Summer School ML

V Non-Linear Models

Prof. Dr.-Ing. Janis Keuper

KI Labor Südbaden

INSTITUTE FOR MACHINE LEARNING AND ANALYTICS

Hochschule Offenburg
offenburg.university

Introduction to ML

Basic Types of Machine Learning Algorithms

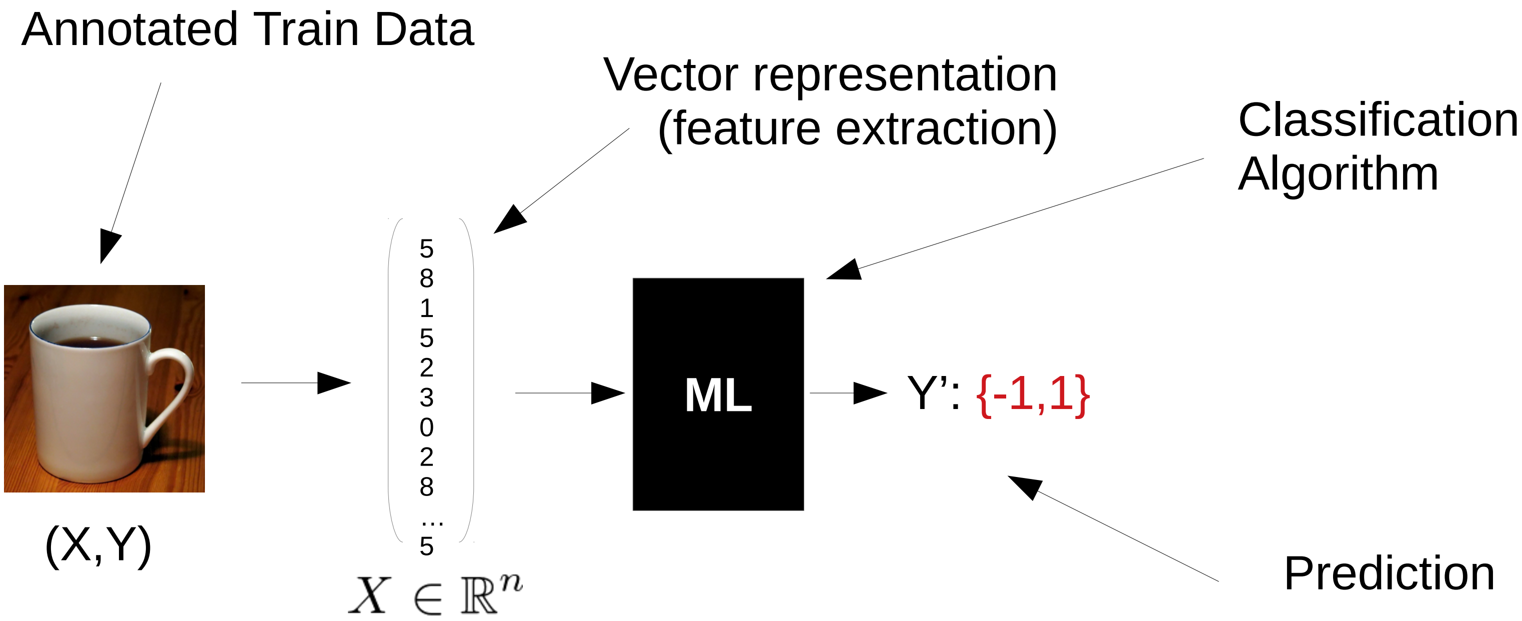
- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

- Labeled data
- Direct and quantitative evaluation

- Learn model from „ground truth“ examples
- Predict unseen examples

Recall: Classification

Supervised Learning: Annotated Training Data



Recall: Classification

LEARNING: is a optimization problem → Finding the best function separating

$$min(\|f(X,Y), Y'\|)$$

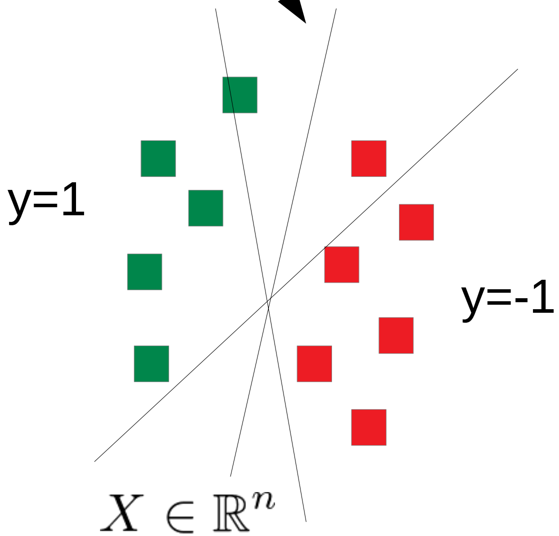
(X,Y)

$$\begin{pmatrix} 5 \\ 8 \\ 1 \\ 5 \\ 2 \\ 3 \\ 0 \\ 2 \\ 8 \\ \dots \\ 5 \end{pmatrix}$$

$X \in \mathbb{R}^n$

ML

Y': {-1,1}



Recall: Linear Classifier

A Simple Linear Model: binary classification

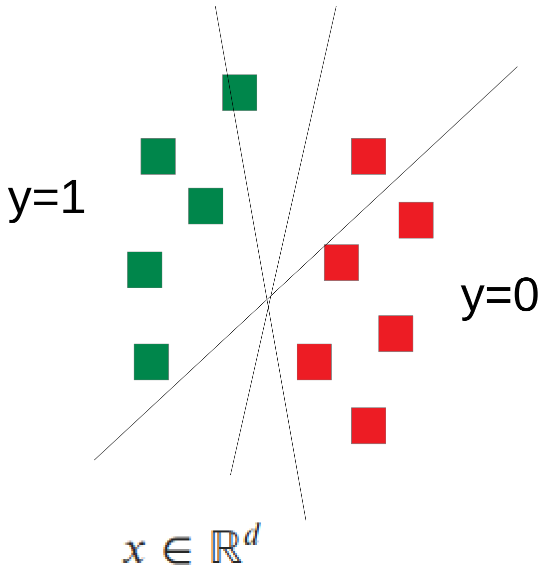
Parameterization of prediction function f with d-dimensional data as:

$$f(x) = y' = w^T x = \sum_{j=0}^{d} w_j x_j$$

With data samples $x \in \mathbb{R}^d$

Model parameters $w \in \mathbb{R}^d$

Model: hyper plane

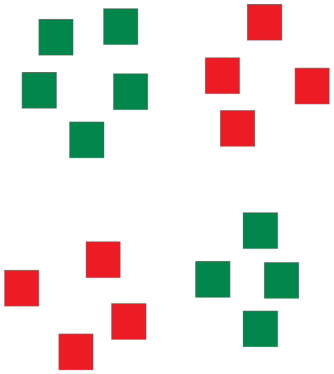


Limitations of Linear Models

(Obviously), linear models have limitations

Consider this very simple binary classification Example:

Simple counter example



$x \in \mathbb{R}^d$

How to separate "green" from "red" with a linear Model (= hyper plane)?

