Data Mining
(Mining Knowledge from Data)

K-Nearest Neighbors

Marcel Jiřina, Pavel Kordík

Lecture

1) Model
2) Tasks and methods of learning models
    1) classification
    2) regression
    3) prediction
    4) clustering
3) Method of K-nearest neighbors
4) Plasticity of model
5) Linear separation

Model and its role in data mining

- The model contains knowledge about the system that generated the data (e.g. dependencies of variables)
- We create it deductively or inductively
  - Deductions - describe the behavior of the system using external knowledge, experience
  - Induction - induces model from data using algorithms
- We will deal with inductive models only
- A high-quality model generalizes well
  - This is the main reason why we use models – they simulate the behavior of the system and know how to cope with the new data (inputs)

Inductive models we create by learning from data

Learning

Classification, regression, prediction

“with the teacher"

(we have information,

how to classify to classes)

clustering

“without the teacher"

(we have no information

how to classify to classes)

Overview of methods that generate methods

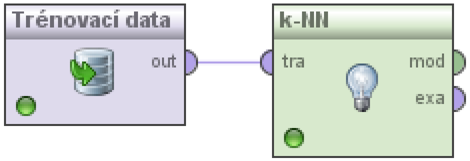
Task	Algorithms
Classification	K-nearest neighbors, Linear separation, decision trees, Bayes classifier, Neural networks.
Regression, forecasting	K-nearest neighbors, Linear regression, Regression trees, Neural networks.
Clustering	K-means, Hierarchical clustering.
Detection of abnormalities, rules, associations	Association rules, K-means.

Algorithms for classification models

- The nearest neighbors
  - 1-NN
  - k-NN
- Linear separation
- Polynomial separation
- Bayesian classification
- Neural Networks
  - Perceptron
  - MLP

Creating and using a model

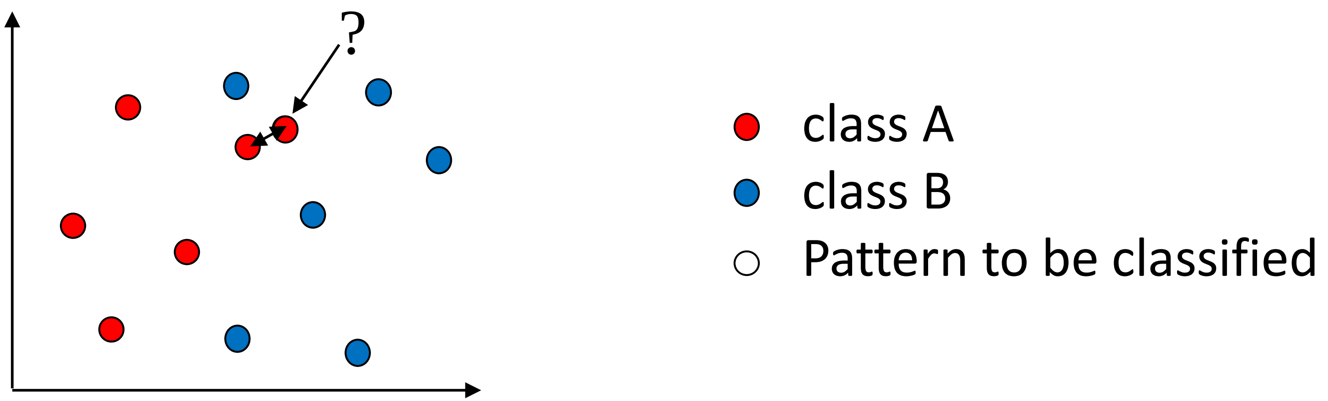
- Two phases
  1. Stage of learning, training
     - The model is generated, internal structure (parameters) are modified



  2. Stage of use, equipping
     - The model is used to calculate output, the model is not modified

1 Nearest Neighbor

1. Training - model generation
   1. Save the training data
2. Classification – use of the model
   1. Find the nearest neighbor and classify to the same class



http://www.theparticle.com/applets/ml/nearest_neighbor/

Metrics, Euclidean distance

- Similarity of patterns must be somehow determined – e.g. by their distance
- Distance must meet certain conditions:
  1. $d(x,y) \geq 0$.
  2. $d(x,y) = 0$ iff $x = y$.
  3. $d(x,y) = d(y,x)$.
  4. $d(x,y) \leq d(x,z) + d(z,y)$ (*triangular inequality*).

