# Classification

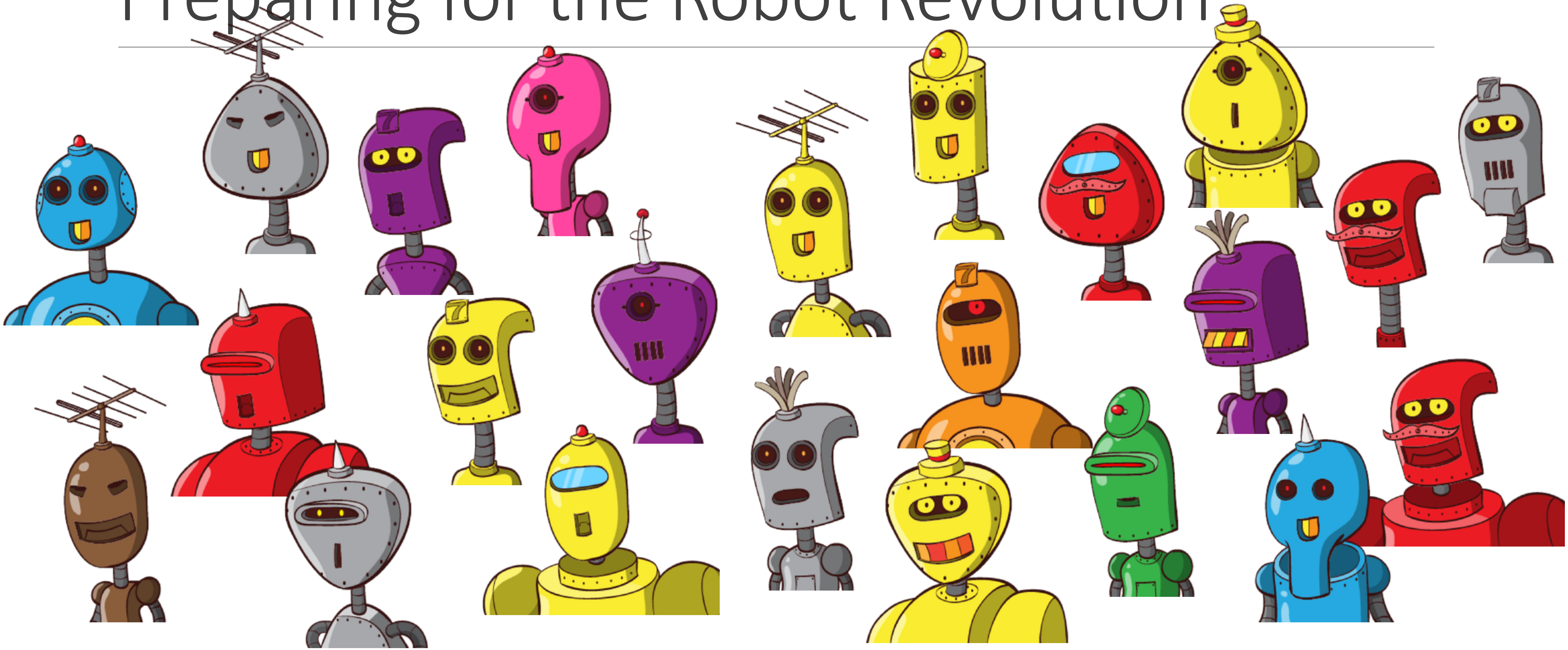
KURT DRIESSENS

DEPARTMENT OF DATA SCIENCE AND KNOWLEDGE ENGINEERING

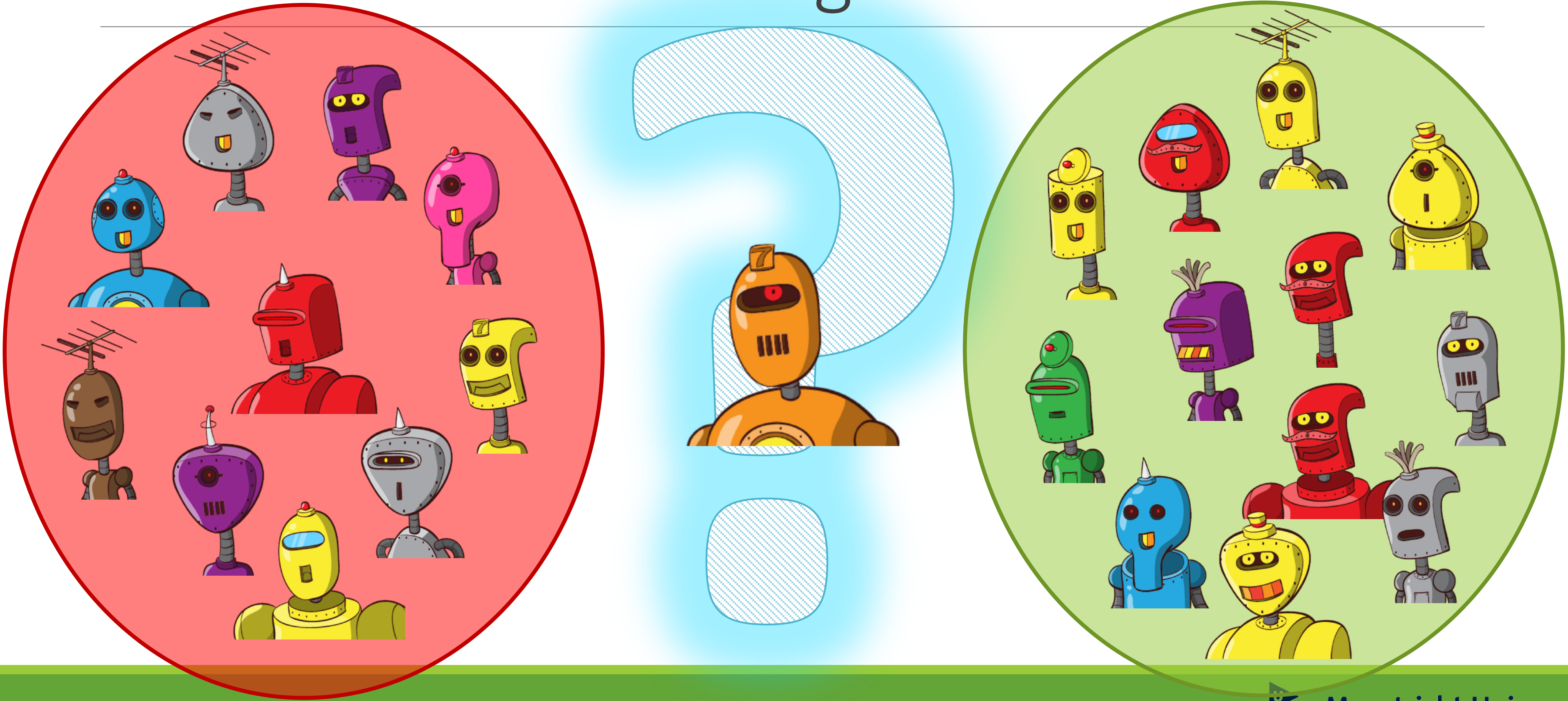
MAASTRICHT UNIVERSITY

Images thanks to https://robohash.org

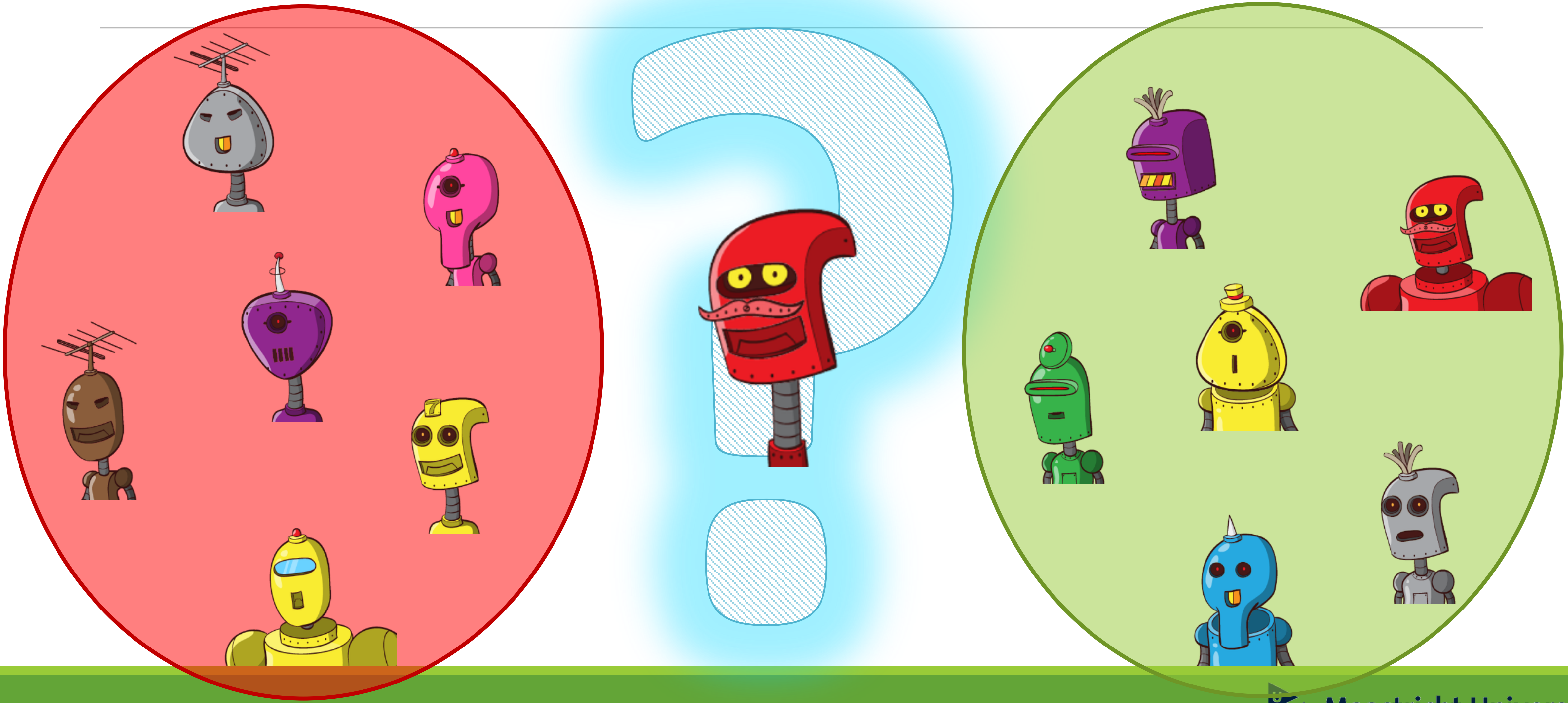
# Preparing for the Robot Revolution



# Classification = Making Predictions



# Your turn!!



# Nearest Neighbor Classification

“Birds of a feather, flock together.”

Key ingredient: similarity measure … or dissimilarity measure: distance!