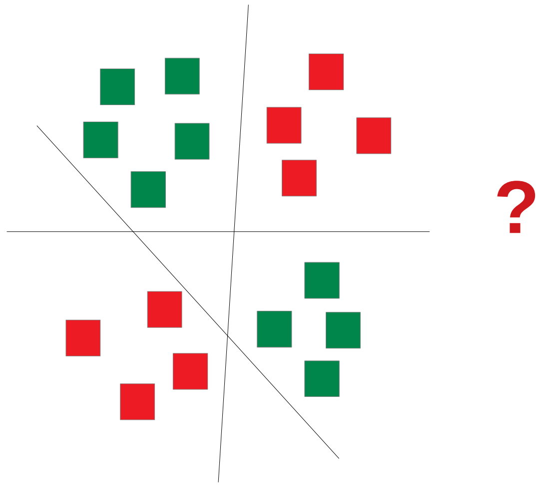
Limitations of Linear Models

(Obviously), linear models have limitations

Consider this very simple binary classification Example:

Simple counter example

How to separate "green" from "red" with a linear Model (= hyper plane)?



$x \in \mathbb{R}^d$

Limitations of Linear Models

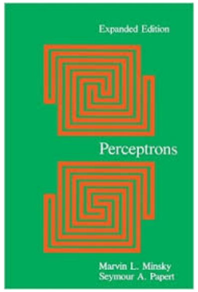
(Obviously), linear models have limitations

Consider this very simple binary classification Example:

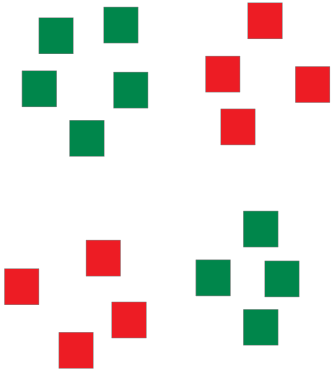
Simple counter example

→ **known as "X-Or" Problem**

→ **one reason for the so-called "AI Winter"**



Caused by the Minsky book
On the shortcomings of the
First neural networks...

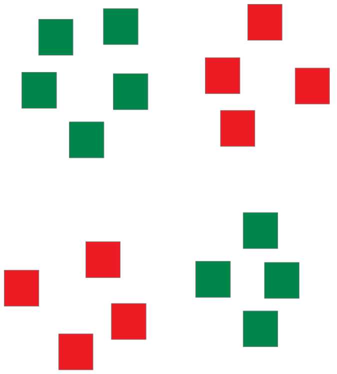
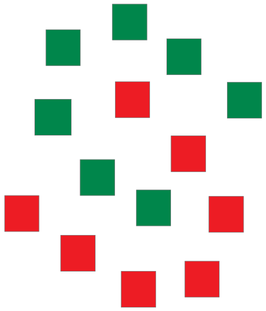
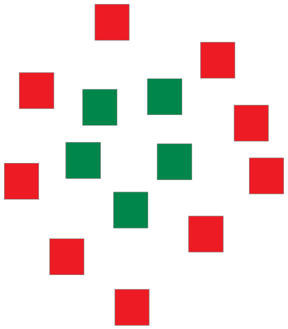
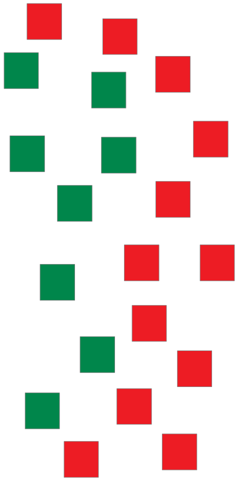


$x \in \mathbb{R}^d$

Limitations of Linear Models

(Obviously), linear models have limitations

More simple (binary 2D) examples:

Limitations of Linear Models

KI Labor Südbaden

Why are linear models working at all?

Limitations of Linear Models

KI Labor Südbaden

Why are linear model working at all?

→ very high dimensional feature spaces often time allow linear models to Separate the data

→ very simple (linear) model even can be of advantage in theses settings:

- **“curse of dimensionality”**
- **Avoid overfitting**

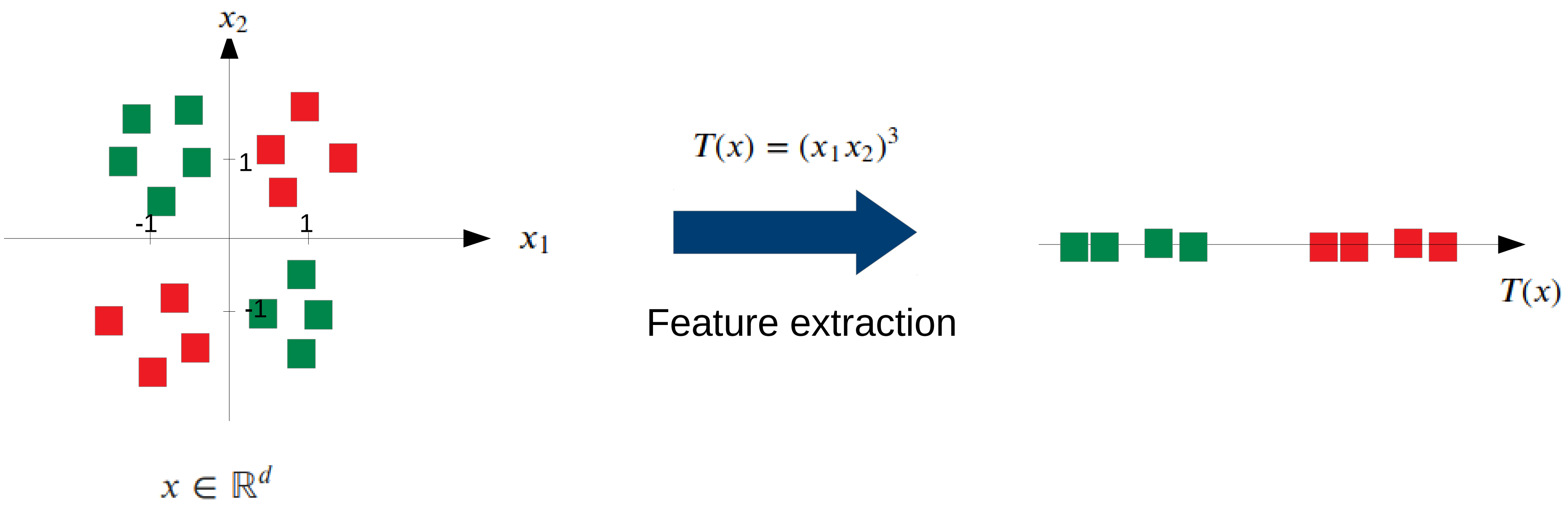
Adding non-linearity

Three canonical ways (we already saw two of them):

Adding non-linearity

Three canonical ways:

1. extracting non-linear features:

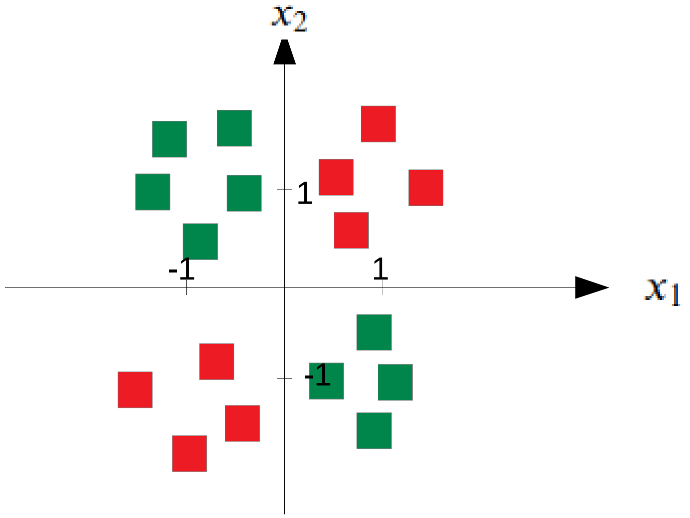


Adding non-linearity

Three canonical ways:

1. extracting non-linear features:

NOTE: just an example, usually not
That simple for real data.