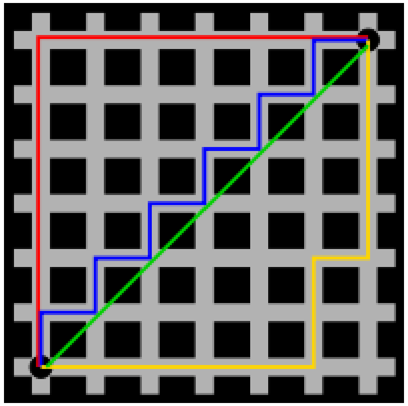
Two points in n-dimensional space: $P = (p_1, p_2, \dots, p_n)$ $Q = (q_1, q_2, \dots, q_n)$

Euclidean distance of P and Q = $\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}.$

- Root extraction is not necessary when comparing distances

Manhattan distance

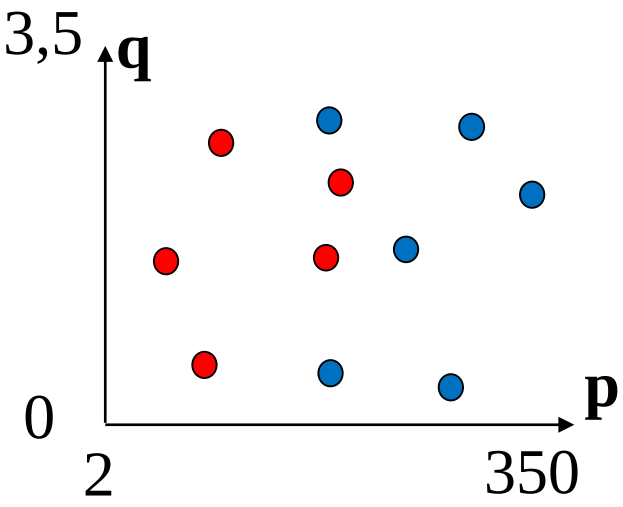
- How do we calculate the distance between two cyclists in Manhattan?



$$M(P,Q)=|p_1-q_1|+|p_2-q_2|+\ldots+|p_n-q_n|$$

Weight of attributes

- Problem - different ranges of distances
- Attributes can have different weight in determining their Euclidean distance – e.g. **p** is 100 times more important than **q**



Normalization of attributes

- The problem can be solved by scaling (normalization) of the attributes:

1. Min-max normalization

$$a_i = \frac{v_i - \min v_i}{\max v_i - \min v_i}$$

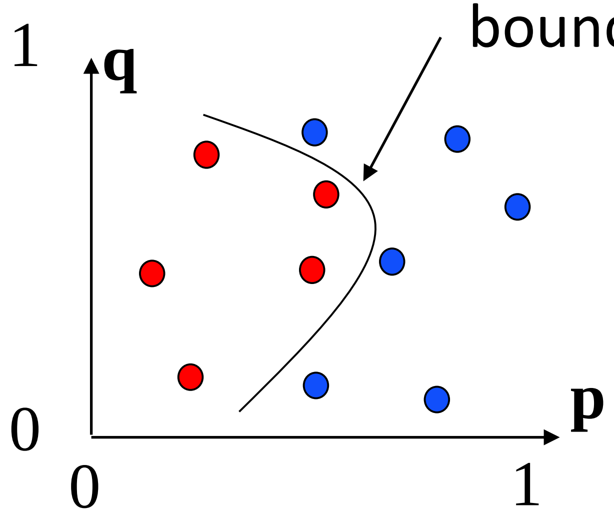
2. Z-score normalization

$$a_i = \frac{v_i - Avg(v_i)}{StDev(v_i)}$$

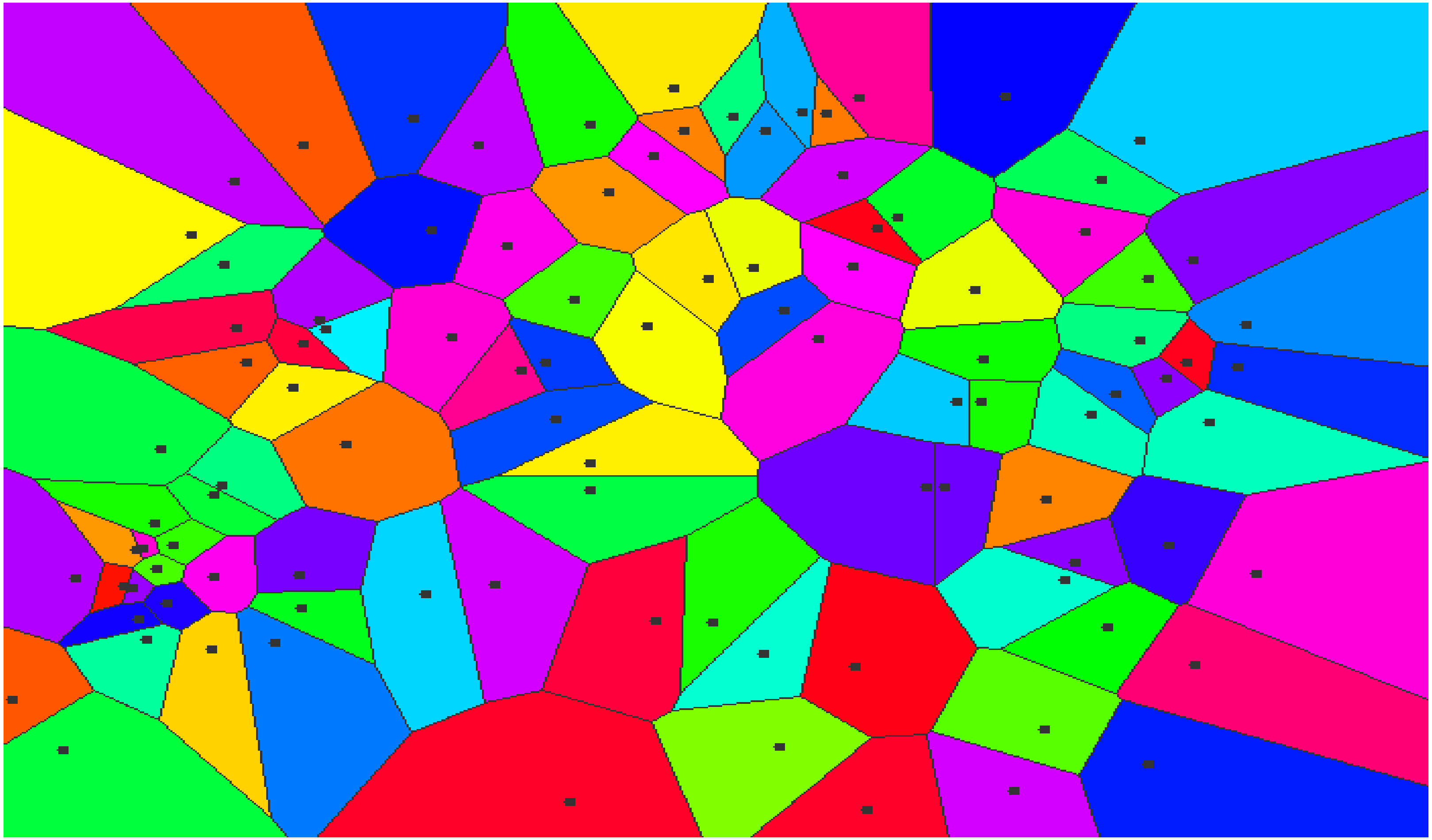
- The original ranges for both transformations are transformed into <0,1>

Decision boundary

Where is exactly the decision-making boundary between classes?