Algorithm:

1. Store all examples
2. Classify a new example by copying the class of it’s nearest “neighbor”

# Nearest Neighbor Properties

+ Learning is fast

+ No data is lost

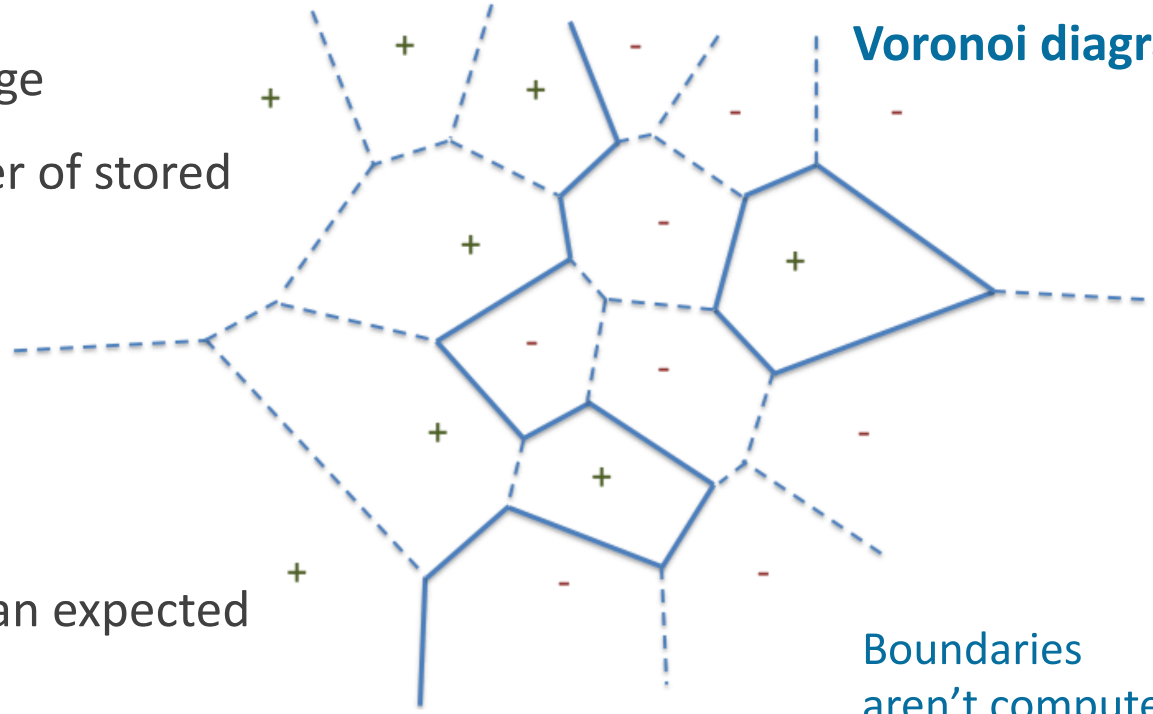
+ Distances are tunable through expert knowledge

+ Complexity of the hypothesis rises with number of stored examples

- Some of the data might be noise

- Computing all distances might be slow

- Distance might be more difficult to get right than expected



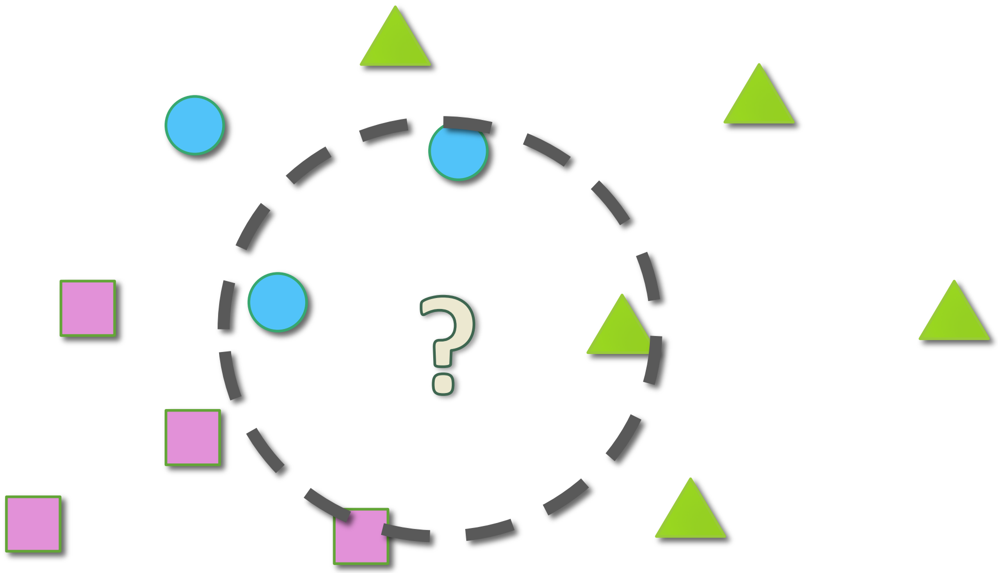
**Voronoi diagram**

Boundaries aren't computed.

# kNN: k-nearest neighbor

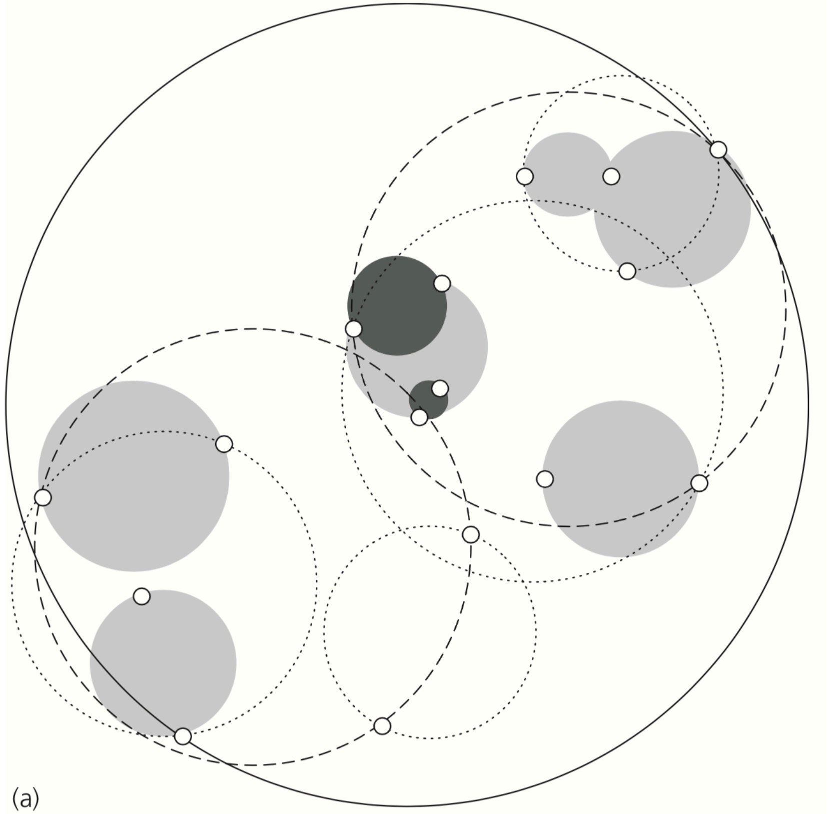
Make the algorithm more robust by using multiple neighbors

E.g.: use voting

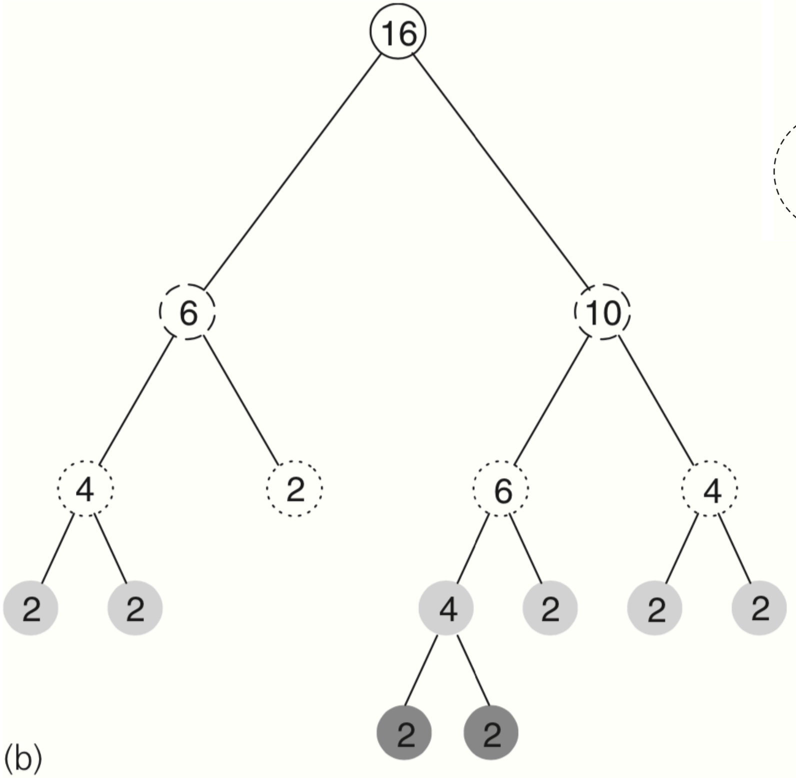


# Don’t just store, but store smartly

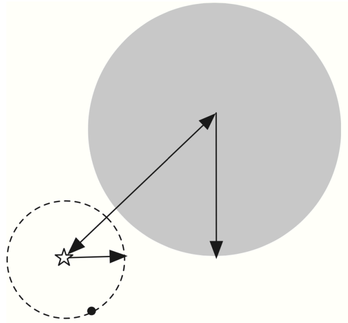
E.g. KD-trees, **Ball-Trees**



(a)



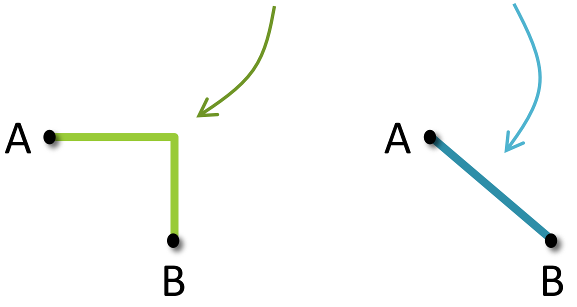

(b)



# Similarity measures

Distance metrics: measure of dis-similarity

E.g. Manhattan, Euclidean or $L_n$-norm for numerical attributes




$$L^n(\mathbf{x_1}, \mathbf{x_2}) = \sqrt[n]{\sum_{i=1}^{\#dim} |x_{1,i} - x_{2,i}|^n}$$

Hamming distance for nominal attributes

$$d(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{i=1}^{n} \delta(x_i, y_i)}$$

*where*

$$\delta(x_i, y_i) = 0 \text{ if } x_i = y_i$$

$$\delta(x_i, y_i) = 1 \text{ if } x_i \neq y_i$$

# Distance definition = critical!

E.g. comparing humans

1. 1.85m, 37yrs
2. 1.83m, 35yrs
3. 1.65m, 37yrs

d(1,2) = 2.00…0999975…

d(1,3) = 0.2

d(2,3) = 2.00808…

1. 185cm, 37yrs
2. 183cm, 35yrs
3. 165cm, 37yrs

d(1,2) = 2.8284…

d(1,3) = 20.0997…

d(2,3) = 18.1107…

# Normalize feature values

Rescale all dimensions such that the range is equal, e.g. [-1,1] or [0,1]

For [0,1] range:

with $m_i$ the minimum and $M_i$ the maximum value for attribute i

$$x_i' = \frac{x_i - m_i}{M_i - m_i}$$

# Curse of dimensionality

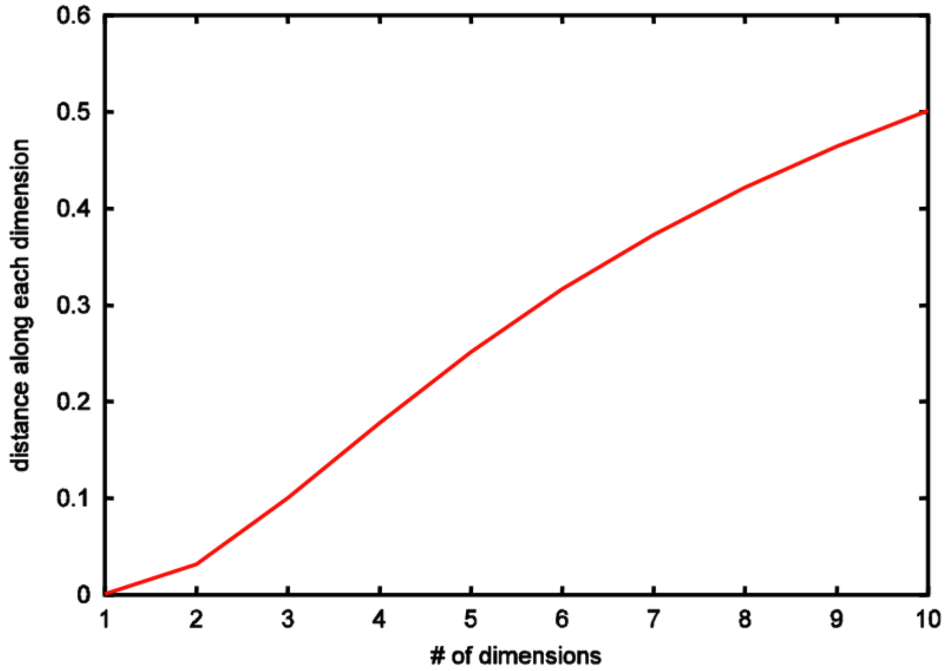
Assume a uniformly distributed set of 5000 examples

