$$

$$h(\mathbf{x}) = sign(\mathbf{w}\cdot\mathbf{x})$$

(our example)

- ‘Projection capabilities of Dot product‘ (simplified)
  - Orthogonal projection of vector $\mathbf{v}$ in the direction of vector $\mathbf{u}$

$$\mathbf{u}\cdot\mathbf{v} = (\|v\|cos(\alpha))\|u\| = v_u\|u\|$$



- Normalize using length of vector

$$\frac{\mathbf{u}}{\|\mathbf{u}\|} \quad \|\mathbf{u}\| = length(\mathbf{u}) = L_2 norm = \sqrt{\mathbf{u}\cdot\mathbf{u}}$$

➢ **Dot Products are important in machine learning, e.g. in Support Vector Machines, see Appendix C**

Perceptron Learning Algorithm – Learning Step

- Iterative Method using (labelled) training data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$

(one point at a time is picked)

1. Pick one misclassified training point where:

$$sign(\mathbf{w}^T \mathbf{x}_n) \neq y_n$$

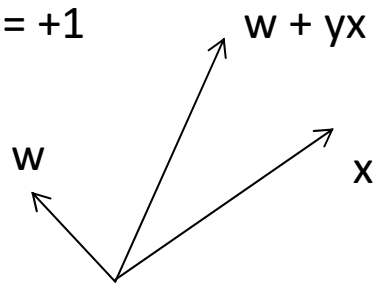
2. Update the weight vector:

$$\mathbf{w} \leftarrow \mathbf{w} + y_n \mathbf{x}_n$$

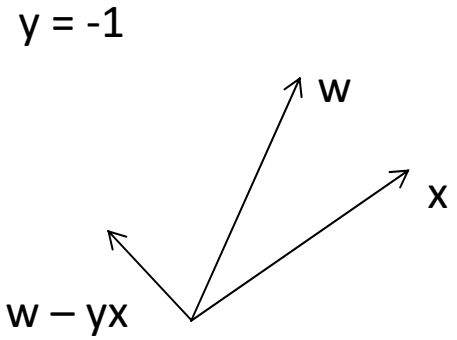
(y_n is either +1 or -1)

(a) adding a vector or
(b) subtracting a vector

(a)



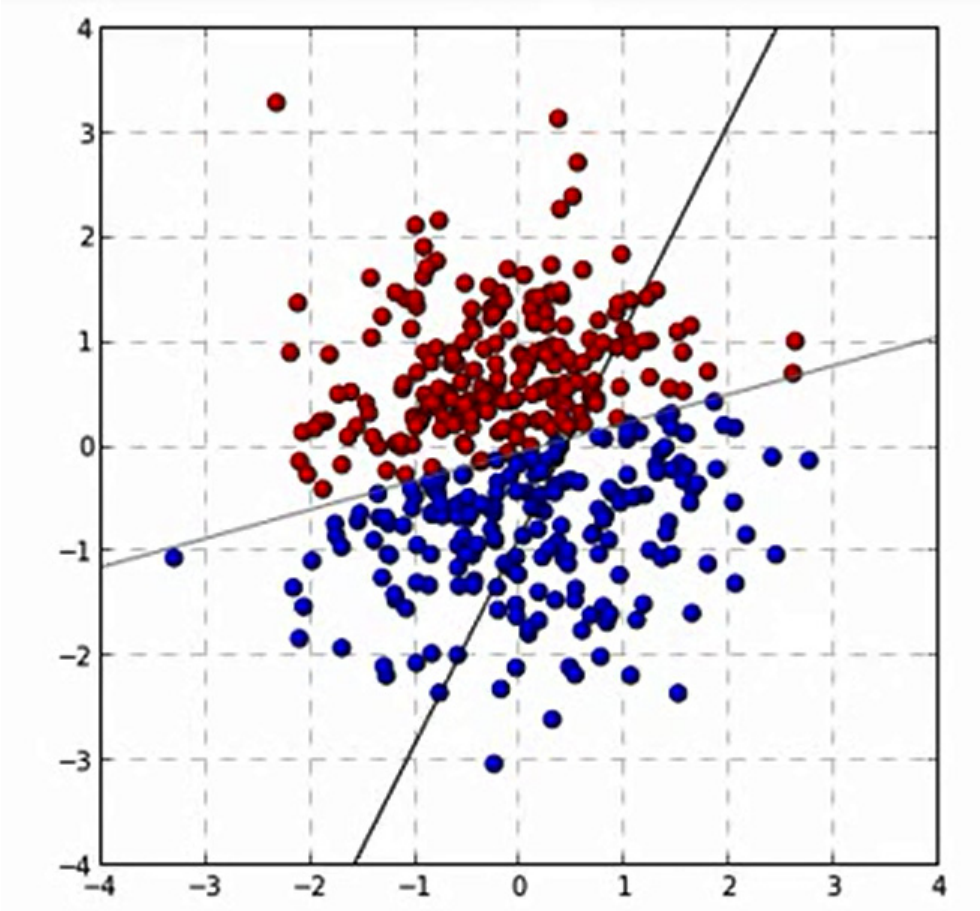
(b)