$x \in \mathbb{R}^d$

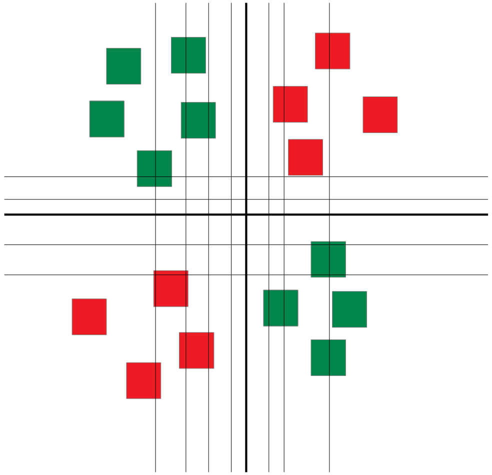
$T(x) = (x_1 x_2)^3$

Feature extraction

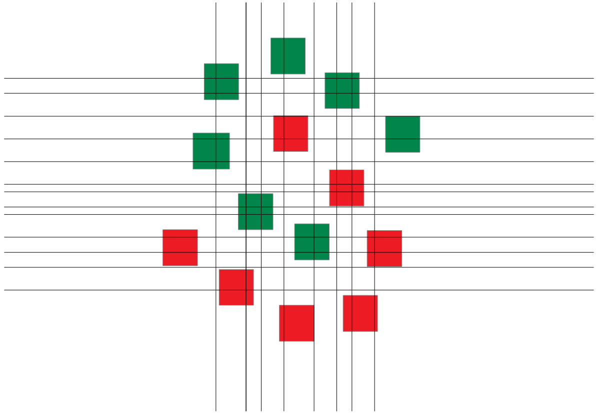
Adding non-linearity

Three canonical ways:

2. use ensembles of linear models (like Random Forrest)



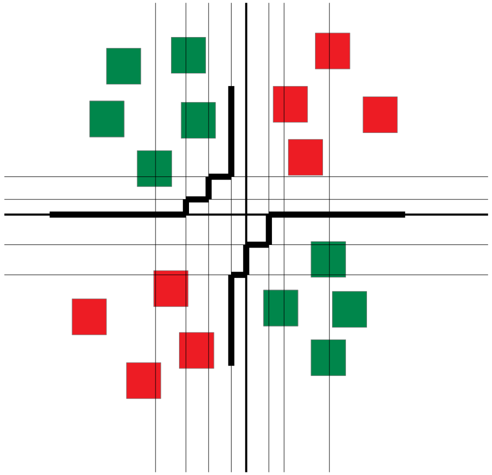
$x \in \mathbb{R}^d$



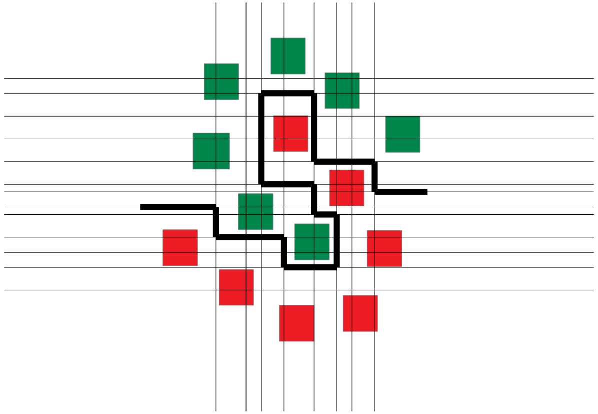
Adding non-linearity

Three canonical ways:

2. use ensembles of linear models → approximation of non-linear models by piece-wise linear models



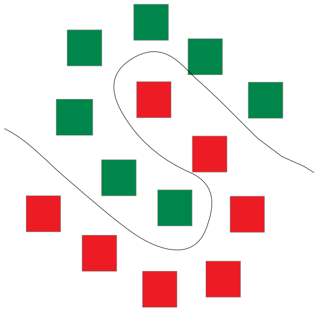
$x \in \mathbb{R}^d$



Adding non-linearity

Three canonical ways:

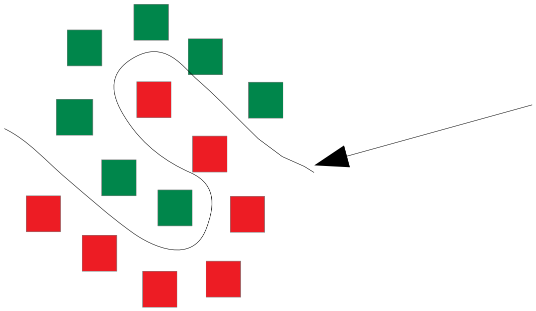
3. use non-linear functions



Adding non-linearity

Three canonical ways:

3. use non-linear functions



How to parameterize the non-linear model?

Adding non-linearity

Adding non-linearity to our simple linear classifier

$$f(x) = y' = w^T x = \sum_{j=0}^{d} w_j x_j \xrightarrow{\text{Train by solving optimization}} \arg\min_{w} \sum_{i=0}^{N} L(y_i, w^T x_i)$$

$$\Downarrow$$

$$f(x) = \phi(w^T x)$$

Step I: add a very simple element-wise
non-linear mapping.

(like in the previous feature extraction example)

Adding non-linearity

Adding non-linearity to our simple linear classifier

$$f(x) = y' = w^T x = \sum_{j=0}^{d} w_j x_j$$

$$f(x) = \phi(w^T x)$$

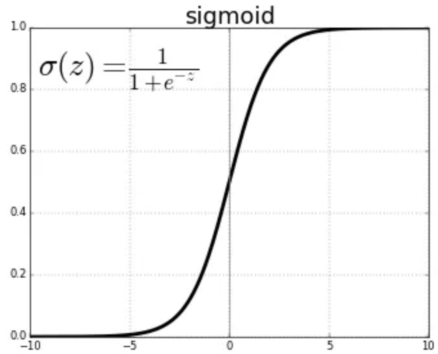
What are good choices for these functions?

Adding non-linearity

Adding non-linearity to our simple linear classifier

$$f(x) = y' = w^T x = \sum_{j=0}^{d} w_j x_j$$

$$f(x) = \phi(w^T x)$$



What are good choices for these functions?

Properties:

- Between 0 and 1 → pseudo probability interpretation
- Stable range of output → gradient optimization

Common choices:

- **Sigmoid function**
- Tanh
- ...