To capture 5 nearest neighbors we need:

- in 1 dim: 0.1% of the range
- in 2 dim: $\sqrt{0.1\%}$ = 3.1% of the range
- in n dim: $0.1\%^{1/n}$

# Curse of Dimensionality (2)

With 5000 points in 10 dimensions, each attribute range must be covered approx. ? % to find 5 neighbors …



# More distances

Cosine distance

- Angle between points as seen from the origin: Think ”Looking for nearby stars.”
- Less subjected to the curse of dimensionality

For Strings

- Levenshtein distance/edit distance

= minimal number of changes to change one word to the other

Allowed edits/changes:

1. delete character
2. insert character
3. change character (not used by some other edit-distances, then counts for 2 edits)

# Even more distances

Given two time series:

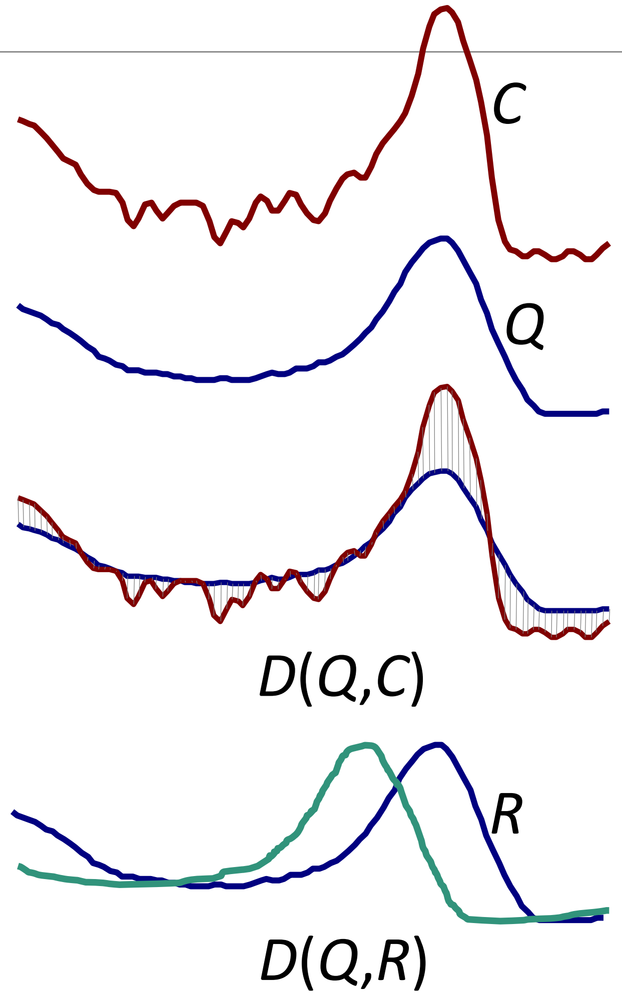
$Q = q_1 \ldots q_n$

$C = c_1 \ldots c_n$

Euclidean

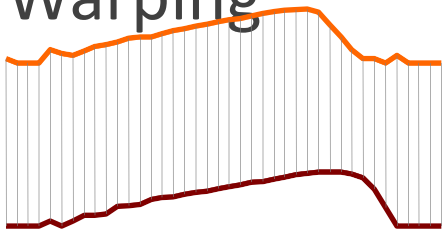
$$D(Q,C) \equiv \sqrt{\sum_{i=1}^{n}(q_i - c_i)^2}$$

Start and end times are critical!



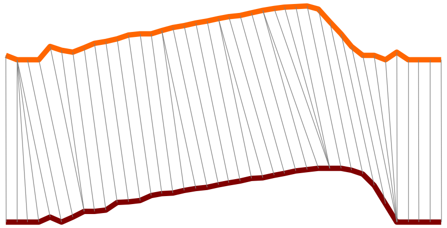
# Sequence distances (2)

## Dynamic Time Warping



**Fixed Time Axis**

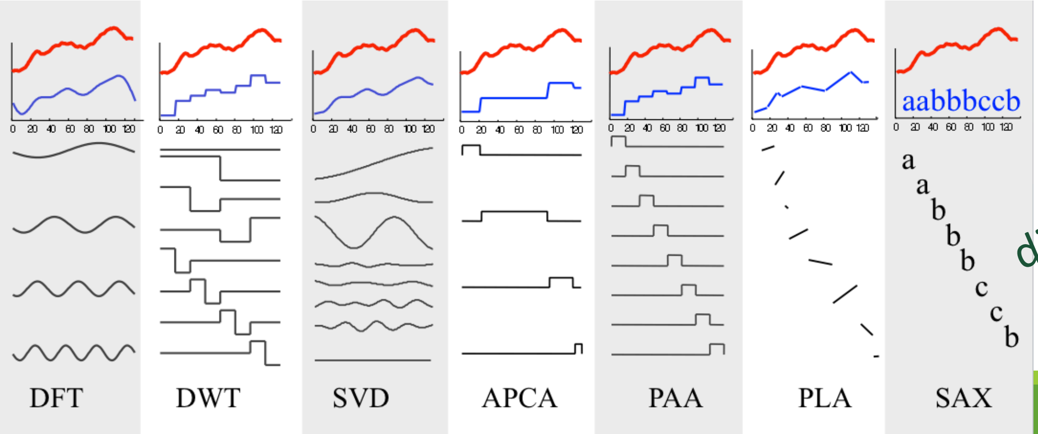
*Sequences are aligned "one to one".*



**"Warped" Time Axis**

*Nonlinear alignments are possible.*

## Dimensionality reduction

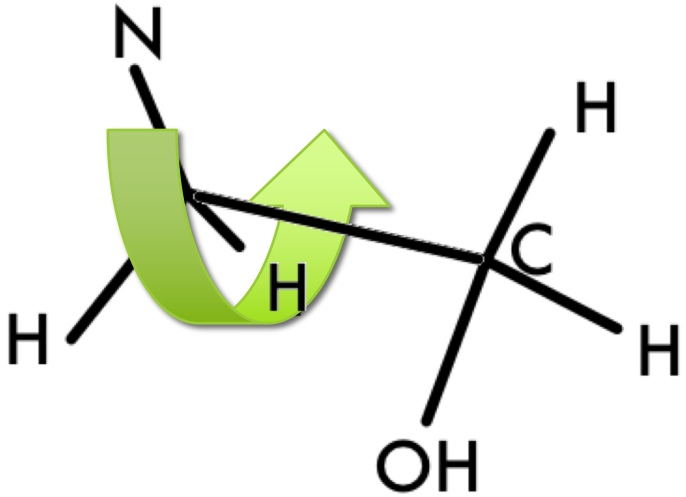


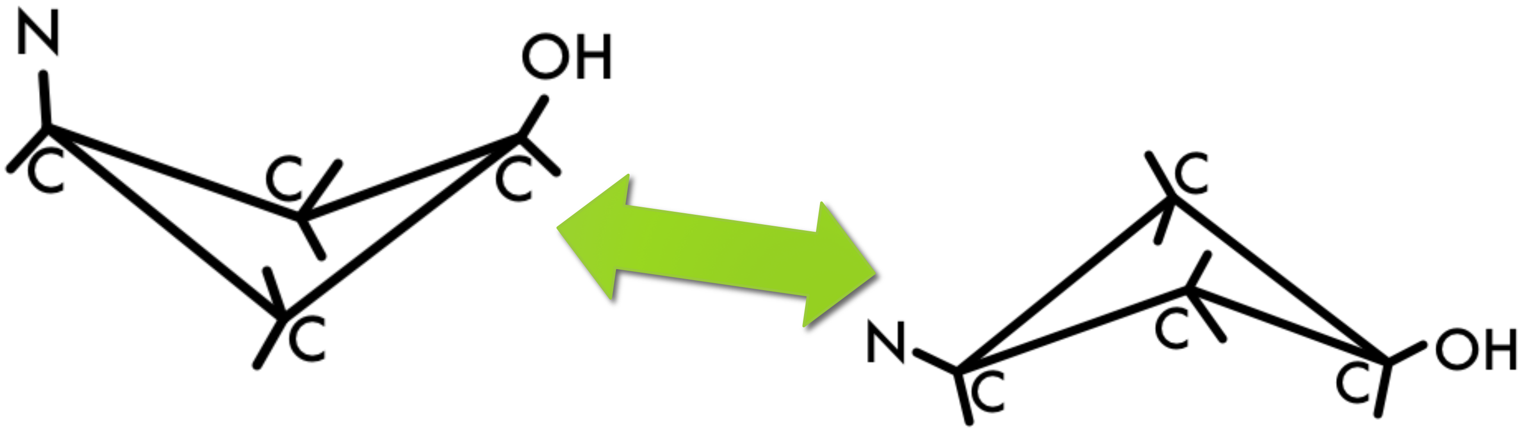

edit distance!

# Even more more distances!

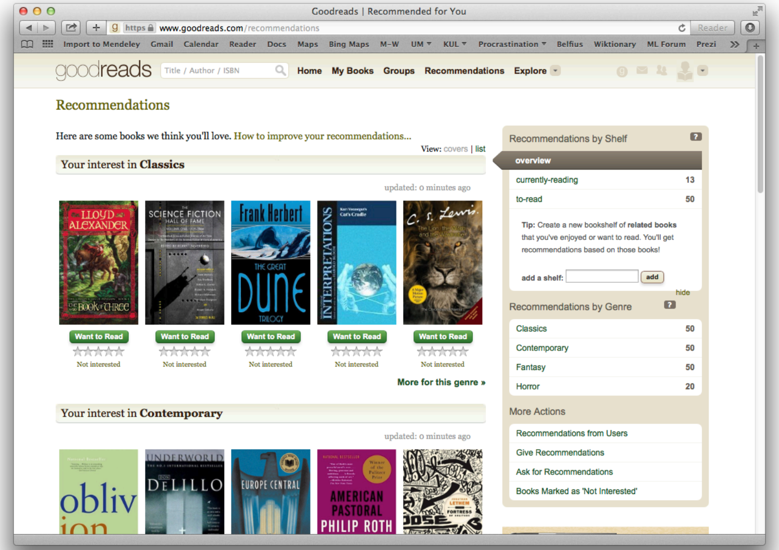
Distances exist for:

- Sets
- Graphs
- Trees
- …



# In the real world: Recommender Systems




| Movie | Alice (1) | Bob (2) | Carol (3) | Dave (4) |
|---|---|---|---|---|
| Love at last | 5 | 5 | 0 | 0 |
| Romance forever | 5 | ? | ? | 0 |
| Cute puppies of love | ? | 4 | 0 | ? |
| Nonstop car chases | 0 | 0 | 5 | 4 |
| Swords vs. karate | 0 | 0 | 5 | ? |

rating by user j of entry k

$$\hat{R}_{ik} = \bar{R}_i + \alpha \sum_{X_j \in N_i} W_{ij}(R_{jk} - \bar{R}_j)$$