- Terminates when there are no misclassified points

(converges only with linearly seperable data)

[Video] Perceptron Learning Algorithm



[9] PLA Video

Machine Learning & Data Mining Tasks in Applications

- **Machine learning tasks can be divided into two major categories: Predictive and Descriptive Tasks**

[11] Introduction to Data Mining

- Predictive Tasks
  - Predicts the value of an attribute based on values of other attributes
  - Target/dependent variable: attribute to be predicted
  - Explanatory/independent variables: attributed used for making predictions
  - E.g. predicting the species of a flower based on characteristics of a flower
- Descriptive Tasks
  - Derive patterns that summarize the underlying relationships in the data
  - Patterns here can refer to correlations, trends, trajectories, anomalies
  - Often exploratory in nature and frequently require postprocessing
  - E.g. credit card fraud detection with unusual transactions for owners

Summary Terminologies & Different Dataset Elements

- Target Function $f : X \rightarrow Y$
  - Ideal function that 'explains' the data we want to learn
- Labelled Dataset (samples)
  - 'in-sample' data given to us: $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$
- Learning vs. Memorizing
  - The goal is to create a system that works well 'out of sample'
  - In other words we want to classify 'future data' (ouf of sample) correct
- Dataset Part One: Training set **(4) Modelling Phase**
  - Used for training a machine learning algorithms
  - Result after using a training set: a trained system
- Dataset Part Two: Test set **(5) Evaluation Phase**
  - Used for testing whether the trained system might work well
  - Result after using a test set: accuracy of the trained model

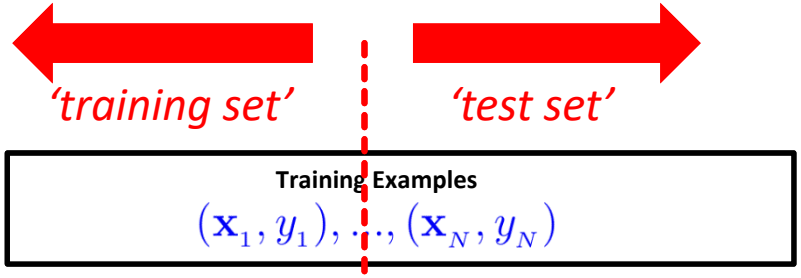
Model Evaluation – Training and Testing Phases

- Different Phases in Learning
  - Training phase is a hypothesis search
  - Testing phase checks if we are on right track (once the hypothesis clear)
- Work on ‘training examples‘
  - Create two disjoint datasets
  - One used for training only (aka training set)
  - Another used for testing only (aka test set)

(4) Modelling Phase

(5) Evaluation Phase

(e.g. student exam training on examples to get E_{in} ‚down‘, then test via exam)



(historical records, groundtruth data, examples)

  - Exact seperation is rule of thumb per use case (e.g. 10 % training, 90% test)
  - Practice: If you get a dataset take immediately test data away (‘throw it into the corner and forget about it during modelling‘)
  - Reasoning: Once we learned from training data it has an ‘optimistic bias‘

Model Evaluation – Testing Phase & Confusion Matrix

(5) Evaluation Phase

- Model is fixed
  - Model is just used with the testset
  - Parameter w_i are set and we have a linear decision function
- Evaluation of model performance
  - Counts of test records that are incorrectly predicted $sign(\mathbf{w}^T\mathbf{x}_n) \neq y_n$
  - Counts of test records that are correctly predicted $sign(\mathbf{w}^T\mathbf{x}_n) = y_n$
  - E.g. create confusion matrix for a two class problem

Counting per sample		Predicted Class	
		Class = 1	Class = 0
Actual Class	Class = 1	f_{11}	f_{10}
	Class = 0	f_{01}	f_{00}

(serves as a basis for further performance metrics usually used)