Voronoi diagram

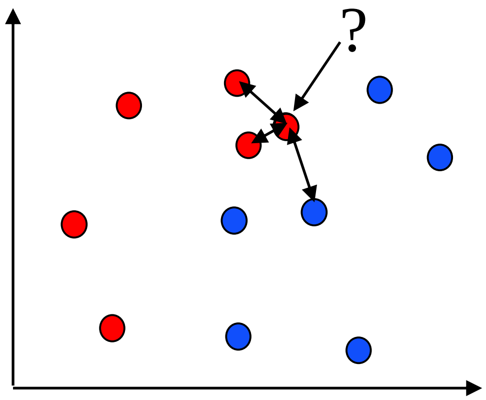


http://www.cs.cornell.edu/Info/People/chew/Delaunay.html

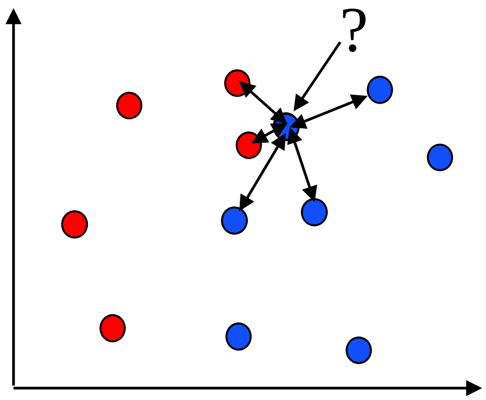
k-NN – k Nearest Neighbors

- Find the closest neighbors and classify to the majority class

3NN classification:

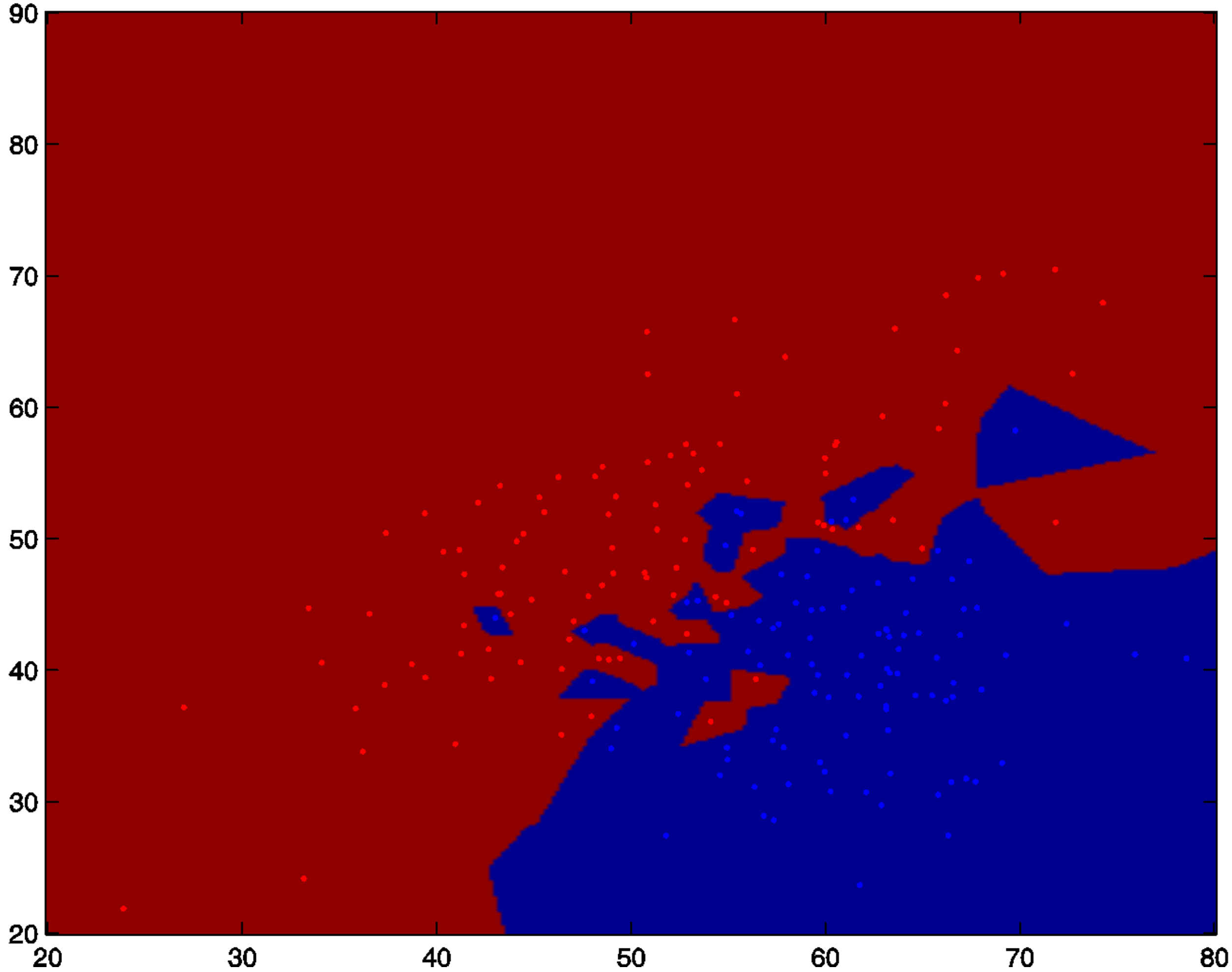


5NN classification:

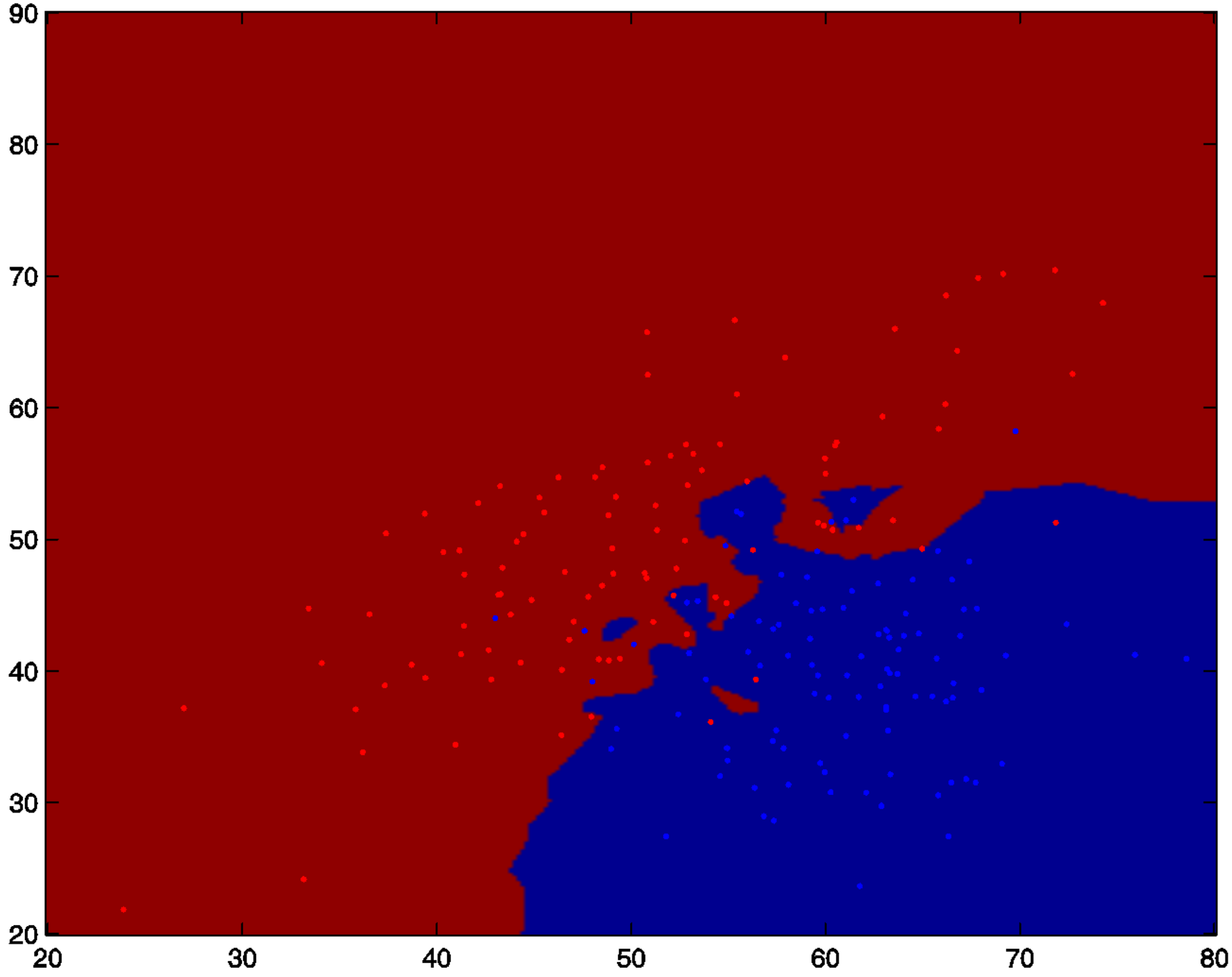


How to choose optimal **k**?