Avg. rating of user i

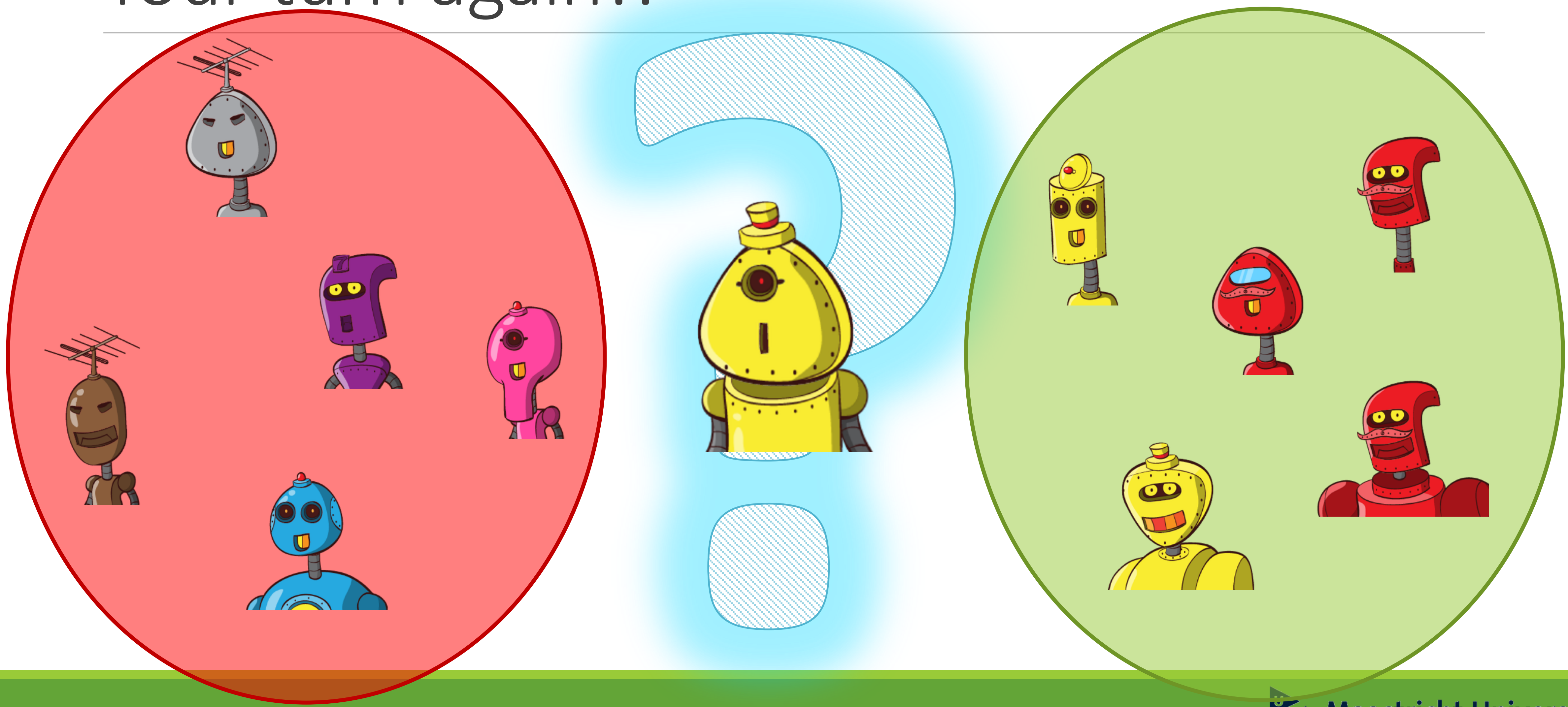
can be all entries or kNN

$$W_{ij} = \frac{\sum_k (R_{ik} - \bar{R}_i)(R_{jk} - \bar{R}_j)}{\sqrt{\sum_k (R_{ik} - \bar{R}_i)^2}\sqrt{\sum_k (R_{jk} - \bar{R}_j)^2}}$$

Pearson coefficient

$$\alpha = \left(\sum |W_{ij}|\right)^{-1}$$

# Your turn again!!

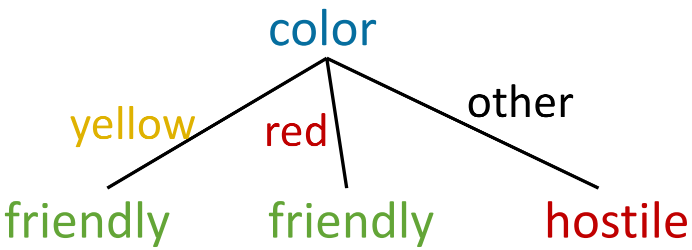


# Decision trees (and rules)

Idea: use properties to select which example a prediction holds for.

E.g.

1. if (color = yellow) then friendly — Decision Rule

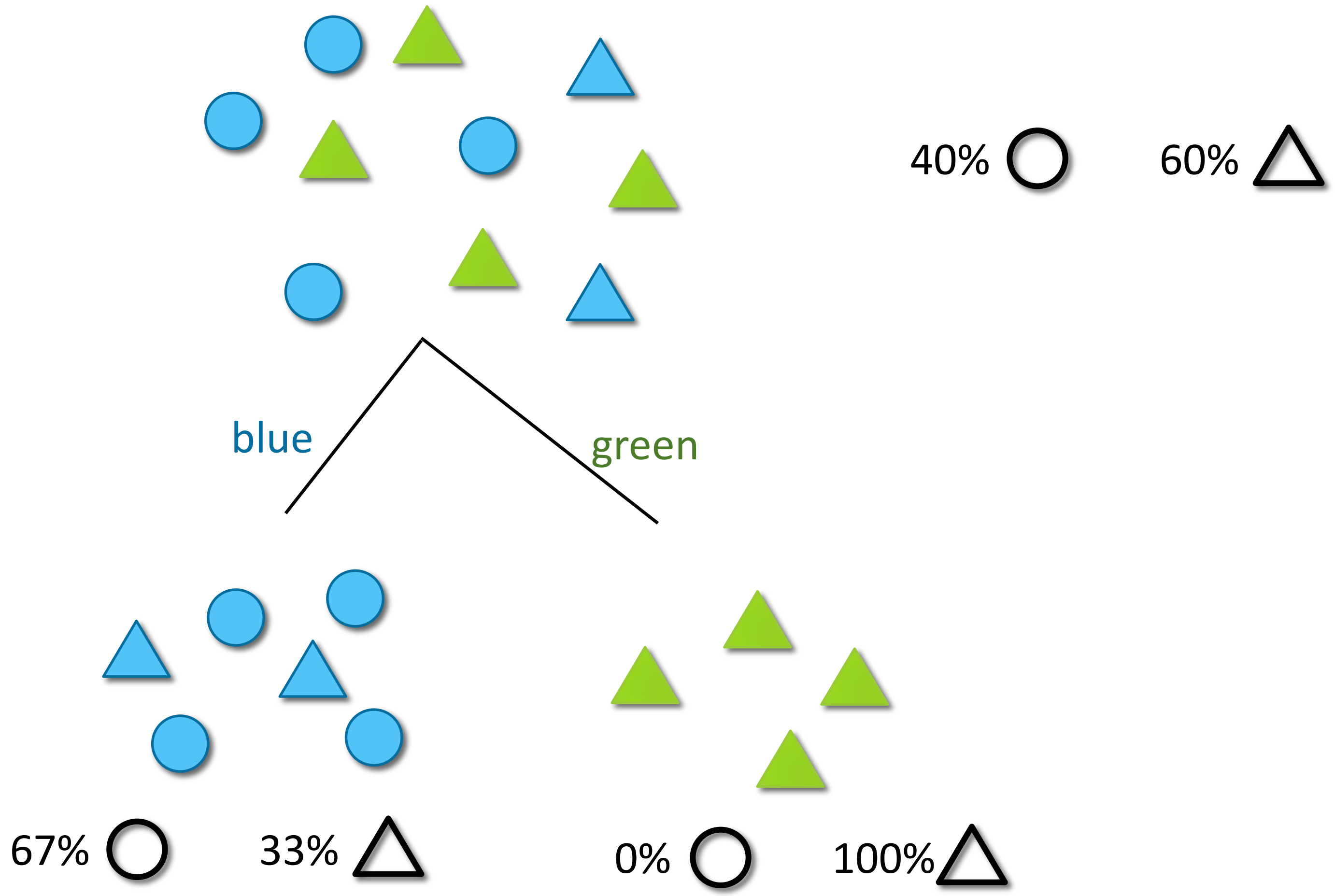
2. 


Decision Tree

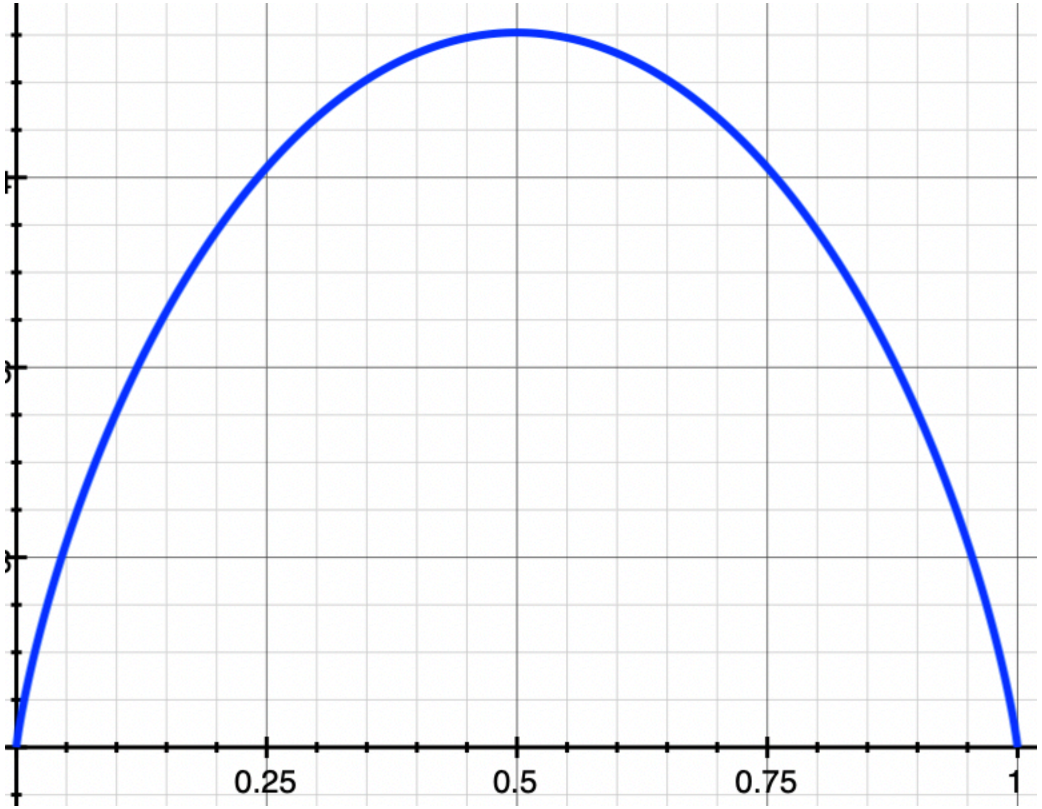
# Decision tree algorithms

Before:

After:



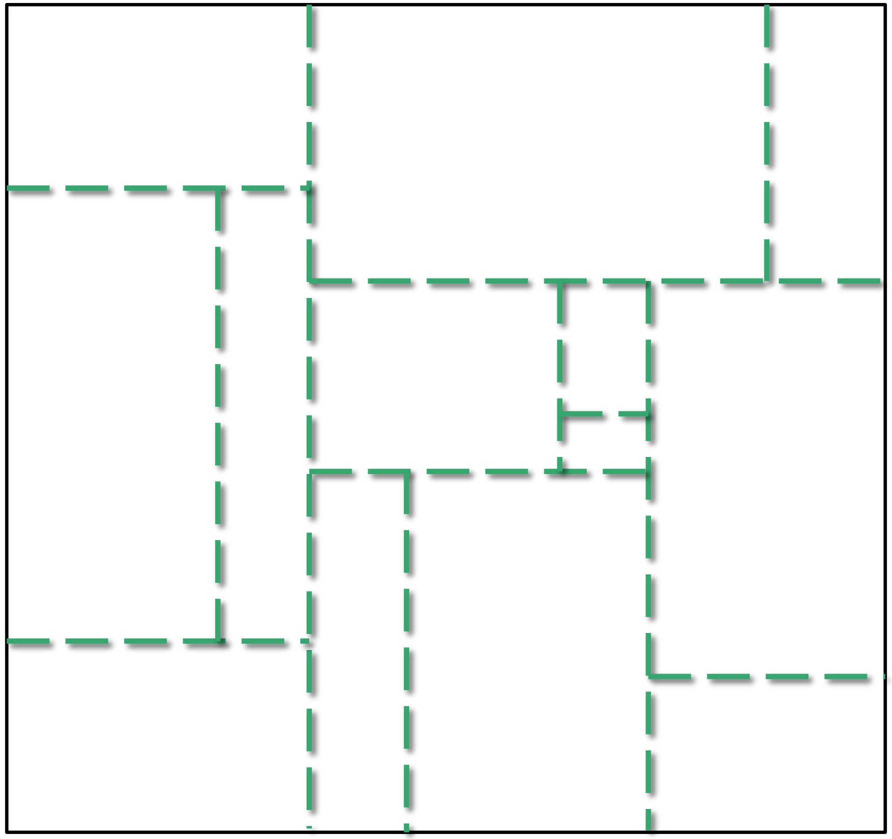
# Information Entropy




$$Entropy = -\sum_i p_i log_2(p_i)$$

Other possibilities, e.g. Gini index

# The completeness of trees



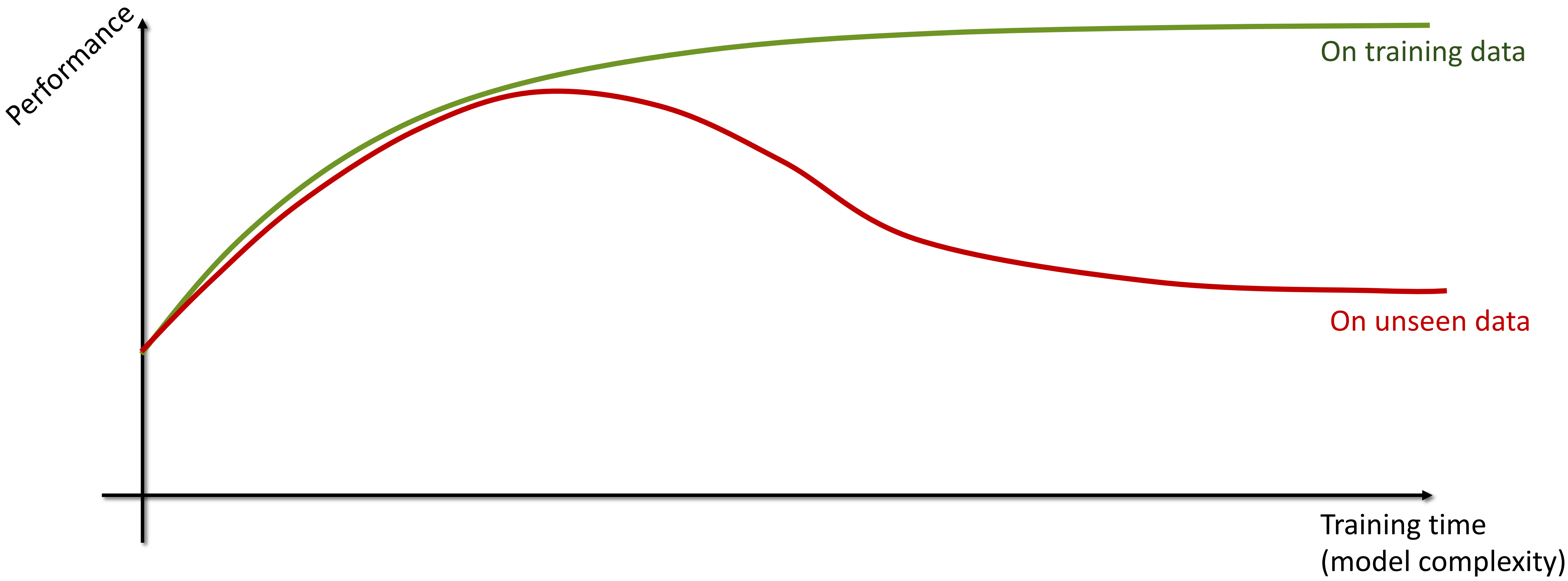
Trees can *represent* any concept

- Every example can be its own leaf
- No generalization

Possibility of "overfitting"

= Adapting the model too much to the training data, so that it does not generalize to unseen data

# Over-fitting



# Pruning

In large trees, some branches can overfit the training data

Pre-pruning

Set a minimum for the number of examples needed to split a leaf node to stop learning early

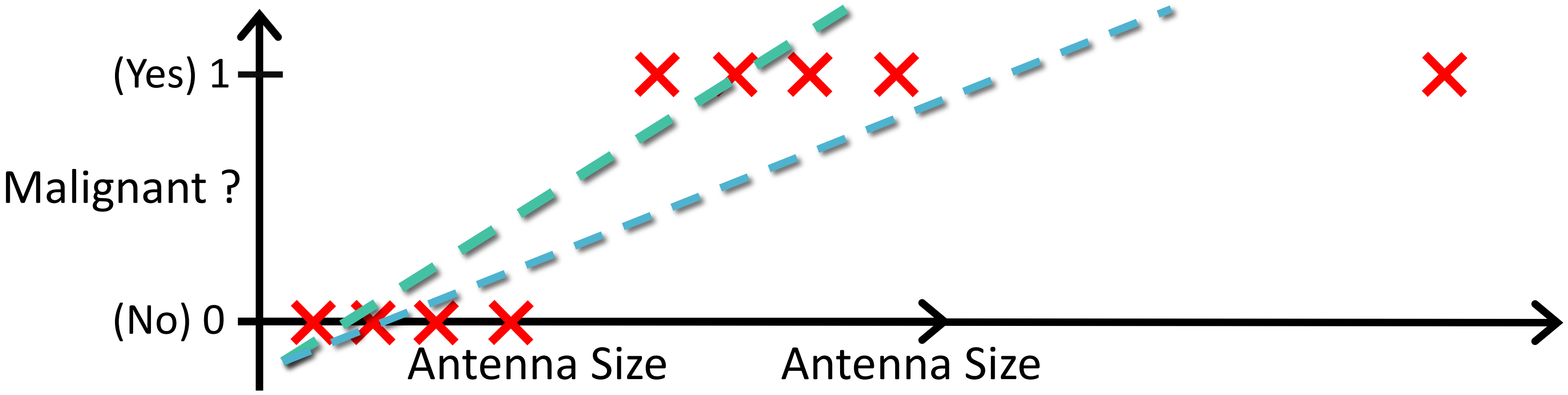
Post-pruning

After learning, use a “validation” set to **prune away** those parts of the tree that are too detailed

# Linear regression?

Cast binary classification problem as a regression problem?

- Negative examples get y = 0
- Positive examples get y = 1
- Predict positive when $h_\theta(x) > 0.5$



# LOGISTIC regression!

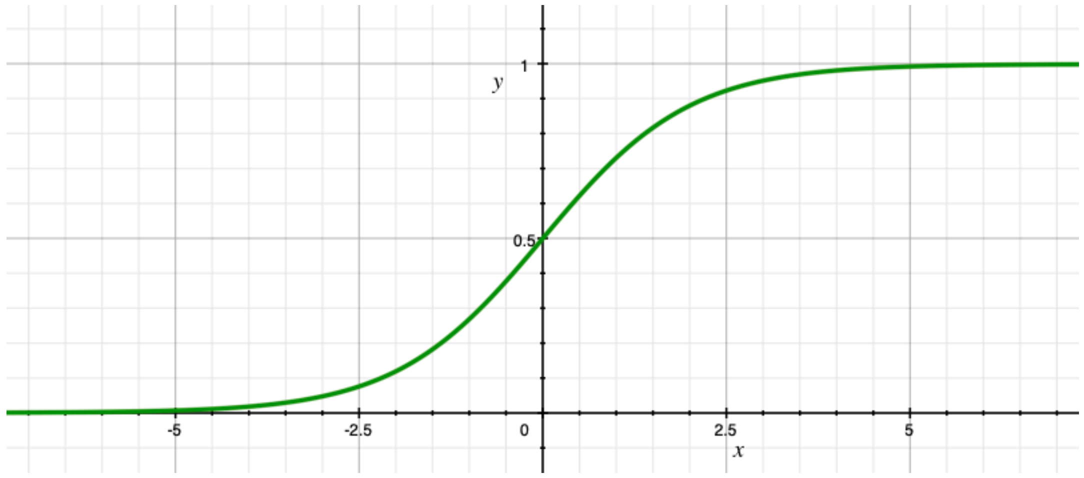
$h_\theta(x)$ should only predict values from [0;1]

Probability that y = 1

$$h_\theta(\mathbf{x}) = g(\theta^T\mathbf{x}) = \frac{1}{1+e^{-\theta^T\mathbf{x}}} = p(y=1|\mathbf{x};\theta)$$

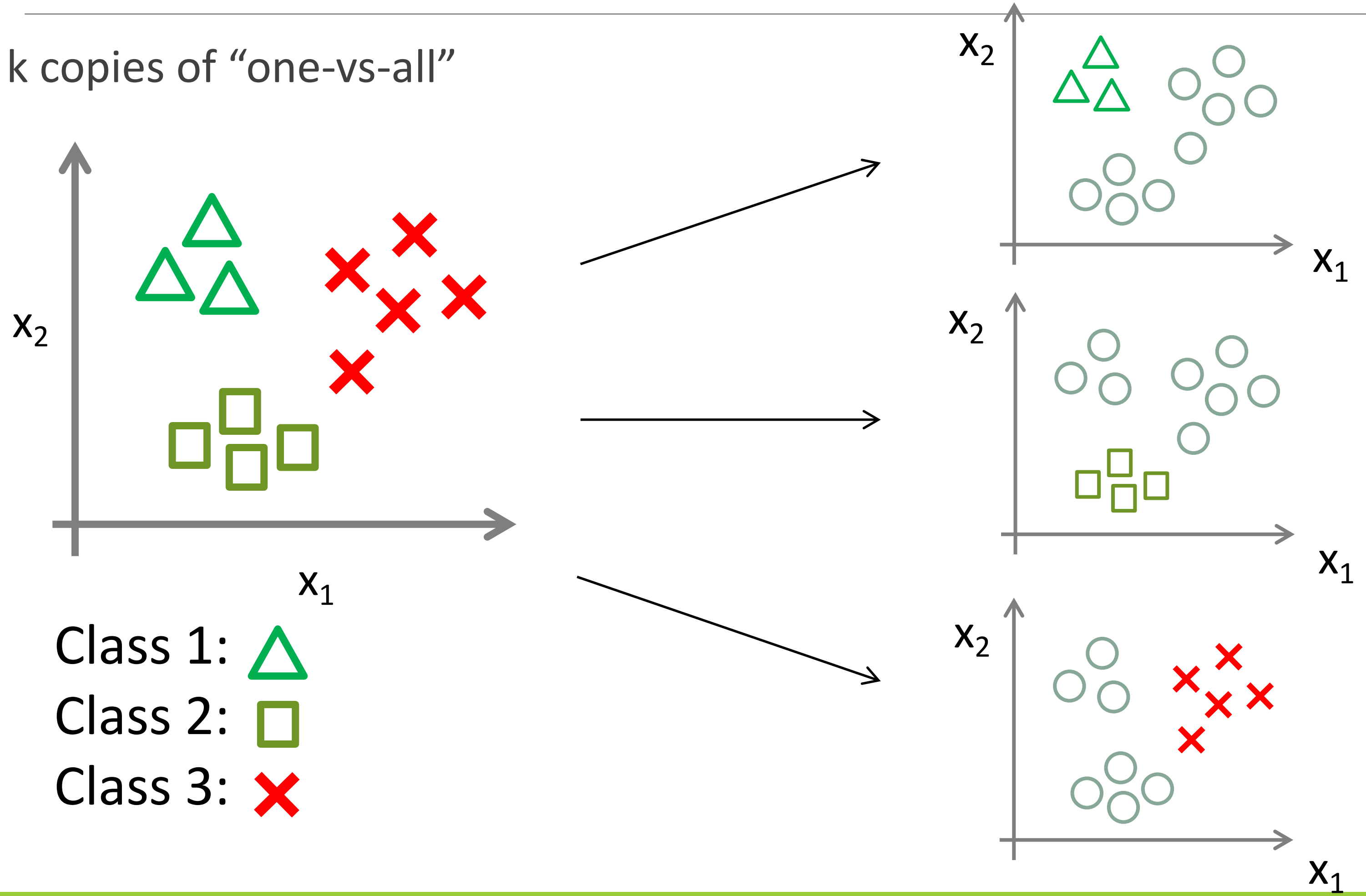
Sigmoid: Logistic function

$$g(z) = \frac{1}{1+e^{-z}}$$



# How about multi-class problems?

k copies of "one-vs-all"