Adding non-linearity

Adding non-linearity to our simple linear classifier

$$f(x) = y' = w^T x = \sum_{j=0}^{d} w_j x_j$$

$$f(x) = \phi_3(w_3 \phi_2(W_2 \phi_1(W_1 x)))$$

W are now Matrices to produce vector outputs

Step II: concatenate several of these operations

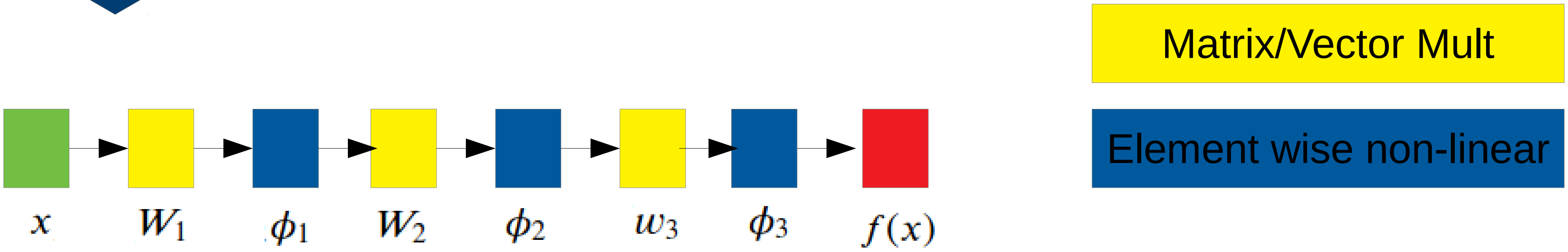
(like we do in the ensemble approach)

Adding non-linearity

Let's display this in a slightly different way (no change in math formulation!)

$$f(x) = y' = w^T x = \sum_{j=0}^{d} w_j x_j$$

$$f(x) = \phi_3(w_3\phi_2(W_2\phi_1(W_1 x)))$$

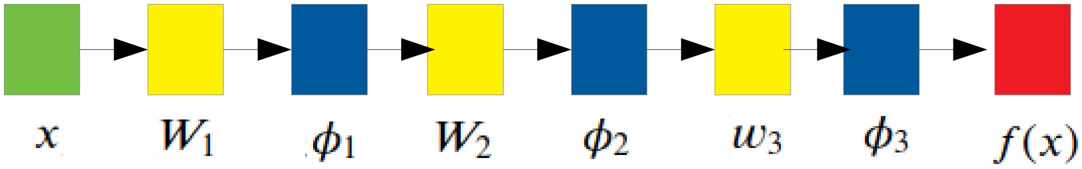


Adding non-linearity

Let's display this in a slightly different way (no change in math formulation!)

$$f(x) = y' = w^T x = \sum_{j=0}^{d} w_j x_j$$

$$f(x) = \phi_3(w_3 \phi_2(W_2 \phi_1(W_1 x)))$$



Note: there is a theoretical prove that we need only two concatenations to approximate any smooth function if the W are large enough!

Matrix/Vector Mult

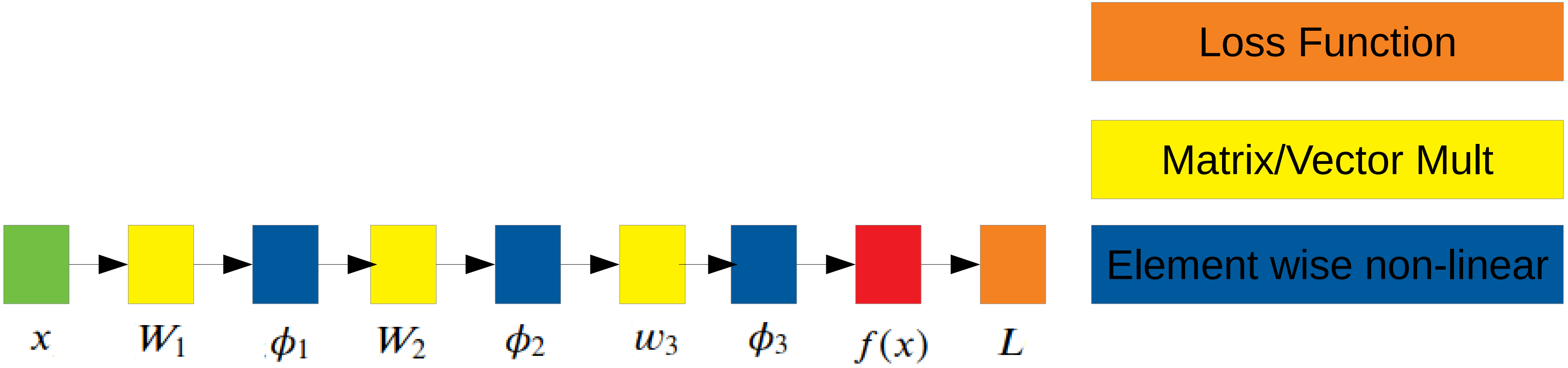
Element wise non-linear

Adding non-linearity

For training (optimization), we need to add loss function

→ **same approach as in the linear case:**

$$\arg\min_{w} \sum_{i=0}^{N} L(y_i, f(x))$$



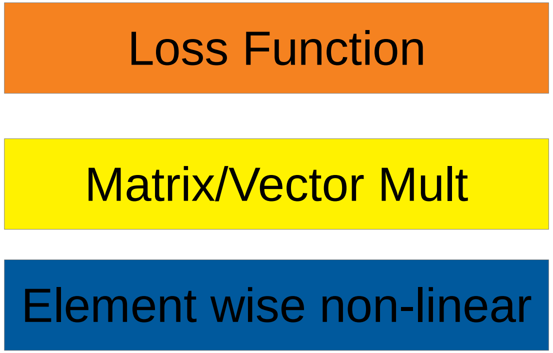
Adding non-linearity

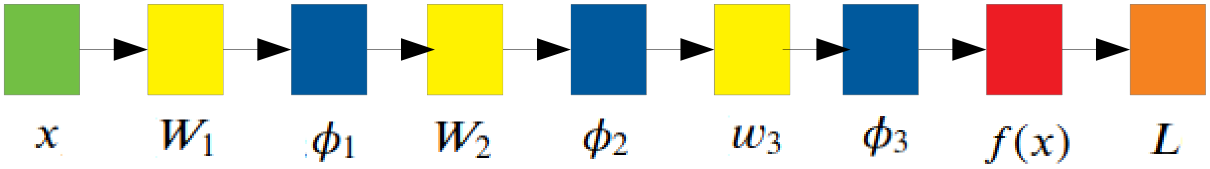
For training (optimization), we need to add loss function

→ **same approach as in the linear case:**

$$\arg\min_{w} \sum_{i=0}^{N} L(y_i, f(x))$$

→ **solve by gradient descent optimization**





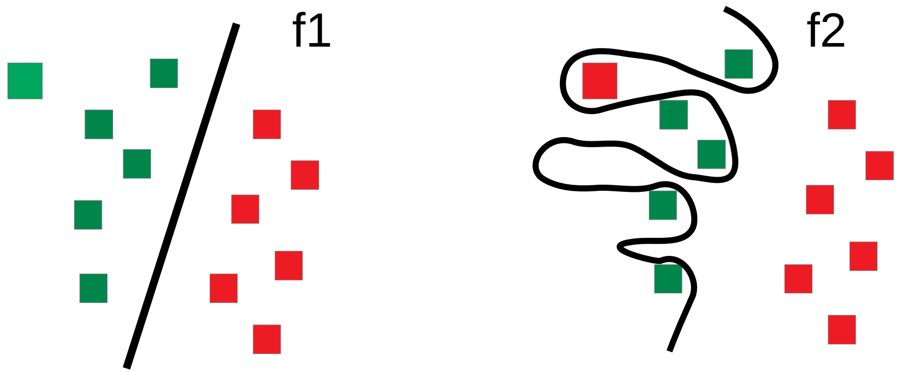
Adding non-linearity

Recall **OVERFITTING**

Model “to close” to train data

With non-linear model much more likely to happen in practice.