Model Evaluation – Testing Phase & Performance Metrics

Counting per sample		Predicted Class	
		Class = 1	Class = 0
Actual Class	Class = 1	f_{11}	f_{10}
	Class = 0	f_{01}	f_{00}

(5) Evaluation Phase

(100% accuracy in learning often points to problems using machine learning methos in practice)

- Accuracy (usually in %)

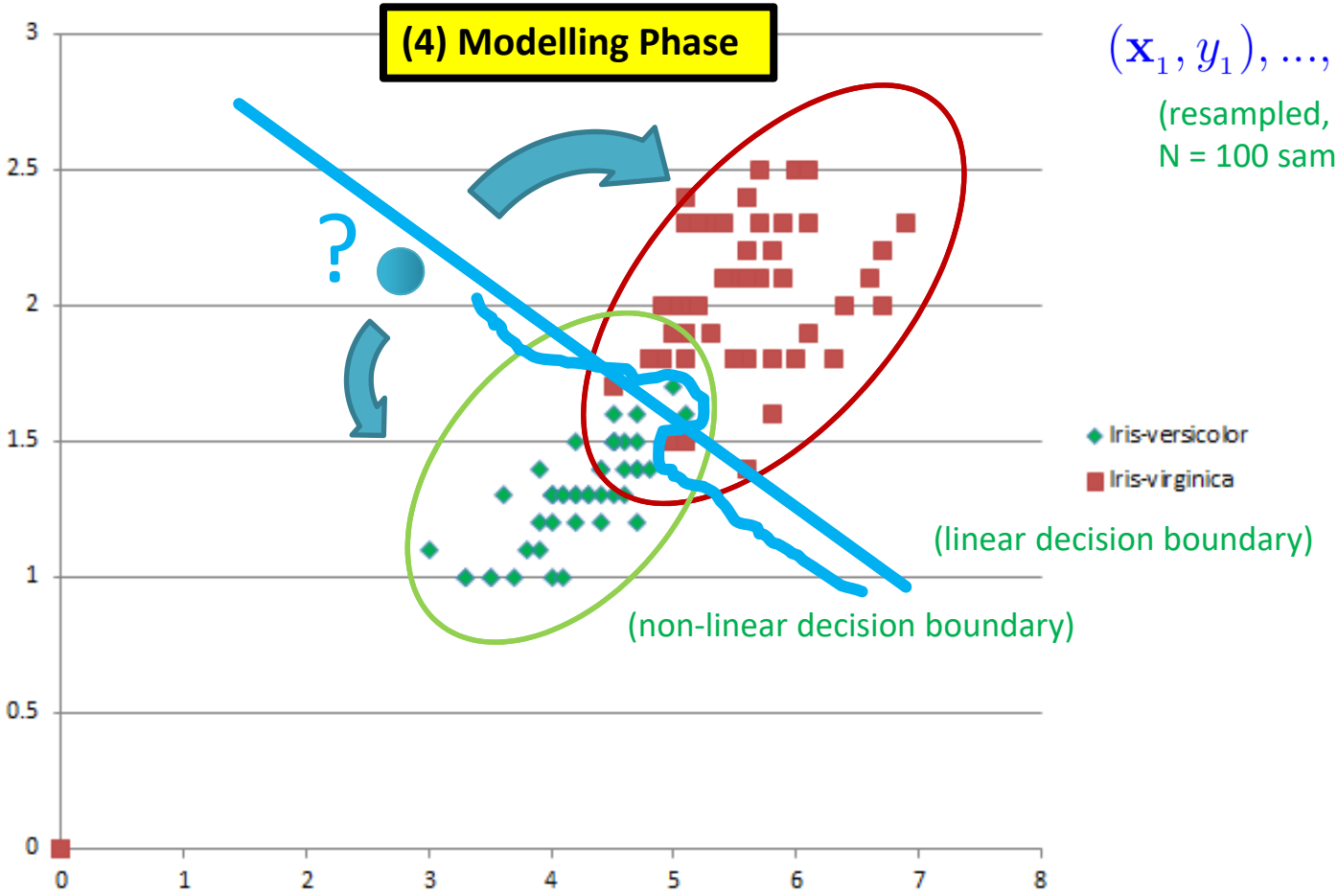
$$Accuracy = \frac{number\ of\ correct\ predictions}{total\ number\ of\ predictions}$$

- Error rate

$$Error\ rate = \frac{number\ of\ wrong\ predictions}{total\ number\ of\ predictions}$$

- If model evaluation is satisfactory: **(6) Deployment Phase**

Non-linearly Seperable Data in Practice – Which model?