Class 1: △

Class 2: □

Class 3: ✕

$$h_\theta^{(i)}(x) = P(y = i | x; \theta)$$

predict

$$\max_i h_\theta^{(i)}(x)$$

# In one go: Softmax Regression

Turn $y^{(i)}$ into

$$\mathbf{y}^{(i)} = \begin{bmatrix} p(y^{(i)} = 1|x^{(i)}, \theta) \\ p(y^{(i)} = 2|x^{(i)}, \theta) \\ \vdots \\ p(y^{(i)} = k|x^{(i)}, \theta) \end{bmatrix}$$

$$\theta = \begin{bmatrix} -\theta_1^T- \\ -\theta_2^T- \\ \vdots \\ -\theta_k^T- \end{bmatrix}$$

and $h(x^{(i)})$ into

$$\mathbf{h}_\theta(x^{(i)}) = \frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}} \begin{bmatrix} e^{\theta_1^T x^{(i)}} \\ e^{\theta_2^T x^{(i)}} \\ \vdots \\ e^{\theta_k^T x^{(i)}} \end{bmatrix}$$

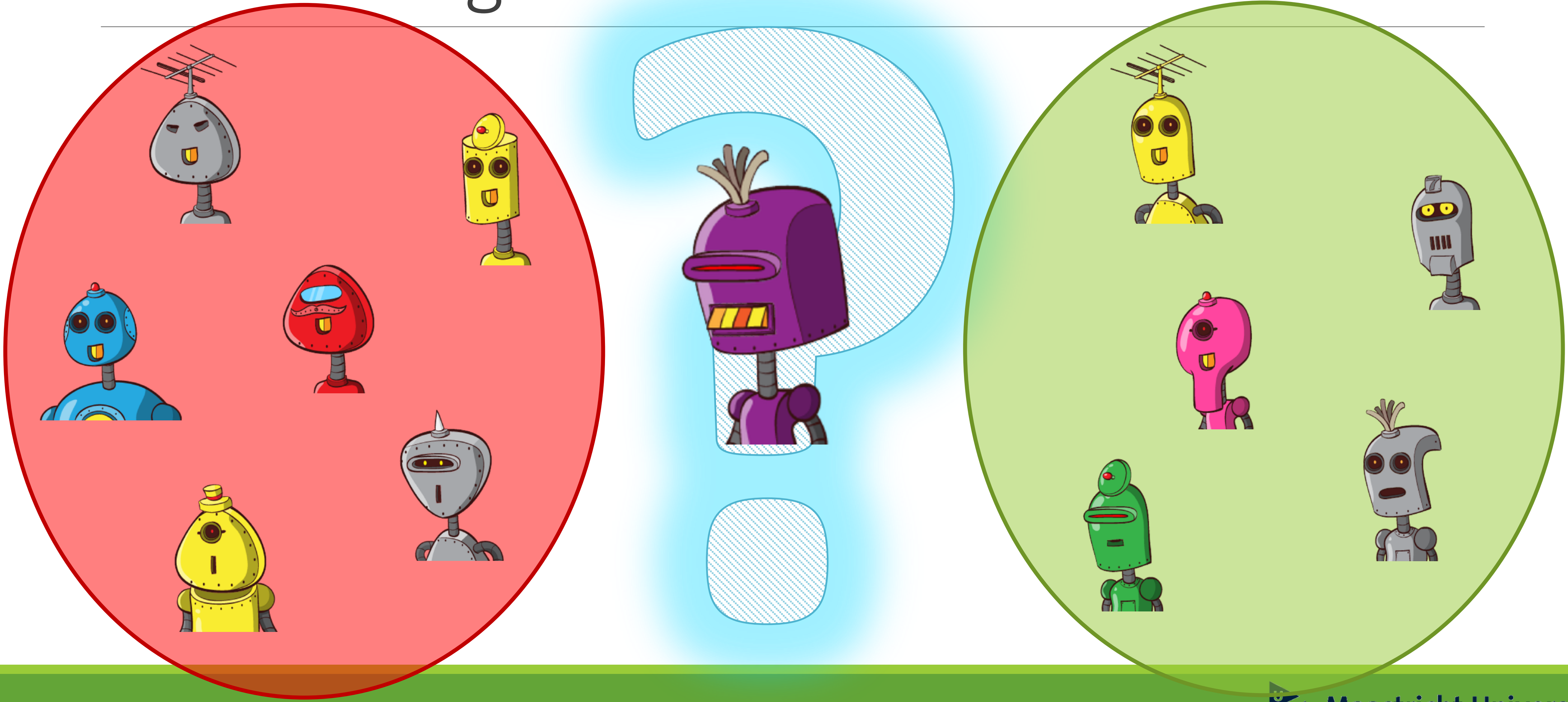
# Overfitting again…

Also here overfitting can happen

$$J(\theta) = \left[ -\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

Punish the use of large weights …
… because they represent steep model changes
… and thus complex decision boundaries!

# Your turn again!!



# Support vector machines

Popular, “go-to” ML approach

- many successes, e.g., using Radial Basis Function kernel

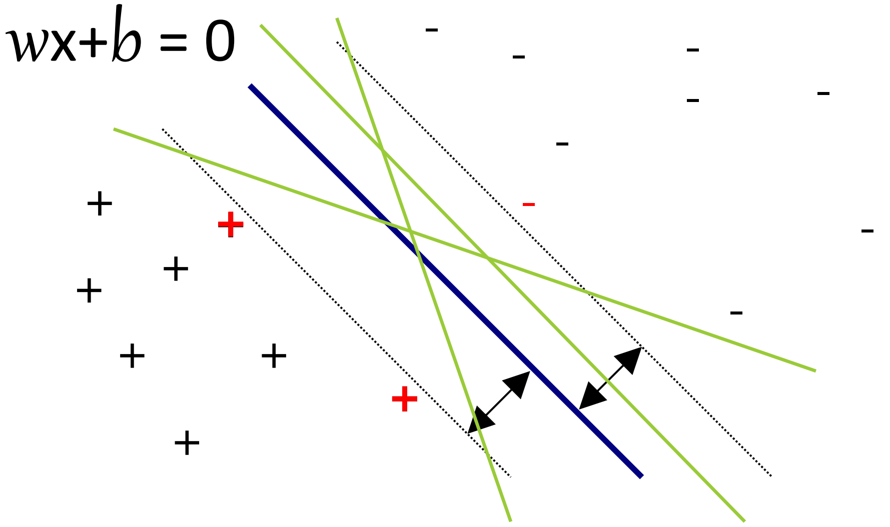
Usable in similar situations as neural networks


Important concepts:

- Finding a "maximal margin" separation
- Transformation into high dimensional space

# Linear SVMs

Idea:

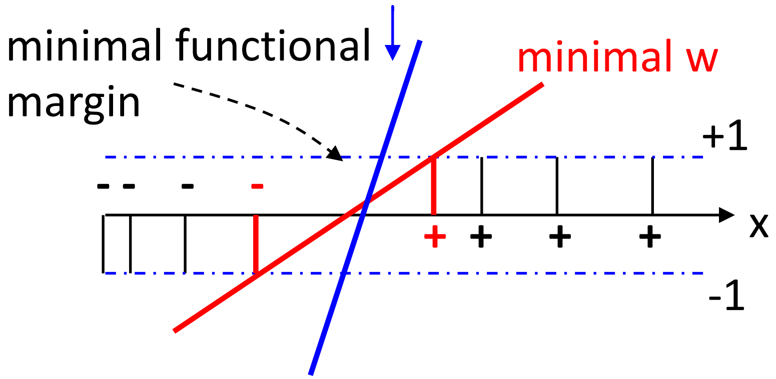



Find hyperplane that discriminates + from -

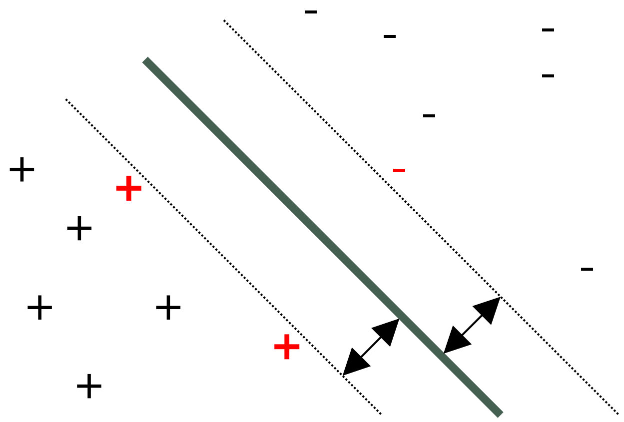
"margin" should be maximal

- margin = distance of hyperplane to closest points
- solution is unique, and determined by just a few points ("support vectors")

# Same goal, easier to compute:

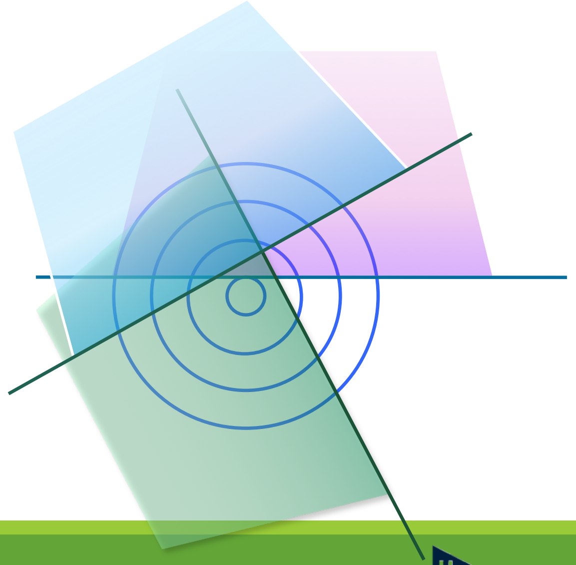



$$\min_{w,b} ||w||$$
$$\text{s.t.} \quad y^{(i)}(w^T x^{(i)} + b) \geq 1$$



## Convex optimization problem

- works as well as logistic regression
- only satisfiable for linearly separable data



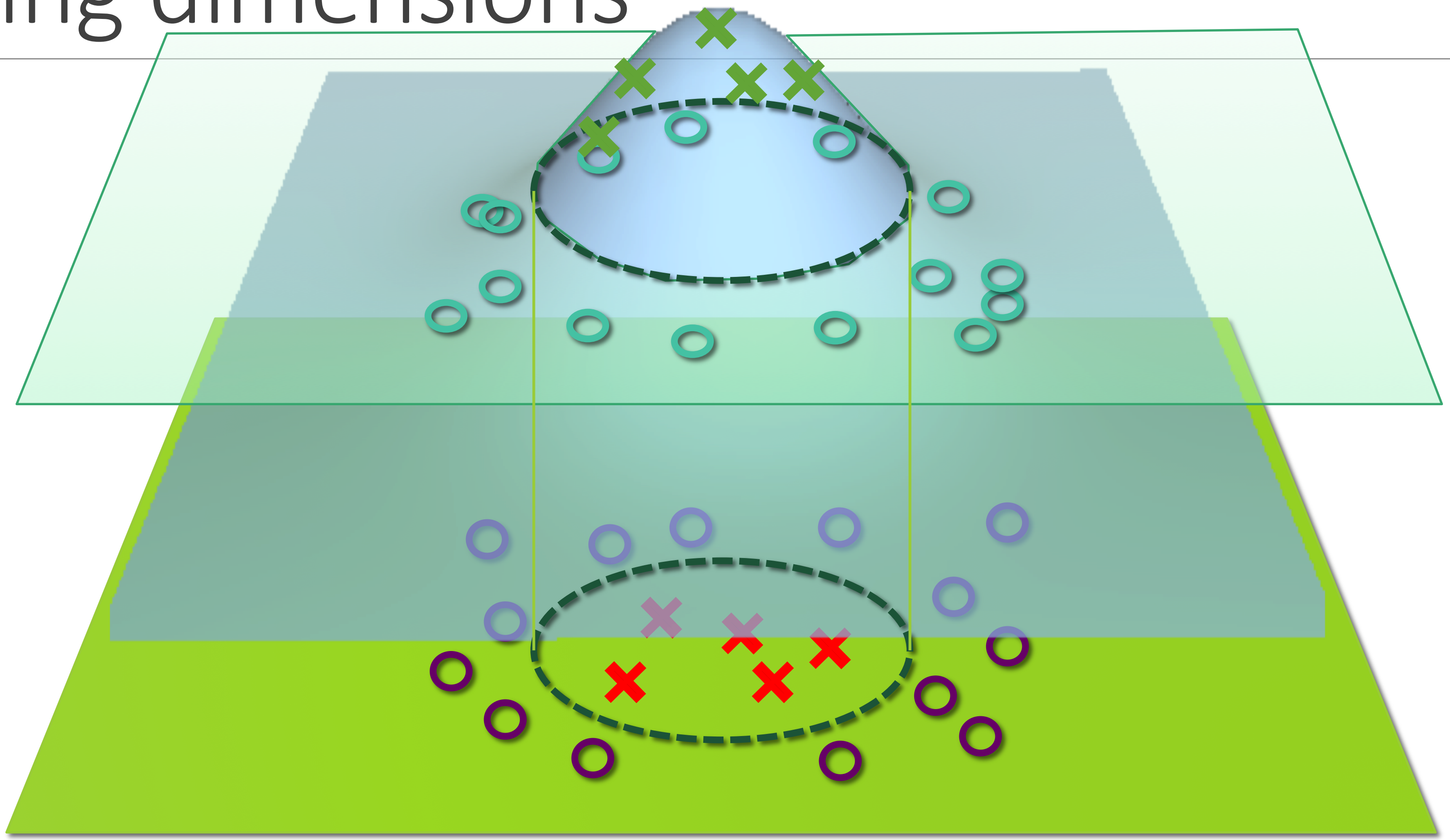
# SVM Model

$$h(x) = \sum_i \mu_i y^{(s_i)} (x^{(s_i)})^T x + \frac{1}{|S|} \sum_i (y^{(s_i)} - \sum_j \mu_j y^{(s_j)} (x^{(s_j)})^T x^{(s_i)})$$