→ **we need to work against this...**



Adding non-linearity

Adding regularization term to the Loss function

→ **here L2 regularization:**

$$\arg\min_{w} \sum_{i=0}^{N} L(y_i, f(x)) + \lambda \sum_j w_j^2$$

All parameters to be learned

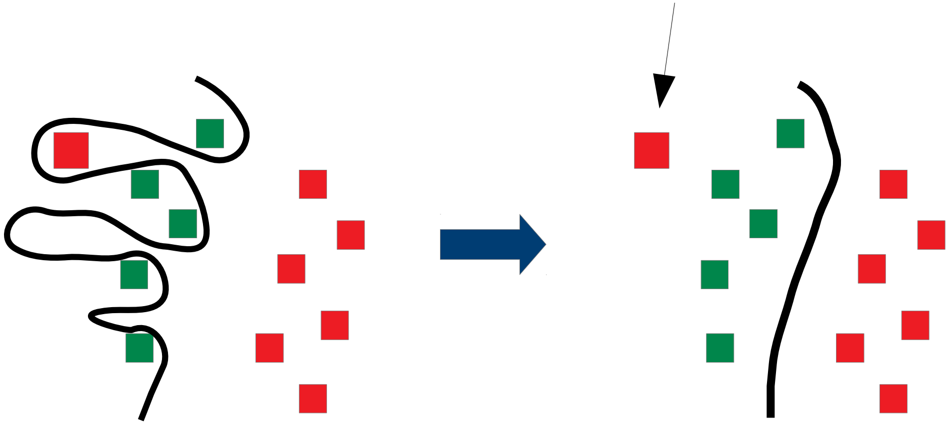
Scalar hyper parameter: impact of regularization

Adding non-linearity

Adding regularization term to the Loss function

→ **here L2 regularization:**

$$\arg\min_{w} \sum_{i=0}^{N} L(y_i, f(x)) + \lambda \sum_j w_j^2$$



→ **L1 regularization:**

$$\arg\min_{w} \sum_{i=0}^{N} L(y_i, f(x)) + \lambda \sum_j |w_j|$$

Regularization will punish high parameter values
→ smoother model
→ training errors allowed !

Support Vector Machines

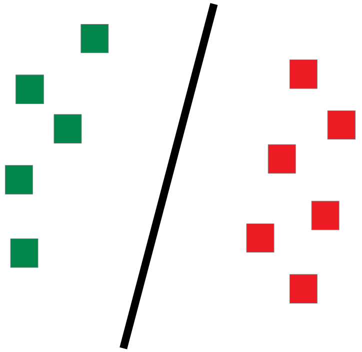
Support Vector Machines

- Invented in the mid 90s by Vapnik
- Classification and Regression
- State of the Art ML Algorithm of the pre Deep Learning era

Support Vector Machines

- Invented in the mid 90s by Vapnik
- Classification and Regression
- State of the Art ML Algorithm of the pre Deep Learning era

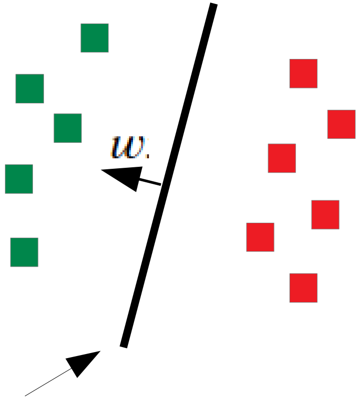
- **Basic model:**
  - Support only two classes {-1,1}
  - Linear classification



Support Vector Machines

- Invented in the mid 90s by Vapnik
- Classification and Regression
- State of the Art ML Algorithm of the pre Deep Learning era

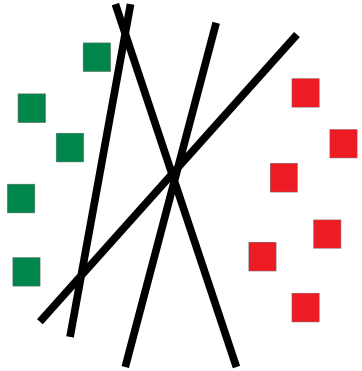
- **Basic model:**
  - Support only two classes {-1,1}
  - Linear classification



Parameterization: $wx - b = 0$

Support Vector Machines

What is the difference compared to previous formulations?

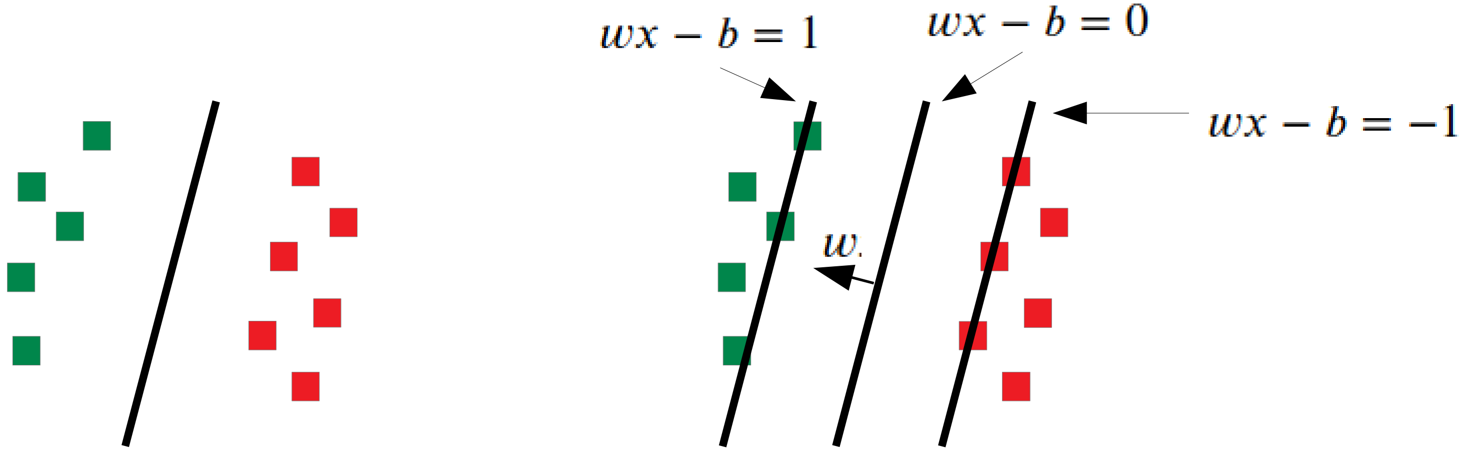


Standard linear model:
→ loss only on accuracy
→ many solutions

$$\arg\min_{w} \sum_{i=0}^{N} L(y_i, w^T x_i)$$

Support Vector Machines

What is the difference compared to previous formulations?