$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$

(resampled, again
N = 100 samples)

(lessons learned from practice: requires soft-thresholds to allow for some errors being overall better for new data
→ Occams razor – 'simple model better')

(lessons learned from practice: requires non-linear decision boundaries)

Learning Approaches – What means Learning?

- **The basic meaning of learning is 'to use a set of observations to uncover an underlying process'**
- **The three different learning approaches are supervised, unsupervised, and reinforcement learning**

- Supervised Learning
  - Majority of methods follow this approach in this course
  - Example: credit card approval based on previous customer applications
- Unsupervised Learning
  - Often applied before other learning → higher level data representation
  - Example: Coin recognition in vending machine based on weight and size
- Reinforcement Learning
  - Typical 'human way' of learning
  - Example: Toddler tries to touch a hot cup of tea (again and again)

➢ **Appendix B provides an introduction to statistical learning theory & feasibility of learning**

Learning Approaches – Supervised Learning

- Each observation of the predictor measurement(s) has an associated response measurement:
  - Input $\mathbf{x} = x_1, ..., x_d$
  - Output $y_i, i = 1, .., n$
  - Data $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)$
- Goal: Fit a model that relates the response to the predictors
  - Prediction: Aims of accurately predicting the response for future observations
  - Inference: Aims to better understanding the relationship between the response and the predictors

- **Supervised learning approaches fits a model that related the response to the predictors**
- **Supervised learning approaches are used in classification algorithms such as SVMs**
- **Supervised learning works with data = [input, correct output]**

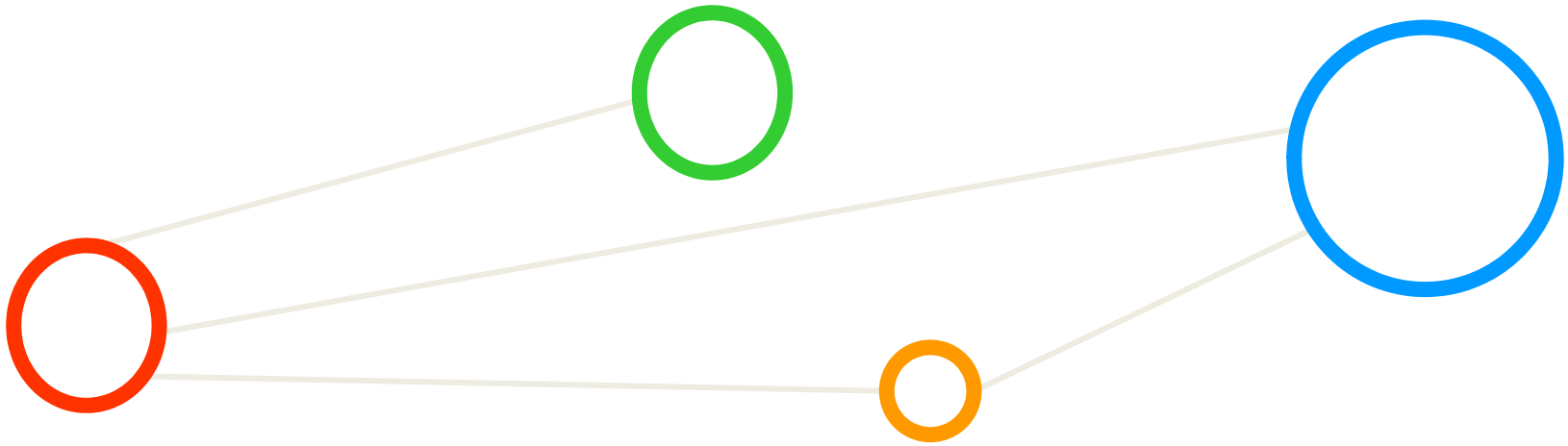
[6] An Introduction to Statistical Learning

Key Challenges: Why is it not so easy in practice?

