

## Non-Linear Machine Learning

Janis Keuper

## Basic Types of Machine Learning Algorithms

**Supervised Learning**

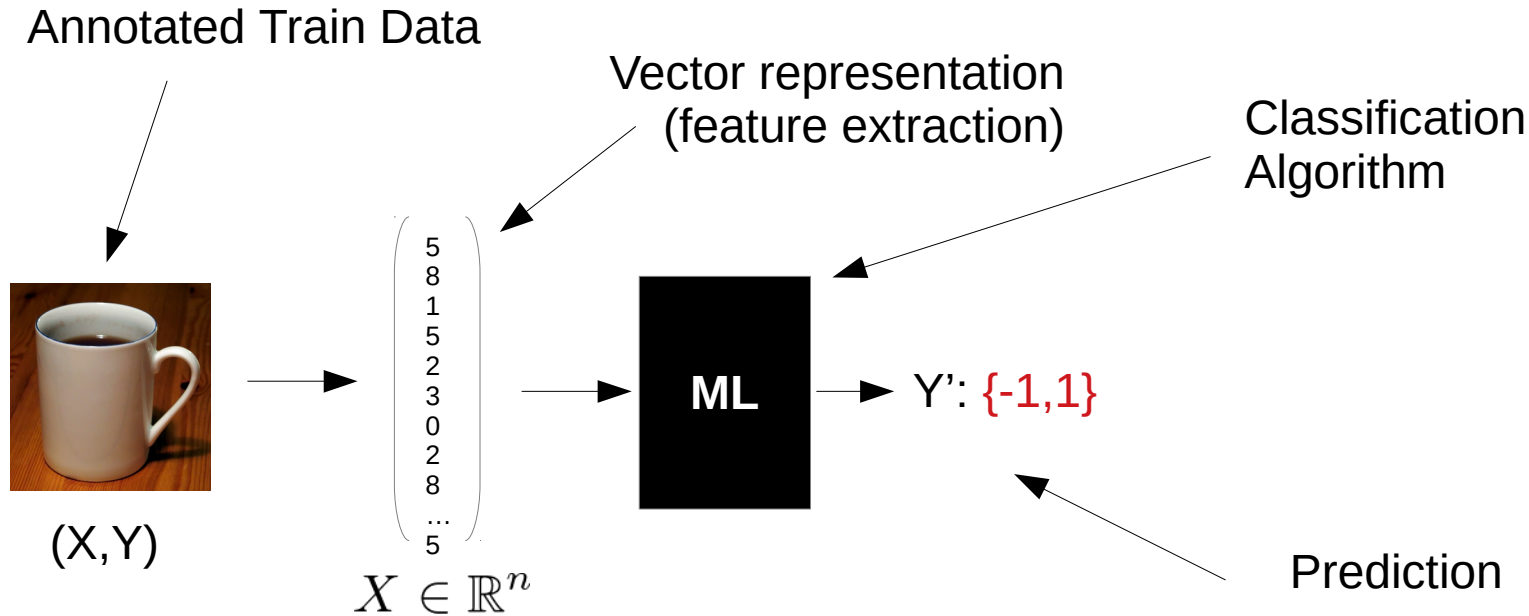
**Unsupervised Learning**

**Reinforcement Learning**

- Labeled data
- Direct and quantitative evaluation
- Learn model from „ground truth“ examples
- Predict unseen examples