Standard linear model:
→ loss only on accuracy
→ many solutions

New optimization problem
→ "Max Margin": $\frac{2}{\|w\|}$

→ only one solution, convex optimization problem

Support Vector Machines

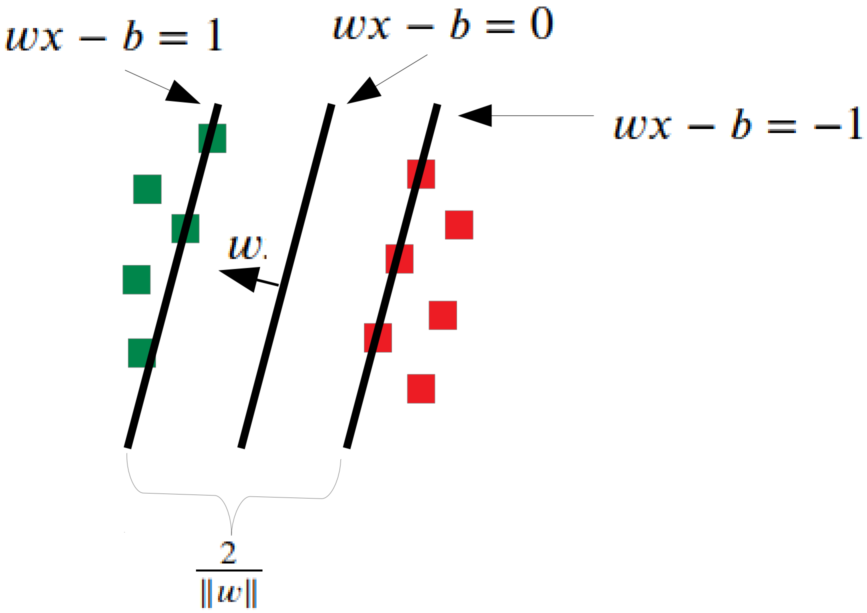
What is the difference compared to previous formulations?

New optimization problem
→ maximize “Margin”:
→ equals minimizing the uncertainty

$$\arg\min_{w} \sum_{i=0}^{N} \xi_i + \lambda \|w_i\|^2$$

subject to $y_i(w \cdot x_i - b) \geq 1 - \zeta_i$ and $\zeta_i \geq 0$, for all i.

$$\zeta_i = \max\left(0, 1 - y_i(w \cdot x_i - b)\right)$$



Support Vector Machines

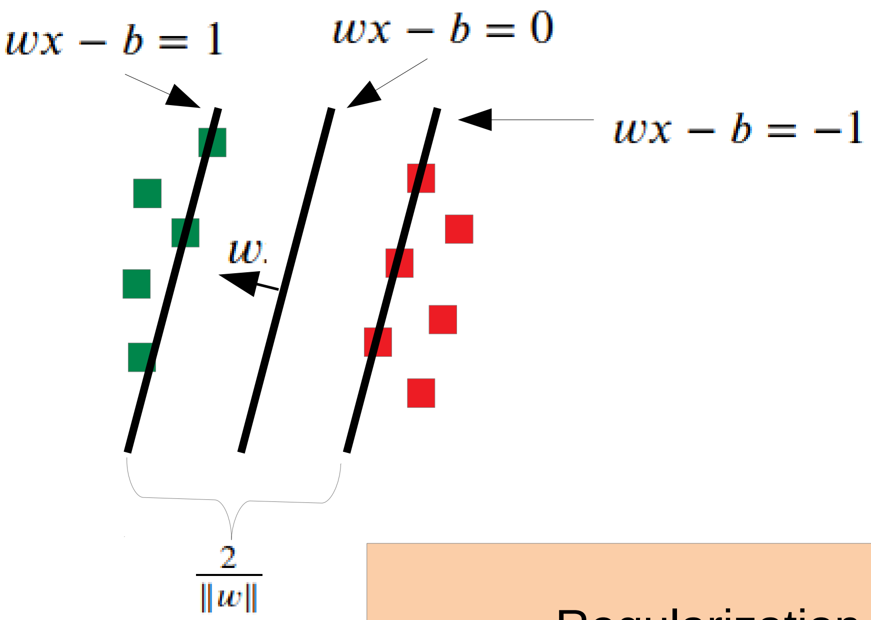
What is the difference compared to previous formulations?

New optimization problem
→ maximize "Margin":
→ equals minimizing the uncertainty

$$\arg\min_{w} \sum_{i=0}^{N} \xi_i + \boxed{\lambda \|w_i\|^2}$$

subject to $y_i(w \cdot x_i - b) \geq 1 - \zeta_i$ and $\zeta_i \geq 0$, for all i.

$$\zeta_i = \max\left(0, 1 - y_i(w \cdot x_i - b)\right)$$



Regularization

Support Vector Machines

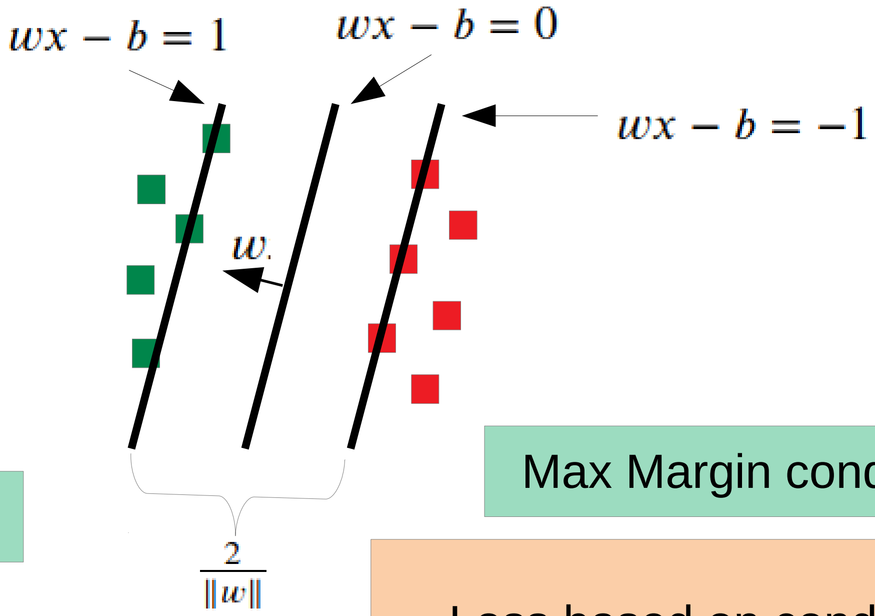
What is the difference compared to previous formulations?

New optimization problem
→ maximize “Margin”:
→ equals minimizing the uncertainty

$$\arg\min_{w} \sum_{i=0}^{N} \xi_i + \lambda \|w_i\|^2$$

subject to $y_i(w \cdot x_i - b) \geq 1 - \zeta_i$ and $\zeta_i \geq 0$, for all i.

$$\zeta_i = \max\left(0, 1 - y_i(w \cdot x_i - b)\right)$$



Max Margin conditions

Loss based on conditions

Support Vector Machines

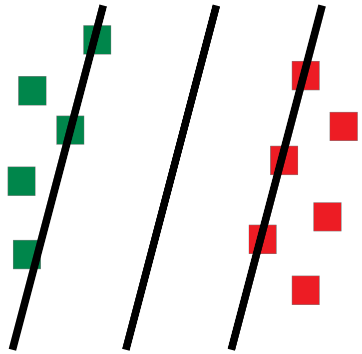
Dual Formulation of the optimization Problem

Via Lagrange dual function, leads to quadratic optimization problem (convex)

$$\vec{w} = \sum_{i=1}^{n} c_i y_i \vec{x}_i$$

$$\text{maximize } f(c_1 \ldots c_n) = \sum_{i=1}^{n} c_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i c_i (x_i \cdot x_j) y_j c_j,$$

$$\text{subject to } \sum_{i=1}^{n} c_i y_i = 0, \text{ and } 0 \leq c_i \leq \frac{1}{2n\lambda} \text{ for all } i.$$



Support Vector Machines

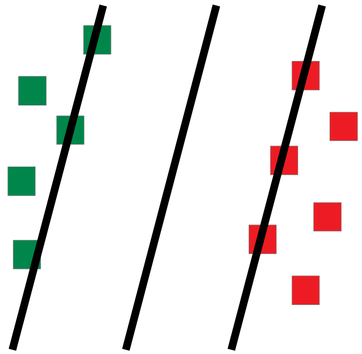
Dual Formulation of the optimization Problem