- Scalability
  - Gigabytes, Terabytes, and Petabytes datasets that fit not into memory
  - E.g. algorithms become necessary with out-of-core/CPU strategies
- High Dimensionality
  - Datasets with hundreds or thousand attributes become available
  - E.g. bioinformatics with gene expression data with thousand of features
- Heterogenous and Complex Data
  - More complex data objects emerge and unstructured data sets
  - E.g. Earth observation time-series data across the globe
- Data Ownership and Distribution
  - Distributed datasets are common (e.g. security and transfer challenges)

- **Key challenges faced when doing traditional data analysis and machine learning are scalability, high dimensionality of datasets, heterogenous and complex data, data ownership & distribution**
- **Combat 'overfitting' is the key challenge in machine learning using validation & regularization**

Lecture Bibliography



Lecture Bibliography (1)

- [1] Species Iris Group of North America Database,
  Online: http://www.signa.org
- [2] UCI Machine Learning Repository Iris Dataset,
  Online: https://archive.ics.uci.edu/ml/datasets/Iris
- [3] Wikipedia 'Sepal',
  Online: https://en.wikipedia.org/wiki/Sepal
- [4] Rattle Library for R,
  Online: http://rattle.togaware.com/
- [5] B2Share, 'Iris Dataset LibSVM Format Preprocessing',
  Online: https://b2share.eudat.eu/records/37fb24847a73489a9c569d7033ad0238
- [6] An Introduction to Statistical Learning with Applications in R,
  Online: http://www-bcf.usc.edu/~gareth/ISL/index.html
- [7] F. Rosenblatt, 'The Perceptron--a perceiving and recognizing automaton',
  Report 85-460-1, Cornell Aeronautical Laboratory, 1957
- [8] Rosenblatt,'The Perceptron: A probabilistic model for information storage and orgainzation in the brain',
  Psychological Review 65(6), pp. 386-408, 1958
- [9] PLA Algorithm, YouTube Video,
  Online:
- [10] C. Shearer, CRISP-DM model, Journal Data Warehousing, 5:13
- [11] Pete Chapman, 'CRISP-DM User Guide', 1999,
  Online: http://lyle.smu.edu/~mhd/8331f03/crisp.pdf

Lecture Bibliography (2)

- [11] Introduction to Data Mining, Pang-Ning Tan, Michael Steinbach, Vipin Kumar, Addison Wesley, ISBN 0321321367, English, ~769 pages, 2005
- [12] Udacity, 'Overfitting',
  Online: https://www.youtube.com/watch?v=CxAxRCv9WoA
- [13] Wikipedia on 'Statistical Learning Theory',
  Online: http://en.wikipedia.org/wiki/Statistical_learning_theory
- [14] Leslie G. Valiant, '*A Theory of the Learnable*', Communications of the ACM 27(11):1134–1142, 1984,
  Online: https://people.mpi-inf.mpg.de/~mehlhorn/SeminarEvolvability/ValiantLearnable.pdf
- [15] Y.S. Abu-Mostafa, M. Magdon-Ismail, Hsuan-Tien Lin, 'Learning from Data – A short course',
  AML Book, ISBN 978-1-60049-006-4
- [16] Big Data Tips, 'Gradient Descent',
  Online: http://www.big-data.tips/gradient-descent
- [17] Keras Python Deep Learning Library,
  Online: https://keras.io/
- [18] YouTube Video, 'Neural Networks, A Simple Explanation',
  Online: http://www.youtube.com/watch?v=gcK_5x2KsLA
- [19] YouTube Video, "What is Remote Sensing?"
  Online: https://www.youtube.com/watch?v=nU-CjAKry5c
- [20] E. Kim, 'Everything You Wanted to Know about the Kernel Trick',
  Online: http://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html

Lecture Bibliography (3)

- [21] Visualization of high-dimensional space,
  Online: http://i.imgur.com/WuxyO.png
- [22] Morris Riedel, 'Introduction to Machine Learning Algorithms', Invited YouTube Lecture, six lectures University of Ghent, 2017
  Online: https://www.youtube.com/watch?v=KgiuUZ3WeP8&list=PLrmNhuZo9sgbcWtMGN0i6G9HEvh08JG0J
- [23] DEEP Projects Web Page,
  Online: http://www.deep-projects.eu/
- [24] Keras Python Deep Learning Library,
  Online: https://keras.io/
- [25] Tensorflow Deep Learning Framework,
  Online: https://www.tensorflow.org/
- [26] A Tour of Tensorflow,
  Online: https://arxiv.org/pdf/1610.01178.pdf
- [27] K. Hwang, G. C. Fox, J. J. Dongarra, 'Distributed and Cloud Computing', Book,
  Online: http://store.elsevier.com/product.jsp?locale=en_EU&isbn=9780128002049
- [28] YouTube Lecture Series MIT 6.S191, 'Introduction to Deep Learning'
  Online: https://www.youtube.com/watch?v=JN6H4rQvwgY

Slides Available at http://www.morrisriedel.de/talks