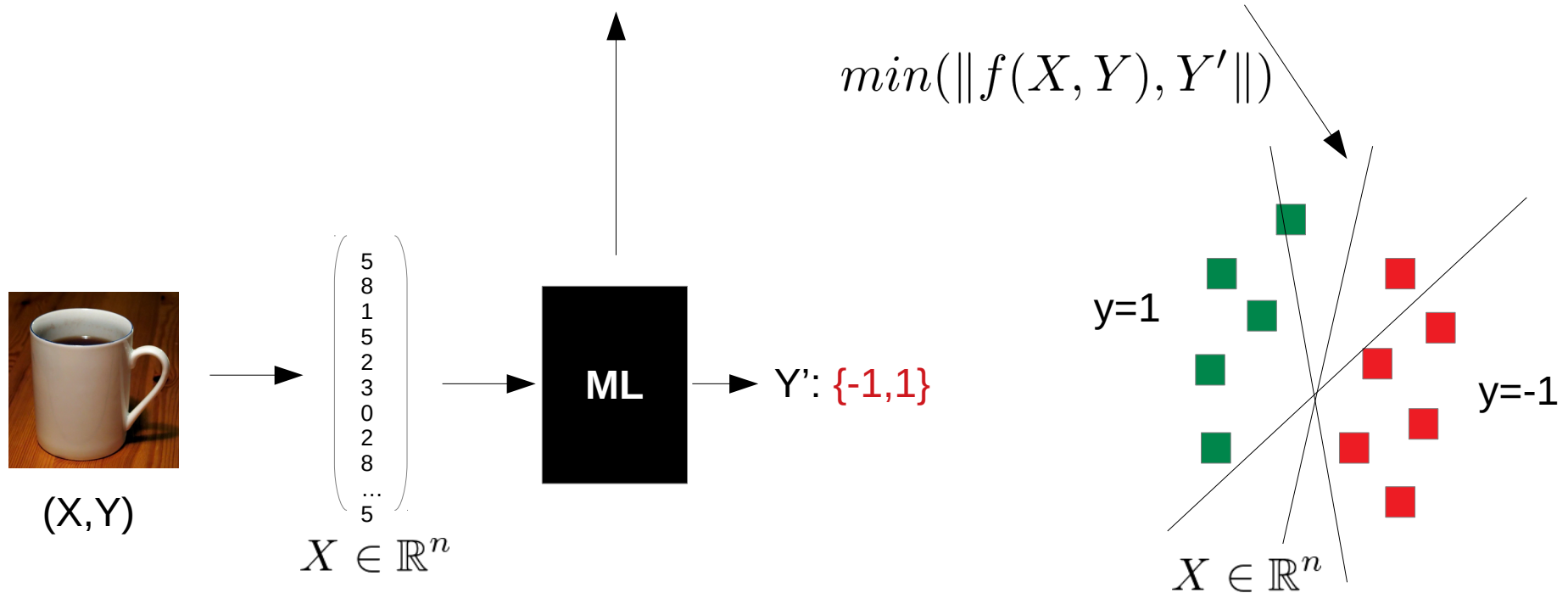
# Recall: Classification

## Supervised Learning: Annotated Training Data



# Recall: Classification

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



# Recall: Linear Classifier

A Simple Linear Model: **binary** classification

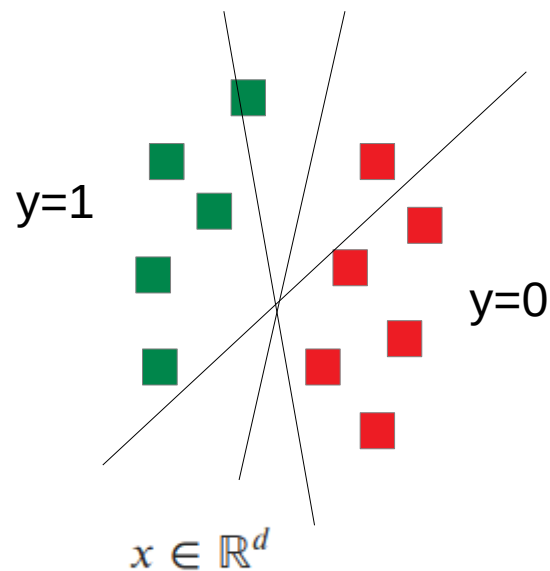
Parameterization of prediction function  $f$   
with  $d$ -dimensional data as:

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

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

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

Model: hyper plane





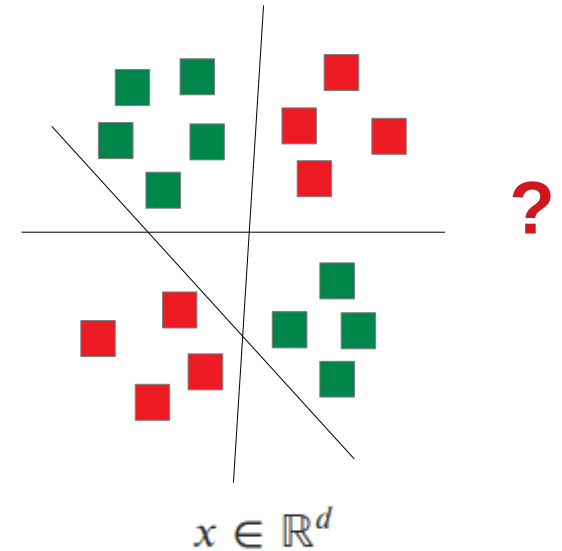
# Limitations of Linear Models

(Obviously), linear models have limitations

Consider this very simple binary classification  
Example:

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

Simple counter example



# Limitations of Linear Models

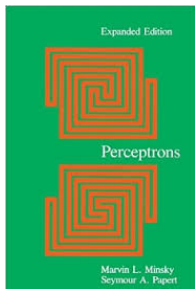
(Obviously), linear models have limitations

Consider this very simple binary classification  
Example:

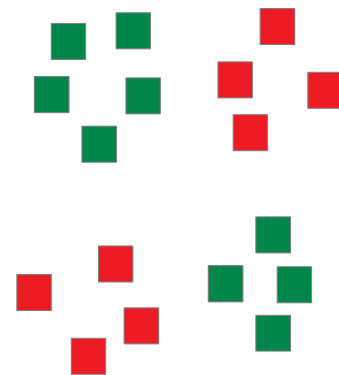
Simple counter example

→ 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...



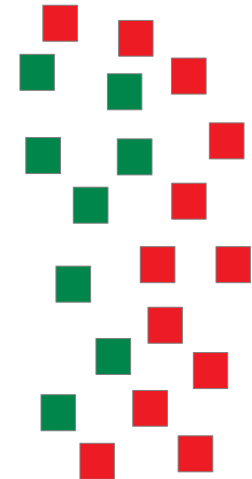
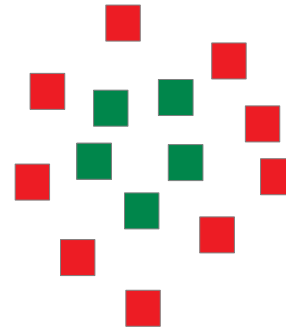
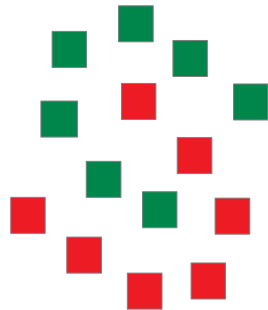
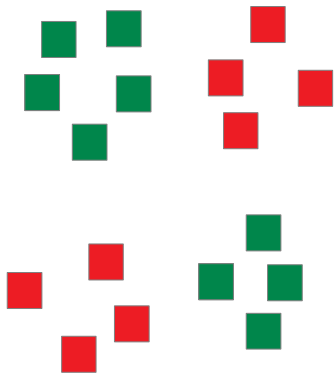
$x \in \mathbb{R}^d$



# Limitations of Linear Models

(Obviously), linear models have limitations

More simple (binary 2D) examples:



**Why are linear models working at all?**

**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

---

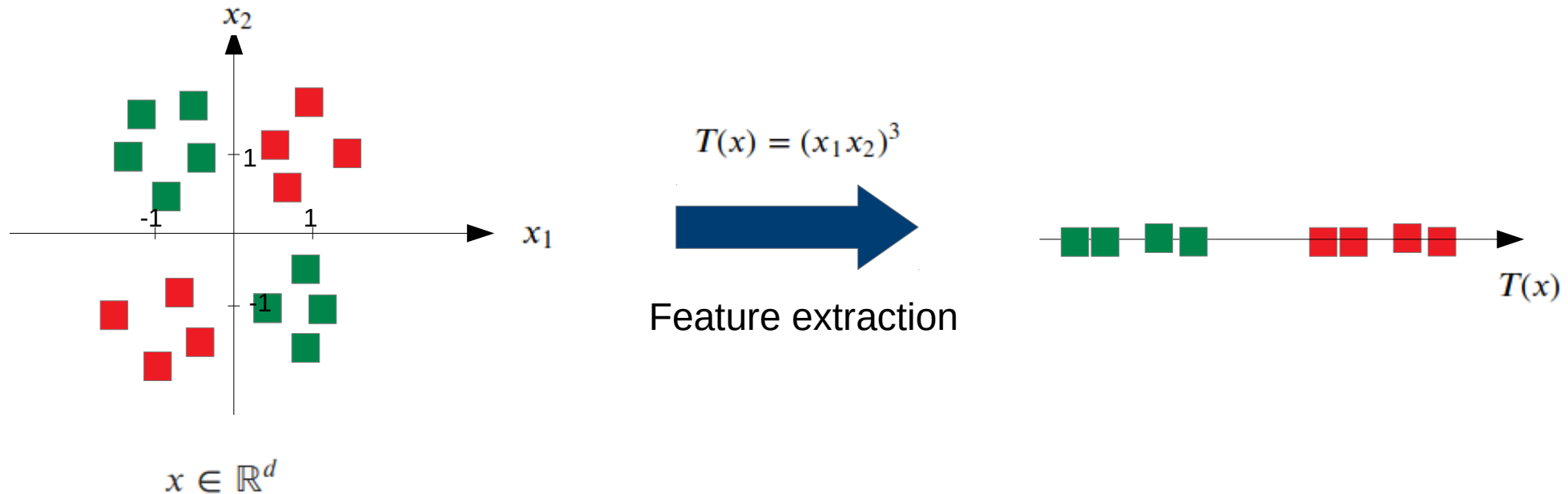


**Three canonical ways (we already saw two of them):**

# Adding non-linearity

Three canonical ways:

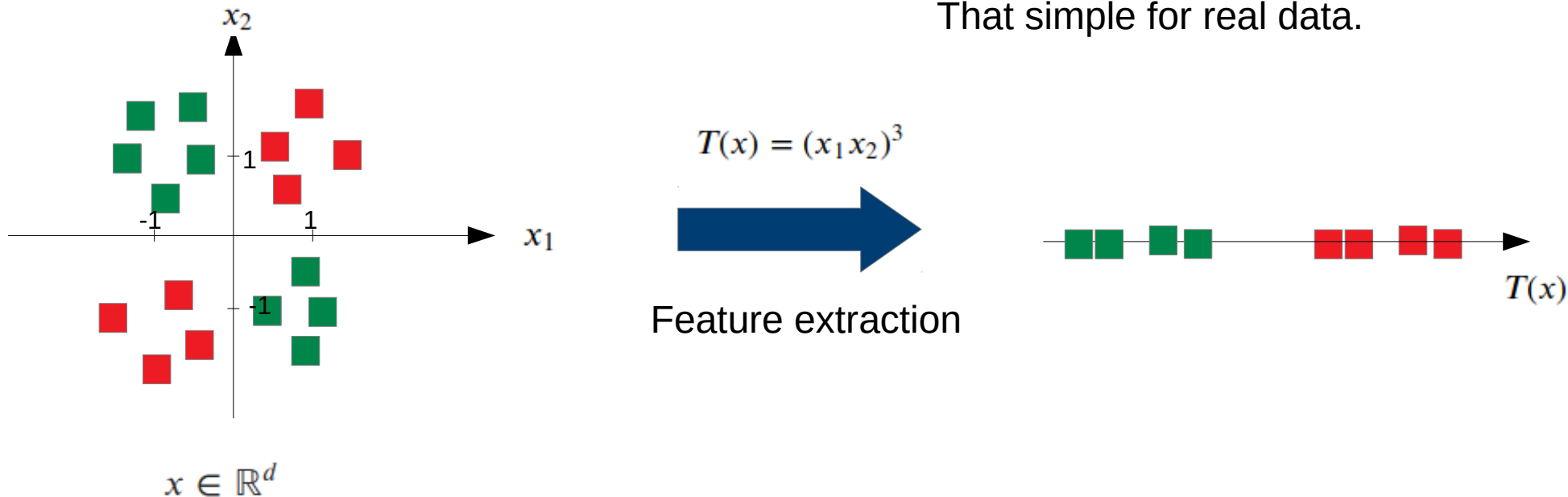
1. extracting non-linear features:



Three canonical ways:

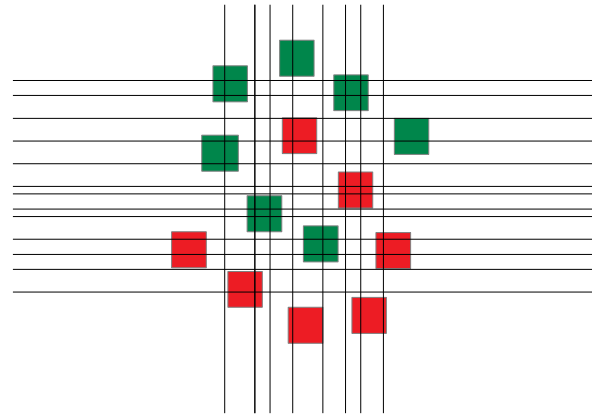
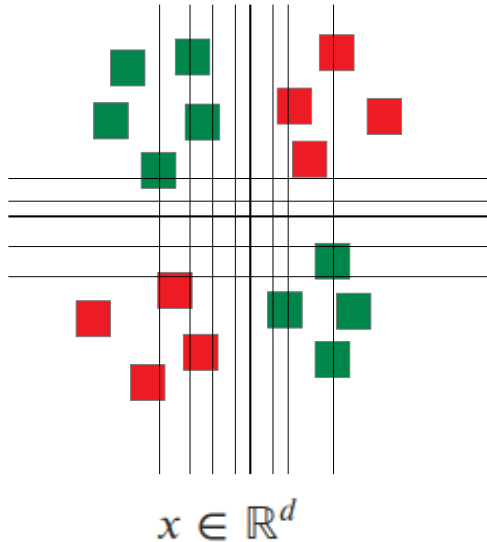
1. extracting non-linear features:

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



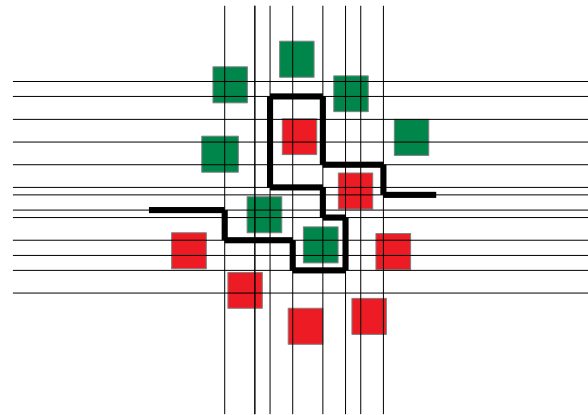
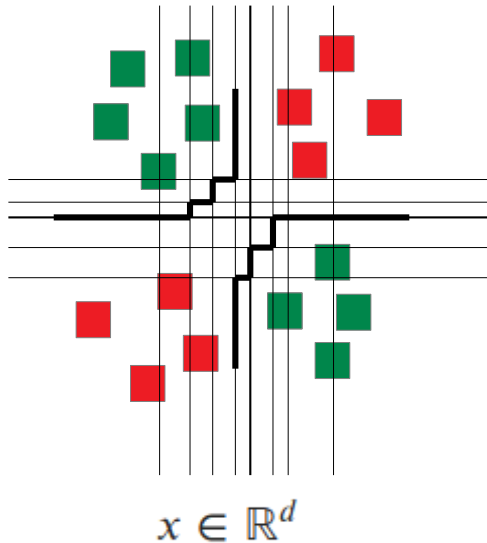
Three canonical ways:

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



Three canonical ways:

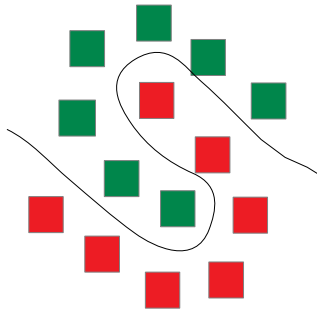
2. use ensembles of linear models → approximation of non-linear models by piece-wise linear models





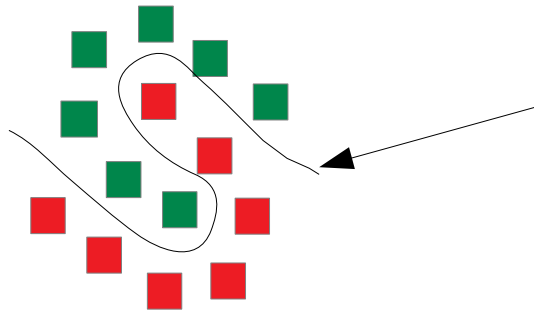
Three canonical ways:

3. use non-linear functions



Three canonical ways:

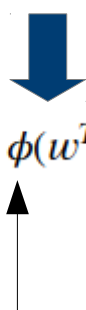
3. use non-linear functions



How to parameterize the non-linear model?

## Adding non-linearity to our simple linear classifier

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



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

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

(like in the previous feature extraction example)

## Adding non-linearity 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 to our simple linear classifier

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



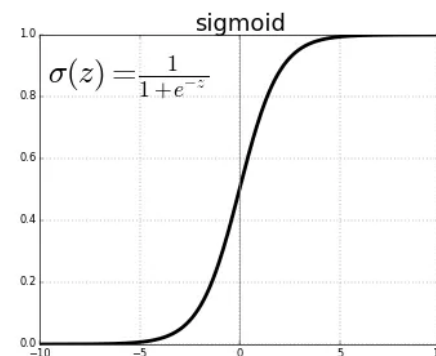
$$f(x) = \phi(w^T x)$$



What are good choices for these functions?

### Properties:

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



Common choices:

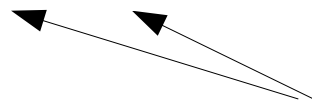
- **Sigmoid function**
- Tanh
- ...

## Adding non-linearity 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 )

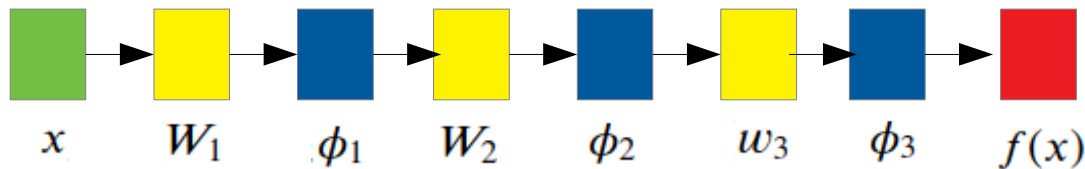
# Adding non-linearity

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

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



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



Matrix/Vector Mult

Element wise non-linear

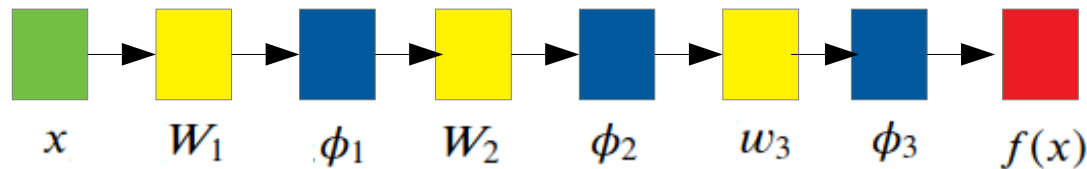
# Adding non-linearity

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

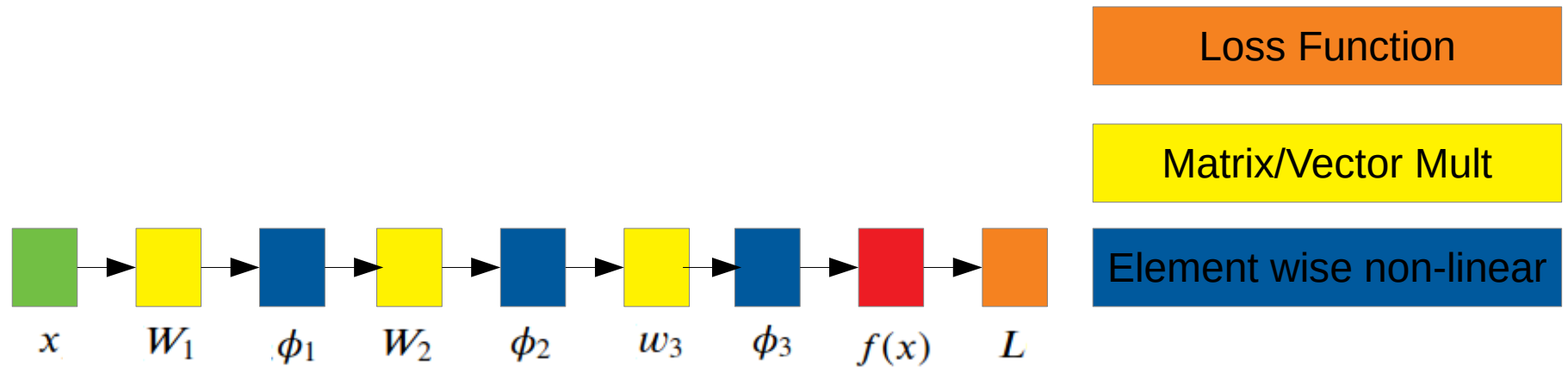


# Adding non-linearity

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

→ same approach as in the linear case:

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



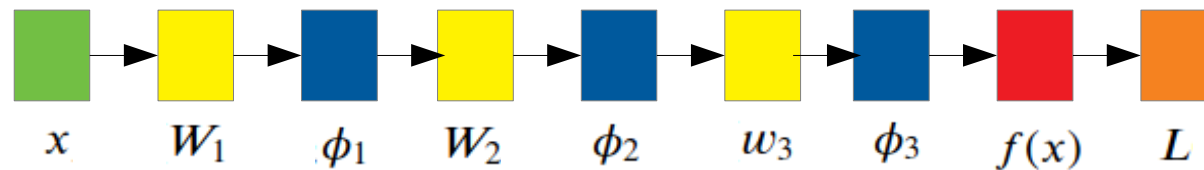
# Adding non-linearity

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



Loss Function

Matrix/Vector Mult

Element wise non-linear

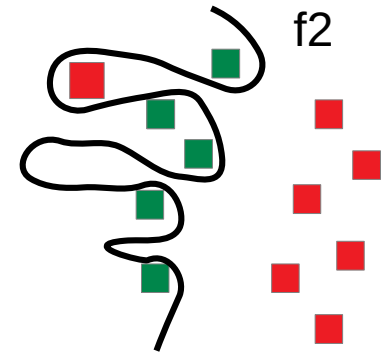
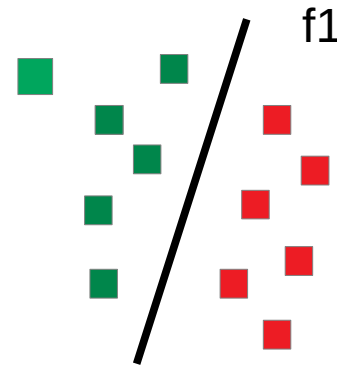
# Adding non-linearity

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 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

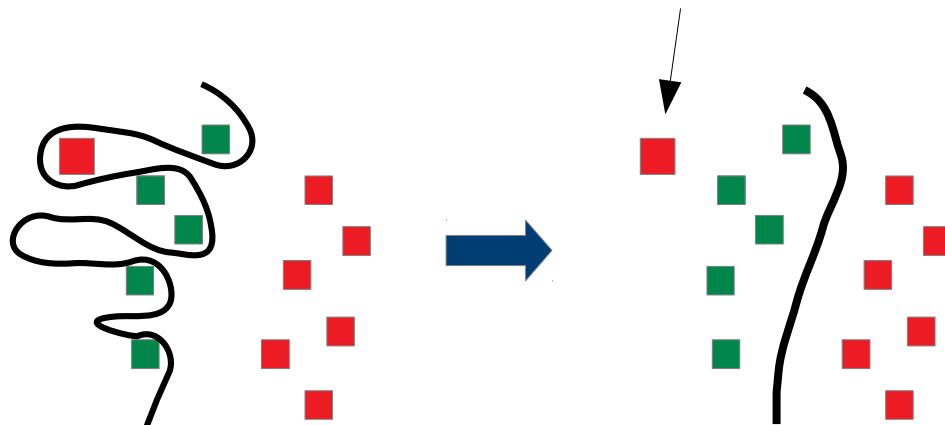
## Adding regularization term to the Loss function

→ 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

---



INSTITUTE FOR MACHINE  
LEARNING AND ANALYTICS

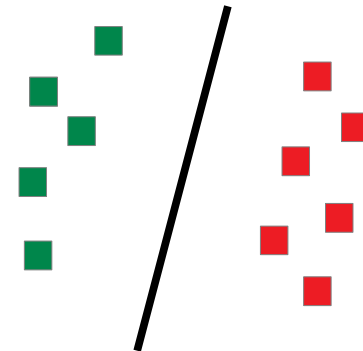
# Support Vector Machines

---



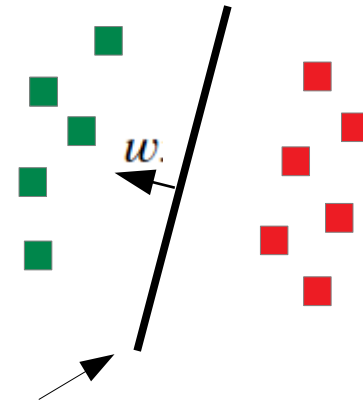
- Invented in the mid 90s by Vapnik
- Classification and Regression
- State of the Art ML Algorithm of the pre Deep Learning era

- 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



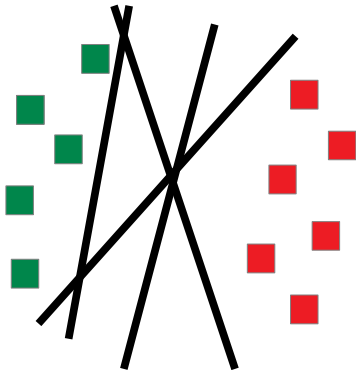


- Invented in the mid 90s by Vapnik
- Classification and Regression
- State of the Art ML Algorithm of the pre Deep Learning era
- **Basic model:**
  - Support only two classes  $\{-1,1\}$
  - Linear classification



Parameterization:  $w x - b = 0$

What is the difference compared to previous formulations?

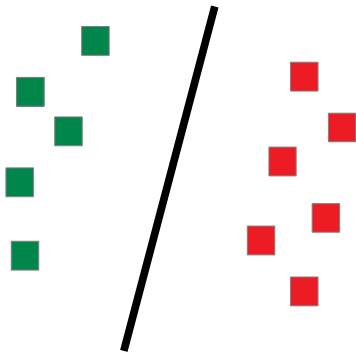


Standard linear model:

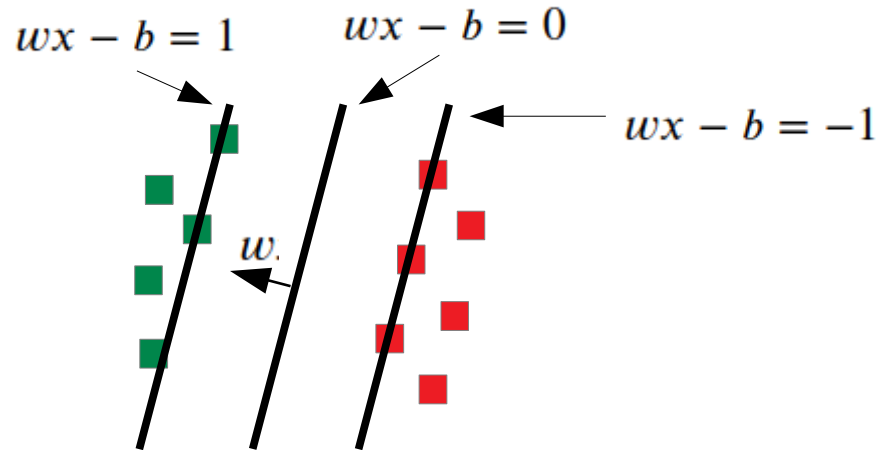
- loss only on accuracy
- many solutions

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

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

What is the difference compared to previous formulations?