Via Lagrange dual function, leads to quadratic optimization problem (convex)

$$\boxed{\vec{w} = \sum_{i=1}^{n} c_i y_i \vec{x}_i}$$

$$\text{maximize } f(c_1 \dots c_n) = \sum_{i=1}^{n} c_i - \boxed{\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i c_i (x_i \cdot x_j) y_j c_j,}$$

$$\text{subject to } \sum_{i=1}^{n} c_i y_i = 0, \text{ and } 0 \leq c_i \leq \frac{1}{2n\lambda} \text{ for all } i.$$



Re-write model as linear combination of all weighted (c) data points

Support Vector Machines

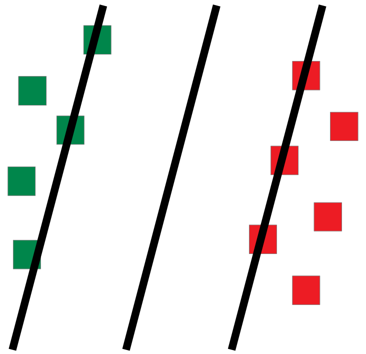
Dual Formulation of the optimization Problem

Via Lagrange dual function, leads to quadratic optimization problem (convex)

$$\vec{w} = \sum_{i=1}^{n} c_i y_i \vec{x}_i$$

$$\text{maximize } f(c_1 \ldots c_n) = \sum_{i=1}^{n} c_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i c_i (x_i \cdot x_j) y_j c_j,$$

$$\text{subject to } \sum_{i=1}^{n} c_i y_i = 0, \text{ and } 0 \leq c_i \leq \frac{1}{2n\lambda} \text{ for all } i.$$



Regularization

Re-write model as linear combination of all weighted (c) data points

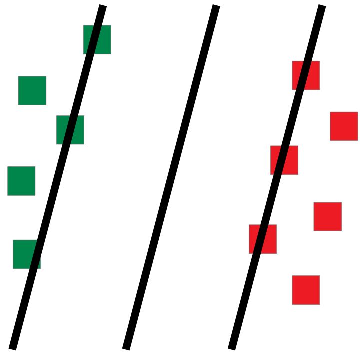
Support Vector Machines

How many Data Points do we need to define a (hyper) plane?

$$\vec{w} = \sum_{i=1}^{n} c_i y_i \vec{x}_i$$

$$\text{maximize } f(c_1 \ldots c_n) = \sum_{i=1}^{n} c_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i c_i (x_i \cdot x_j) y_j c_j,$$

$$\text{subject to } \sum_{i=1}^{n} c_i y_i = 0, \text{ and } 0 \leq c_i \leq \frac{1}{2n\lambda} \text{ for all } i.$$



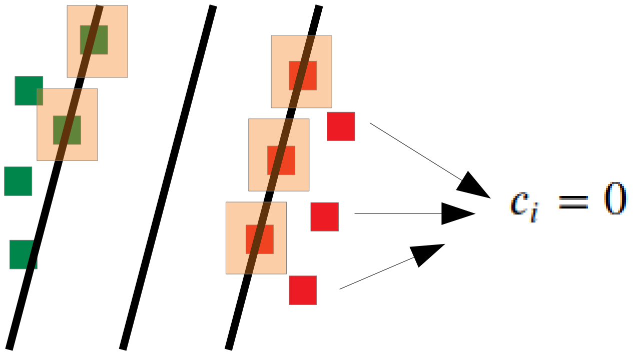
Support Vector Machines

How many Data Points do we need to define a (hyper) plane?

$$\vec{w} = \sum_{i=1}^{n} c_i y_i \vec{x}_i$$

$$\text{maximize } f(c_1 \ldots c_n) = \sum_{i=1}^{n} c_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i c_i (x_i \cdot x_j) y_j c_j,$$

$$\text{subject to } \sum_{i=1}^{n} c_i y_i = 0, \text{ and } 0 \leq c_i \leq \frac{1}{2n\lambda} \text{ for all } i.$$



Most point will not contribute.
Only “**Support Vectors**” will.

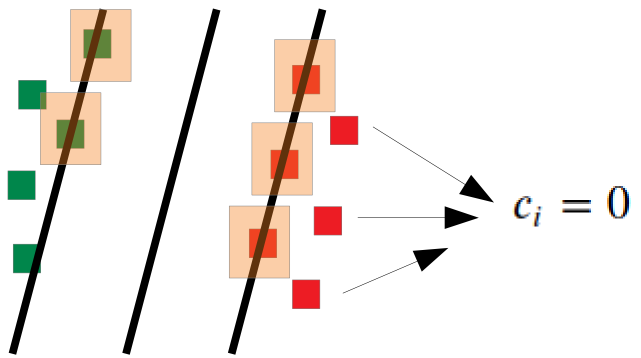
Support Vector Machines

SVM Model: All Support Vectors

$$\vec{w} = \boxed{\sum_{i=1}^{n} c_i y_i \vec{x}_i}$$

$$\text{maximize } f(c_1 \ldots c_n) = \sum_{i=1}^{n} c_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i c_i (x_i \cdot x_j) y_j c_j,$$

$$\text{subject to } \sum_{i=1}^{n} c_i y_i = 0, \text{ and } 0 \leq c_i \leq \frac{1}{2n\lambda} \text{ for all } i.$$



Most point will not contribute.
Only “**Support Vectors**” will.

Support Vector Machines

Non Linear SVMs:

→ follow same strategy as before and add simple non-linear function

At the formulation of the model normal:

$$\vec{w} = \sum_{i=1}^{n} c_i y_i \vec{x}_i \quad \Rightarrow \quad \vec{w} = \sum_{i=1}^{n} c_i y_i \varphi(\vec{x}_i),$$

Support Vector Machines

Non Linear SVMs:

→ follow same strategy as before and add simple non-linear function

At the formulation of the model normal:

$$\vec{w} = \sum_{i=1}^{n} c_i y_i \vec{x}_i \quad \Rightarrow \quad \vec{w} = \sum_{i=1}^{n} c_i y_i \varphi(\vec{x}_i),$$

Insert into dual formulation

$$\text{maximize } f(c_1 \ldots c_n) = \sum_{i=1}^{n} c_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i c_i (\varphi(\vec{x}_i) \cdot \varphi(\vec{x}_j)) y_j c_j$$

Support Vector Machines

“Kernel Trick”: replace explicit non-linear function by *kernel*

$$k(\vec{x}_i, \vec{x}_j) = \varphi(\vec{x}_i) \cdot \varphi(\vec{x}_j)$$

Always a dot-product in some space

$$\text{maximize } f(c_1 \ldots c_n) = \sum_{i=1}^{n} c_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i c_i (\varphi(\vec{x}_i) \cdot \varphi(\vec{x}_j)) y_j c_j$$

Support Vector Machines

“Kernel Trick”: replace explicit non-linear function by *kernel*

$$k(\vec{x}_i, \vec{x}_j) = \varphi(\vec{x}_i) \cdot \varphi(\vec{x}_j)$$

re-write

$$\text{maximize } f(c_1 \ldots c_n) = \sum_{i=1}^{n} c_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i c_i (\varphi(\vec{x}_i) \cdot \varphi(\vec{x}_j)) y_j c_j$$

$$= \sum_{i=1}^{n} c_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i c_i k(\vec{x}_i, \vec{x}_j) y_j c_j$$