… only depends on the independent part of the learning data **x** as part of a so called **inner-product**!!

Even the function to optimize: $$-\frac{1}{2} \sum_i \sum_j \mu_i \mu_j y^{(i)} y^{(j)} (x^{(i)})^T x^{(j)} + \sum_k \mu_k$$

# Adding dimensions



# Example transformations

E.g.: learning quadratic decision surfaces in 2D:

- map $x_1,x_2$ to space with dimensions

$$x_1,\ x_2,\ x_1x_2,\ x_1^2,\ x_2^2$$

- learn "hyperplane" $ax_1+bx_2+cx_1x_2+dx_1^2+ex_2^2+f=0$
- in original space this is a quadratic form

You can do this and use logistic regression!!

- This is almost standard practice when using logistic regression ..

# Kernel trick

Transformation can be done implicitly ...

$$\min_M L(M) = -\frac{1}{2}\sum_i\sum_j \mu_i\mu_j y^{(i)}y^{(j)}\mathbf{x}^{(i)}\mathbf{x}^{(j)} + \sum_i \mu_i$$

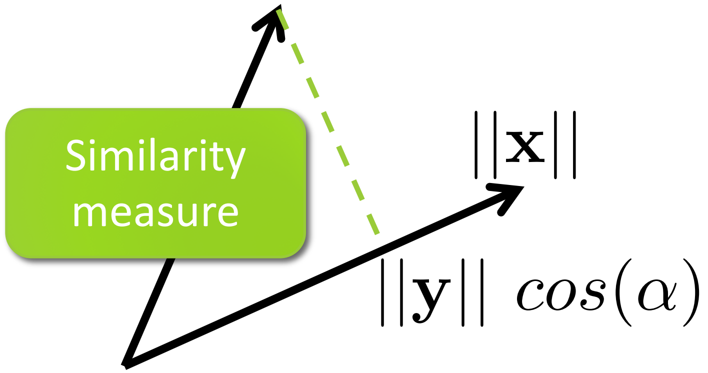
$$h(x) = \sum_i \mu_i y^{(s_i)}\mathbf{x}^{(s_i)}\mathbf{x} + \frac{1}{|S|}\sum_i\Big(y^{(s_i)} - \sum_j \mu_j y^{(s_j)}\mathbf{x}^{(s_j)}\mathbf{x}^{(s_i)}\Big)$$

So:

- Call the transformation F
- We then need to train on F(**x**) instead of **x**
- Define K(x,y) = F(x) F(y)
  - K is called a kernel function
  - choice of K = implicit choice of F

$$h(x) = \sum_i \mu_i y^{(s_i)} K(\mathbf{x}^{(s_i)}, \mathbf{x}) + b$$

# Inner products and non-linearity

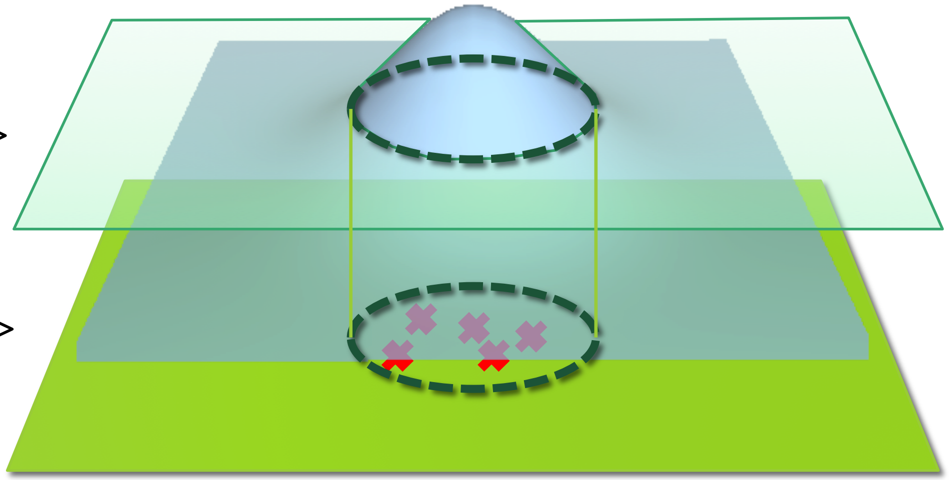



$$< \mathbf{x}, \mathbf{y} >= \mathbf{xy} = \sum_i x_i y_i$$

$$= ||\mathbf{x}|| \; ||\mathbf{y}|| \; cos(\alpha)$$

$$\Phi(\mathbf{x}) \rightarrow < \Phi(\mathbf{x}), \Phi(\mathbf{y}) >$$

$$\mathbf{x} \rightarrow < \mathbf{x}, \mathbf{y} >$$



# Kernel trick

No need to actually compute $\Phi$

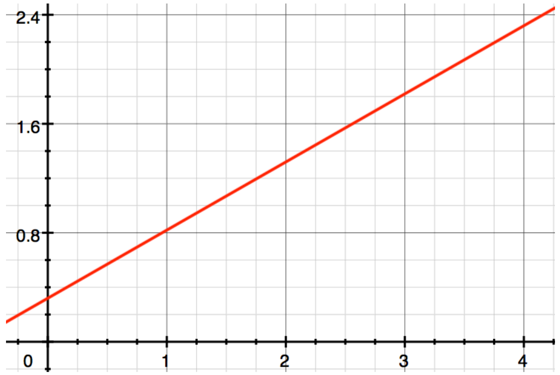
$$< \Phi(\mathbf{x}), \Phi(\mathbf{y}) > \equiv K(\mathbf{x}, \mathbf{y})$$

- allows the transformed space to be of much higher dimension than original without computational cost
- for $\Phi$ to exist, K needs to be symmetrical and positive definite.

# Typical Kernels

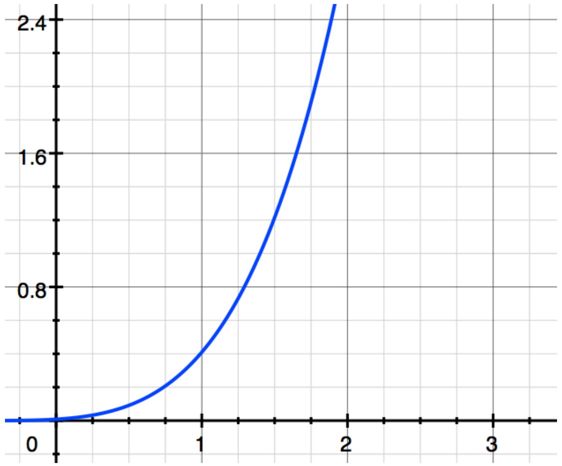
Linear Kernel

$$k(x, y) = x^T y + c$$




Polynomial Kernel

$$k(x, y) = (\alpha x^T y + c)^d$$

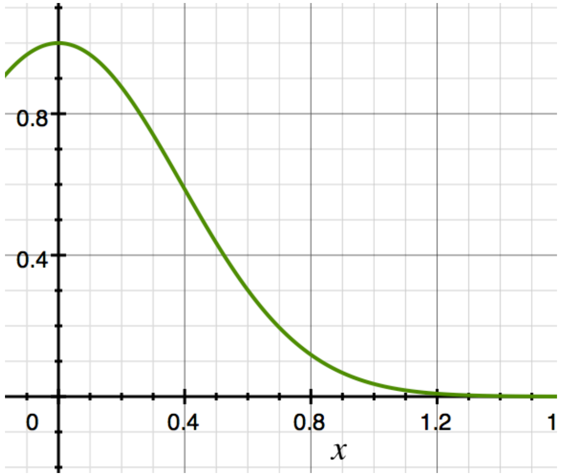



Non-stationary kernels

# Typical Kernels (2)

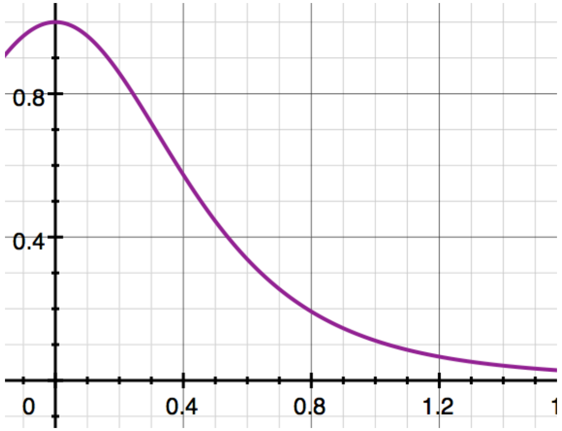
RBF Kernel (Radial Basis Function, a.k.a. Gaussian or Squared Exponential)