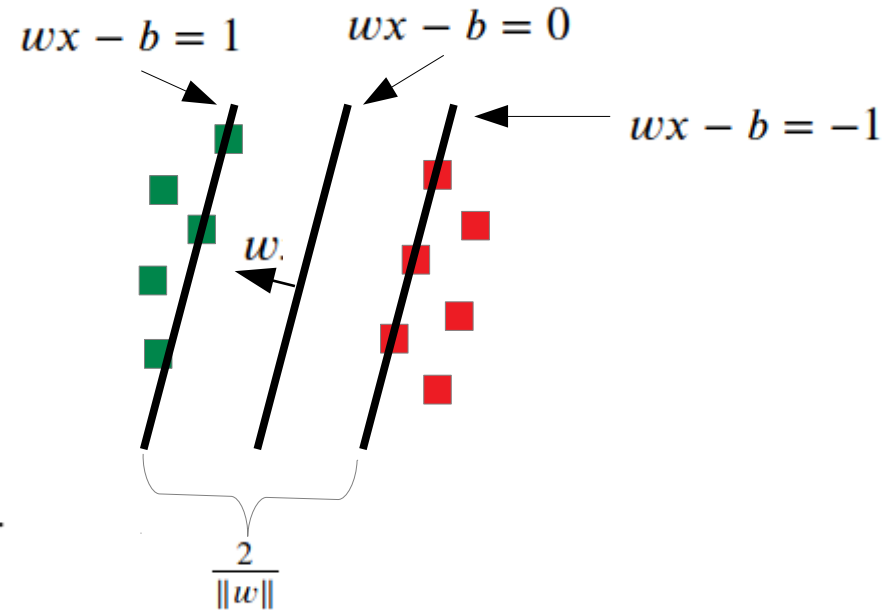
## New optimization problem

- maximize “Margin”:
- equals minimizing the uncertainty

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

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

$$\xi_i = \max(0, 1 - y_i(w \cdot x_i - b))$$



What is the difference compared to previous formulations?

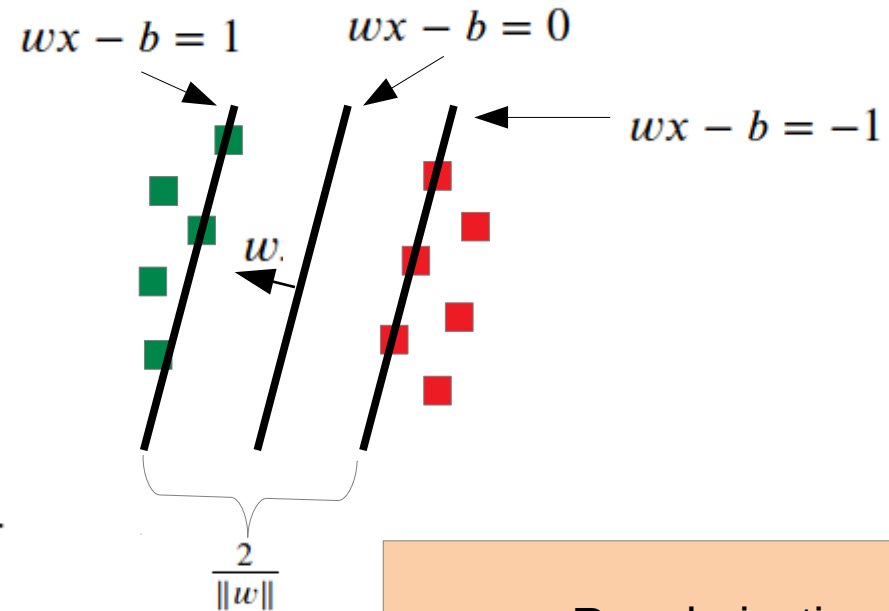
## New optimization problem

- maximize “Margin”:
- equals minimizing the uncertainty

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

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

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



Regularization

What is the difference compared to previous formulations?

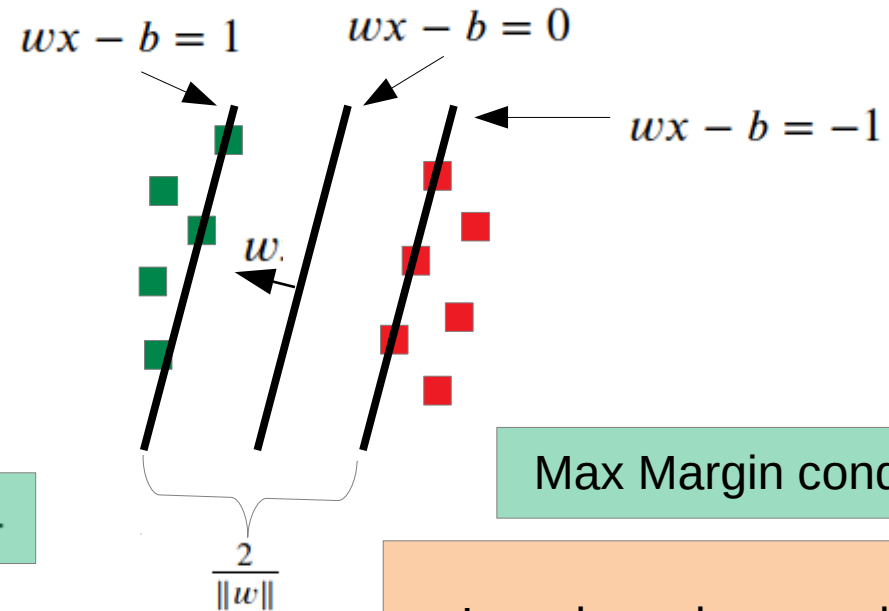
## New optimization problem

- 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  $i$ .