Popular Kernels:

Polynomial (of degree d)

$$k(\overrightarrow{x_i}, \overrightarrow{x_j}) = (\overrightarrow{x_i} \cdot \overrightarrow{x_j})^d$$

Gauss (or RBF)

$$k(\overrightarrow{x_i}, \overrightarrow{x_j}) = \exp(-\gamma \|\overrightarrow{x_i} - \overrightarrow{x_j}\|^2)$$

Support Vector Machines

KI Labor Südbaden

SVM Inference:

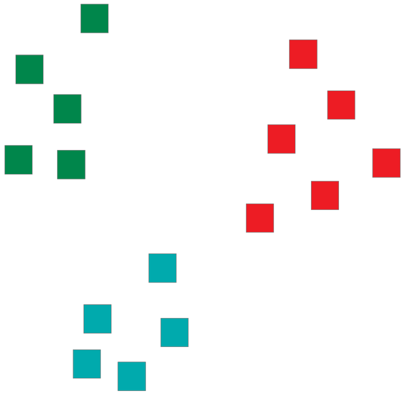
→ **evaluate kernel with all Support Vectors and take the sign**

$$\vec{z} \mapsto \operatorname{sgn}(\vec{w} \cdot \varphi(\vec{z}) - b) = \operatorname{sgn}\left(\left[\sum_{i=1}^{n} c_i y_i k(\vec{x}_i, \vec{z}) \right] - b \right).$$

$$\begin{aligned} b = \vec{w} \cdot \varphi(\vec{x}_i) - y_i &= \left[\sum_{j=1}^{n} c_j y_j \varphi(\vec{x}_j) \cdot \varphi(\vec{x}_i) \right] - y_i \\ &= \left[\sum_{j=1}^{n} c_j y_j k(\vec{x}_j, \vec{x}_i) \right] - y_i. \end{aligned}$$

Support Vector Machines

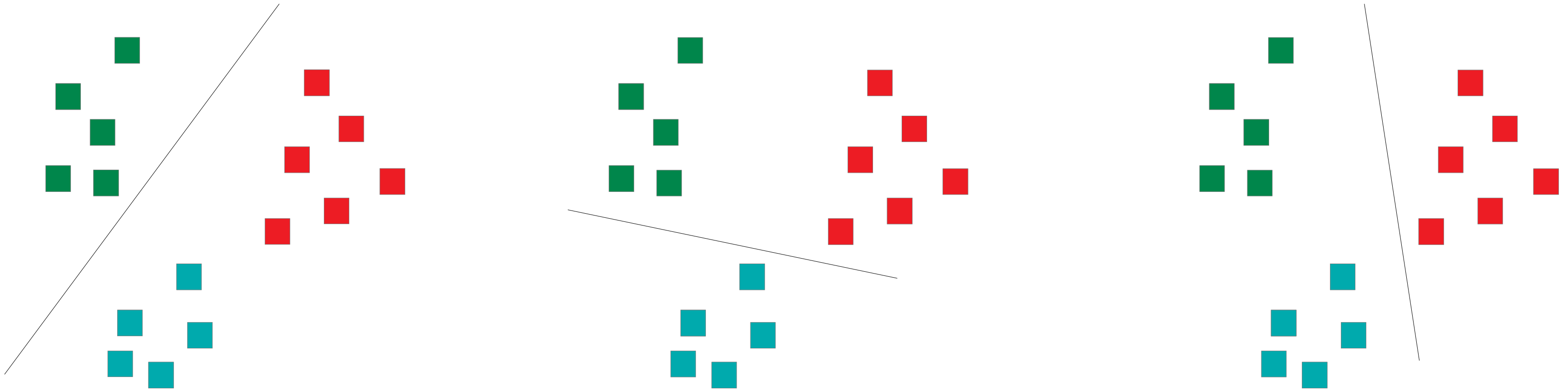
Multi class problems:



Support Vector Machines

KI Labor Südbaden

Multi class problems: 1-vs-Rest

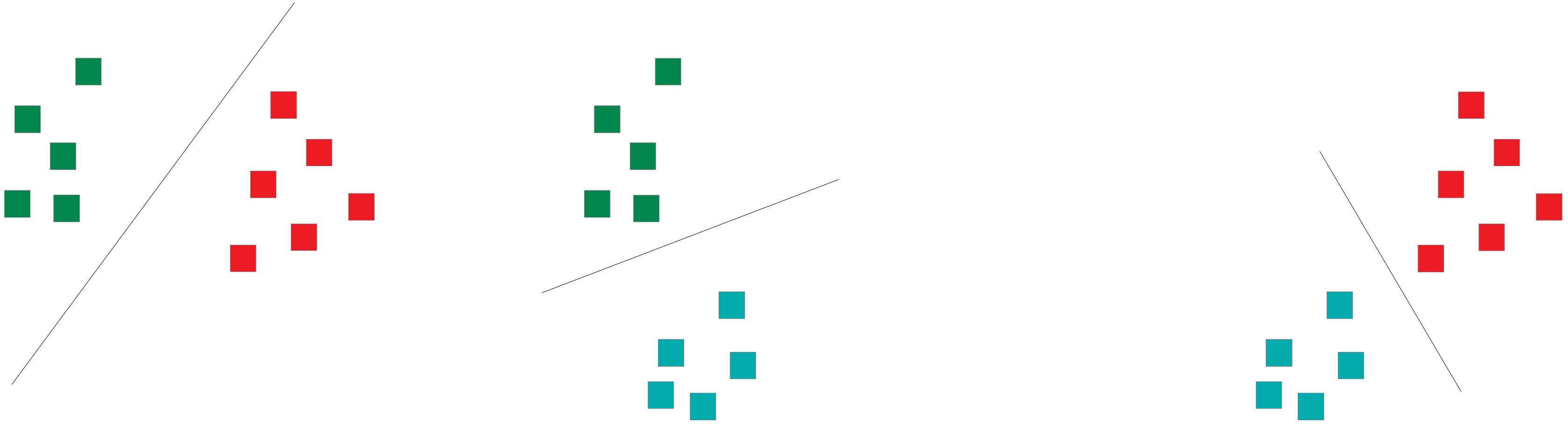


N models, take best.

Support Vector Machines

KI Labor Südbaden

Multi class problems: 1-vs-1



N(N-1)/2 models – tree execution, take best.

Model Selection

Motivation:

- Modeling a learning problem, we have many parameters to set:
  - Feature extraction algorithm
  - Feature selection and reduction
  - Choice of the learning algorithm
  - Parameters of the learning algorithm
  - ...
- **How to find the best model ?**

Model Selection

KI Labor Südbaden

How to find the best model ?

- We already have quality measures to compare models, like
  - Accuracy
  - F-Measure
  - ROC
  - …

- **But there are two problems:**
  - Do not over fit on the test data ($\rightarrow$ can't test too often to stay unbiased)
  - Computational complexity ($\rightarrow$ can't try every possible combination)

Cross-Validation

Tuning a model without test data

Simple approach:
n-fold cross validation

- Split train data into n parts
- Train on n-1 parts – test on the left out part
- Repeat n-times, leaving out a different part each time
- Average test results

Hyper-Parameter Optimization

KI Labor Südbaden

How to find the best model ?

- Grid-search over the parameter space
  - Very expensive
  - How to space the grid ?

- **Random Search**
  - Cheaper than grid-search
  - Quite effective

- Bayesian optimization
  - “optimal” next parameter set for testing

Discussion

Lab exercises coming up ...