$$k(x,y) = \sigma^2 exp\left(-\frac{(x-y)^2}{2l^2}\right)$$



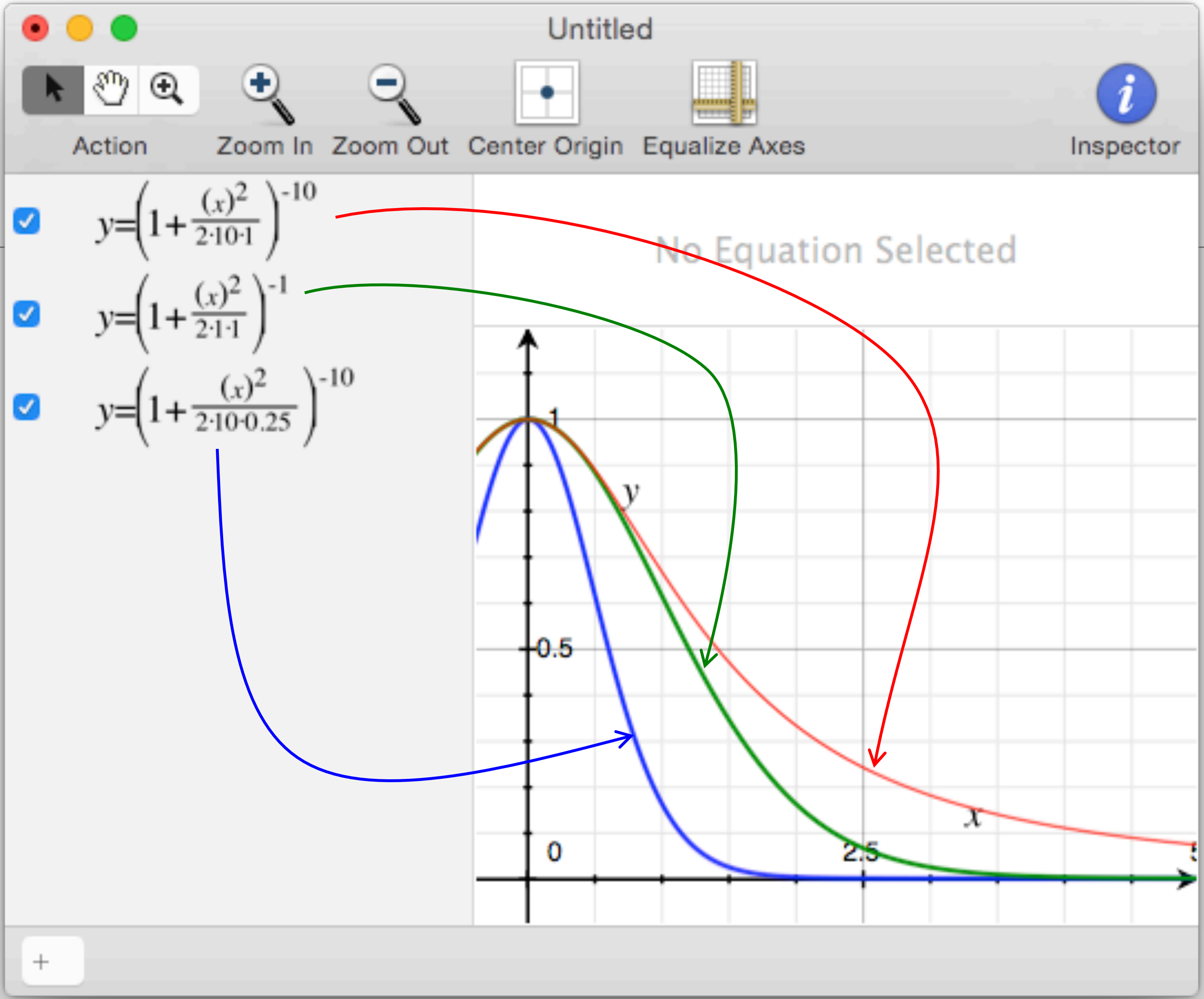

Rational Quadratic Kernel

$$k(x,y) = \sigma^2\left(1+\frac{(x-y)^2}{2\alpha l^2}\right)^{-\alpha}$$




Stationary kernels

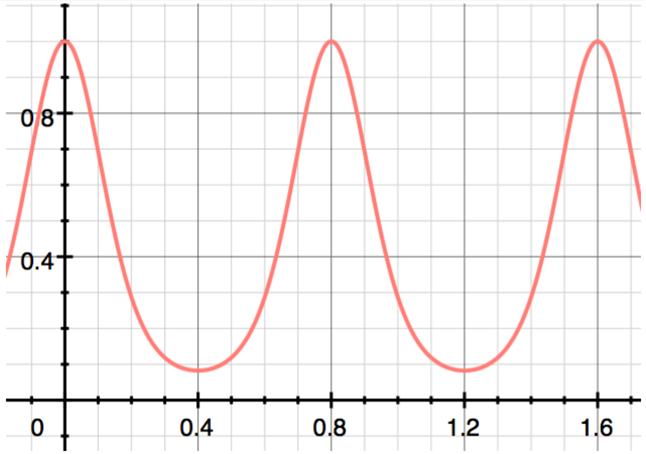
$$\sigma^2 \left(1 + \frac{(x-y)^2}{2\alpha l^2}\right)^{-\alpha}$$



# Typical Kernels (3)

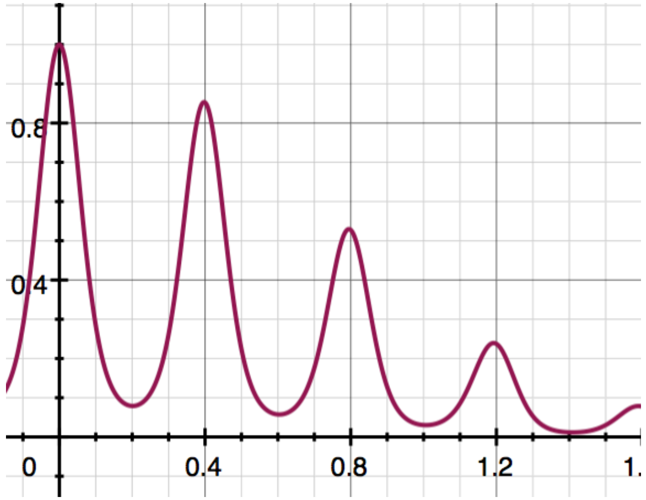
Periodic Kernel

$$k(x,y) = \sigma^2 exp\left(-\frac{2sin^2(\frac{\pi}{p}|x-y|)}{l^2}\right)$$




Local Periodic Kernel

$$k(x,y) = \sigma^2 exp\left(-\frac{2sin^2(\frac{\pi}{p}|x-y|)}{l^2}\right) exp\left(-\frac{(x-y)^2}{2l^2}\right)$$




Stationary kernels

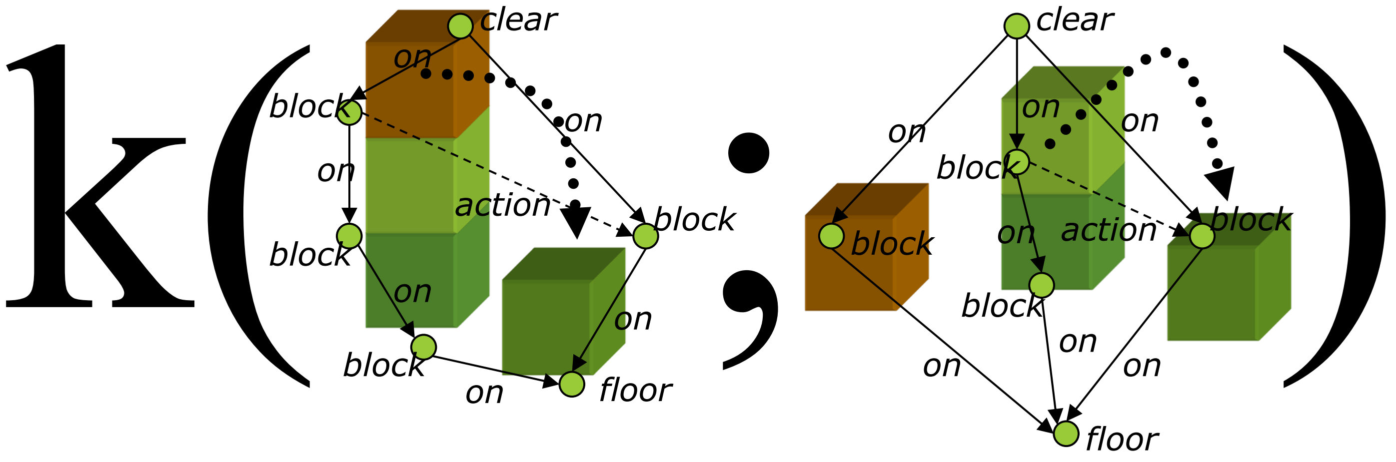
# More complex kernels

Convolutional Kernels:

- Based on a decomposition of the “data-object”
- And kernels defined on the decomposed parts

*Examples:*

- Sets
- Graphs
- Strings
- …



Google: “The Kernel Cookbook” by David Duvenaud

# Designing kernels

Inner product in Euclidean space

+

$$K_s(\mathbf{x},\mathbf{y}) = H(\mathbf{x},\mathbf{y}) + G(\mathbf{x},\mathbf{y})$$ ≈ or

$$K_p(\mathbf{x},\mathbf{y}) = H(\mathbf{x},\mathbf{y})G(\mathbf{x},\mathbf{y})$$ ≈ and

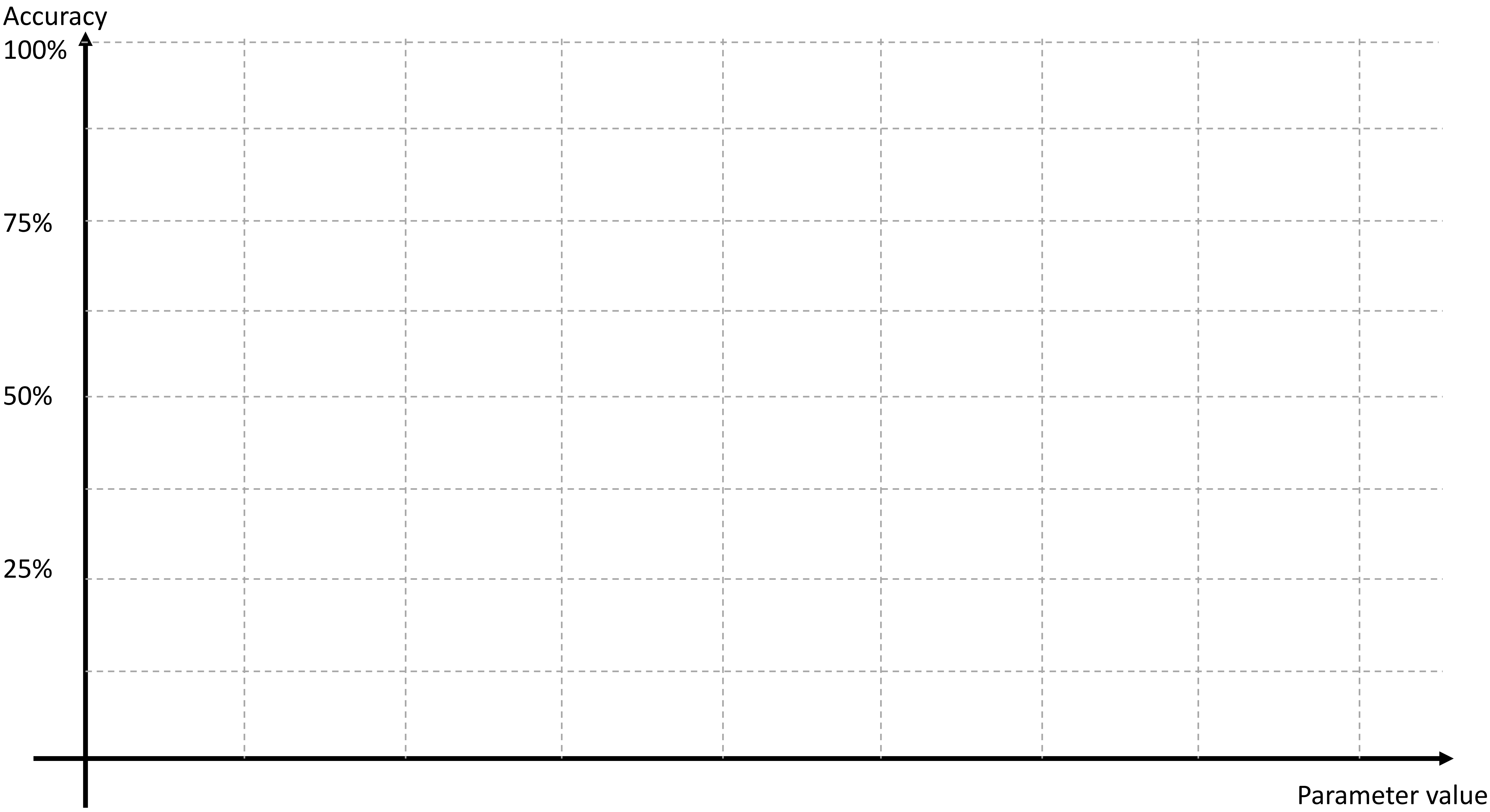
$$K_d(\mathbf{x},\mathbf{y}) = (H(\mathbf{x},\mathbf{y}) + l)^d$$

$$K_g(\mathbf{x},\mathbf{y}) = exp(-\gamma(H(\mathbf{x},\mathbf{x}) - 2H(\mathbf{x},\mathbf{y}) + H(\mathbf{y},\mathbf{y})))$$

$$K_n(\mathbf{x},\mathbf{y}) = \frac{H(\mathbf{x},\mathbf{y})}{\sqrt{H(\mathbf{x},\mathbf{x}).H(\mathbf{y},\mathbf{y})}}$$

# Thanks for listening!

# Lab: Overfitting and using Weka


Accuracy
100%
75%
50%
25%
Parameter value