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



Max Margin conditions

Loss based on conditions

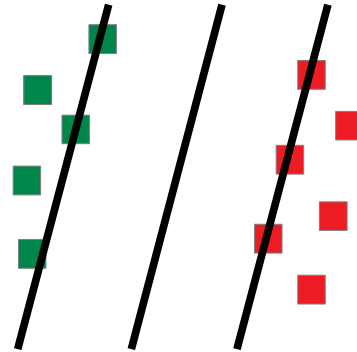
## Dual Formulation of the optimization Problem

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

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

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

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



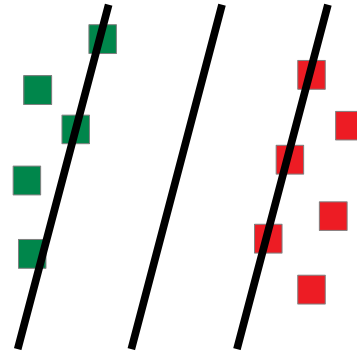
## Dual Formulation of the optimization Problem

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

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

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

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



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



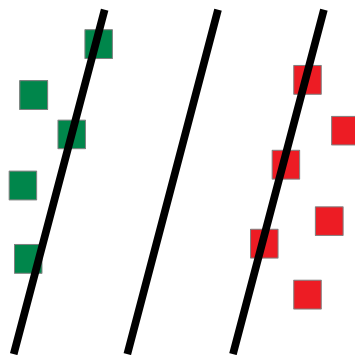
## Dual Formulation of the optimization Problem

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

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

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

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



Regularization

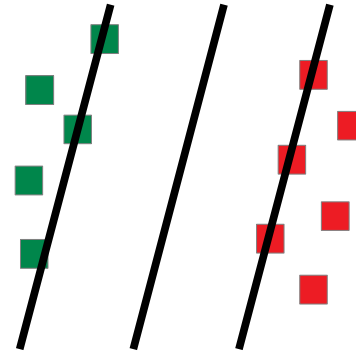
Re-write model as linear combination  
of all weighted (c) data points

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

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

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

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

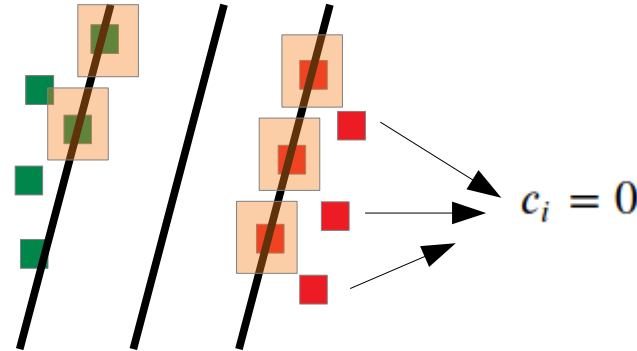


How many Data Points do we need to define a (hyper) plane?

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

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

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



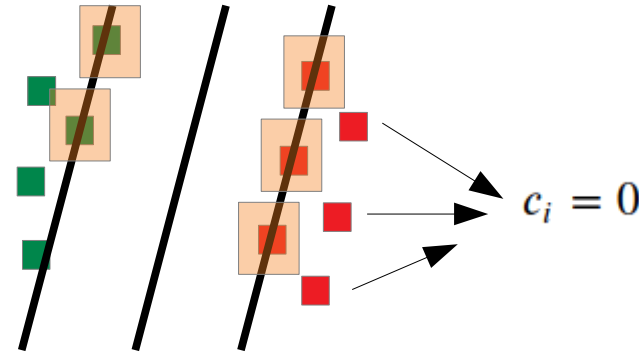
Most point will not contribute.  
Only “**Support Vectors**” will.

## SVM Model: All Support Vectors

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

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

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

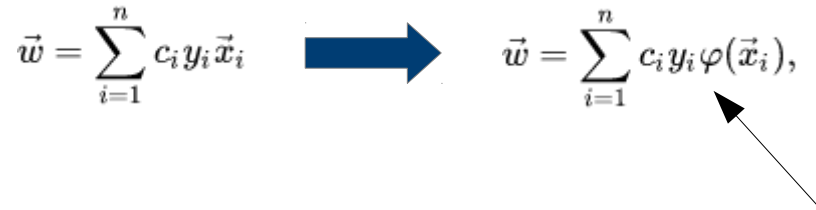


Most point will not contribute.  
Only “**Support Vectors**” will.

## Non Linear SVMs:

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

At the formulation of the model normal:

$$\vec{w} = \sum_{i=1}^n c_i y_i \vec{x}_i \quad \longrightarrow \quad \vec{w} = \sum_{i=1}^n c_i y_i \varphi(\vec{x}_i),$$
A large blue arrow points from the first equation to the second. A thin black arrow points from the phi symbol in the second equation to the right.

## Non Linear SVMs:

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

At the formulation of the model normal:

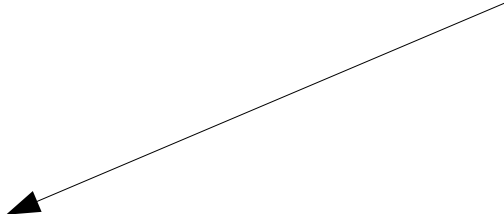
$$\vec{w} = \sum_{i=1}^n c_i y_i \vec{x}_i \quad \longrightarrow \quad \vec{w} = \sum_{i=1}^n c_i y_i \varphi(\vec{x}_i),$$



Insert into dual formulation

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

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

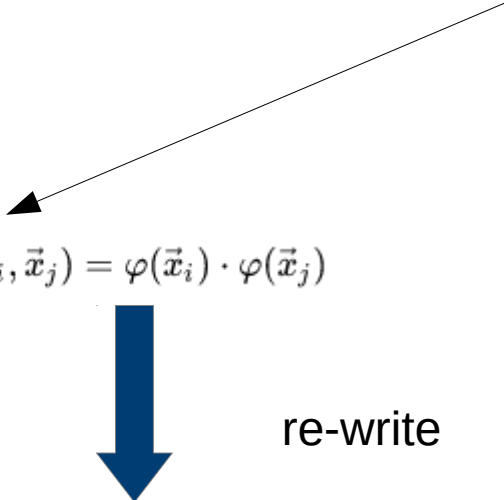

$$k(\vec{x}_i, \vec{x}_j) = \varphi(\vec{x}_i) \cdot \varphi(\vec{x}_j)$$



Always a dot-product in some space

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

“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

$$\begin{aligned} \text{maximize } f(c_1 \dots c_n) &= \sum_{i=1}^n c_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n y_i c_i (\varphi(\vec{x}_i) \cdot \varphi(\vec{x}_j)) y_j c_j \\ &= \sum_{i=1}^n c_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n y_i c_i k(\vec{x}_i, \vec{x}_j) y_j c_j \end{aligned}$$

## Popular Kernels:

Polynomial (of degree d)

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

Gauss (or RBF)

$$k(\vec{x}_i, \vec{x}_j) = \exp(-\gamma \|\vec{x}_i - \vec{x}_j\|^2)$$



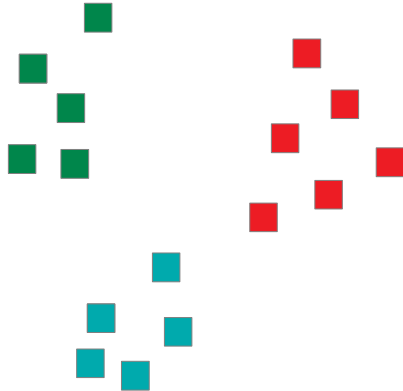
## SVM Inference:

→ evaluate kernel with all Support Vectors and take the sign

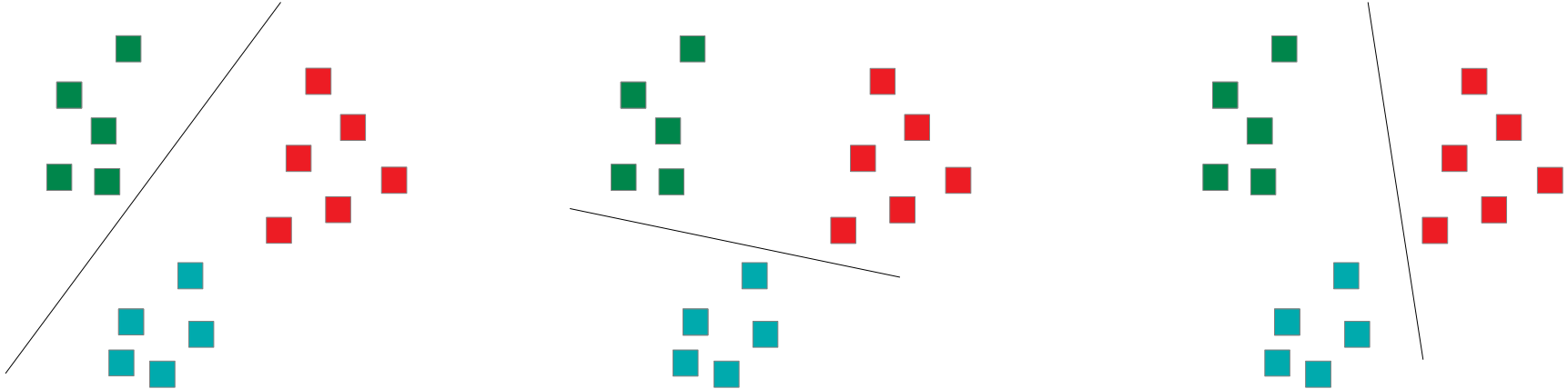
$$\vec{z} \mapsto \text{sgn}(\vec{w} \cdot \varphi(\vec{z}) - b) = \text{sgn}\left(\left[\sum_{i=1}^n c_i y_i k(\vec{x}_i, \vec{z})\right] - b\right).$$

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

Multi class problems:



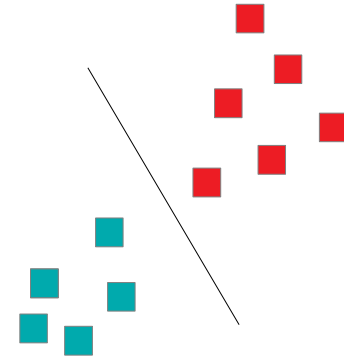
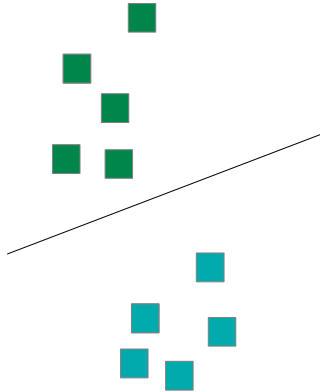
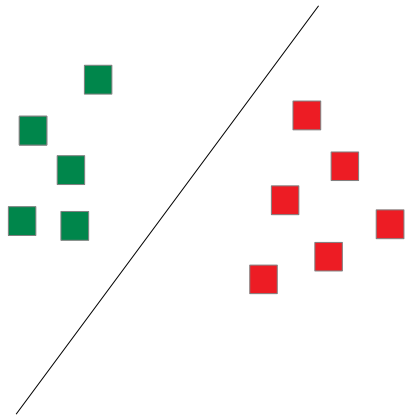
## Multi class problems: 1-vs-Rest



N models, take best.

# Support Vector Machines

Multi class problems: 1-vs-1



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

## 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 ?**

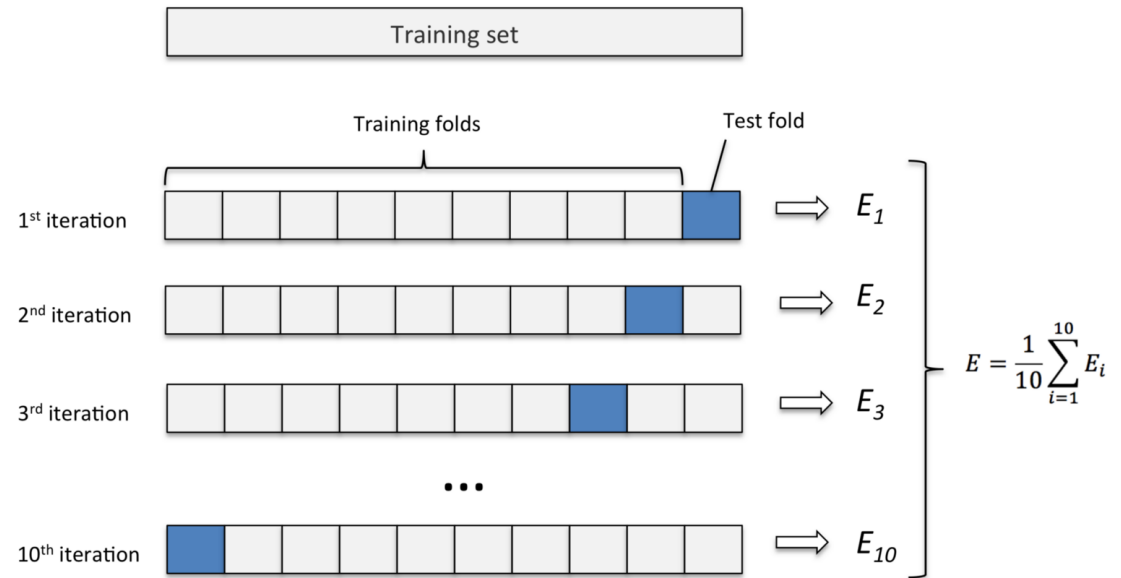
## How to find the best model ?

- 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 (→ can't test too often to stay unbiased)
  - Computational complexity (→ can't try every possible combination)

## 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



## 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



**Lab exercises coming up ...**