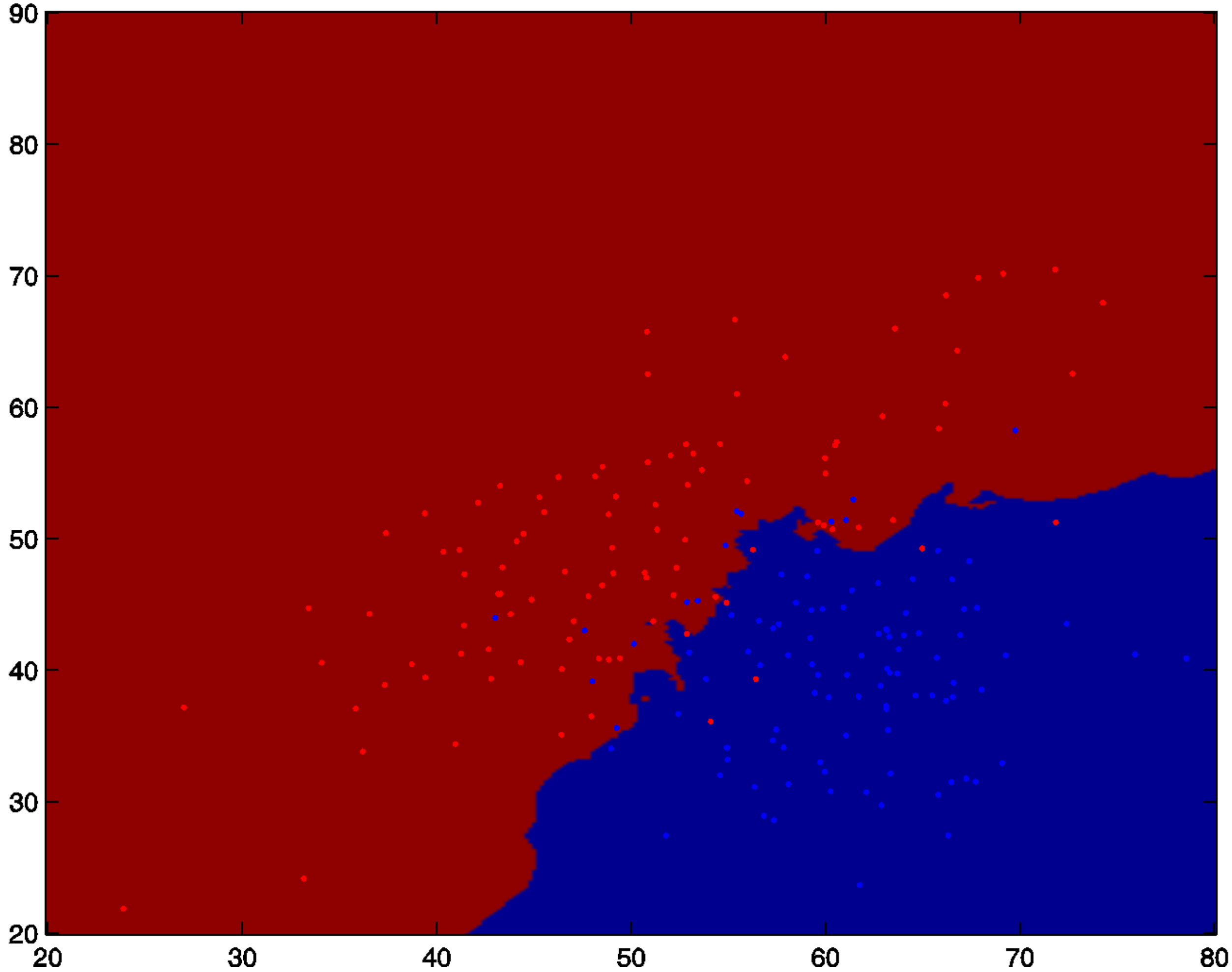
1NN



3NN

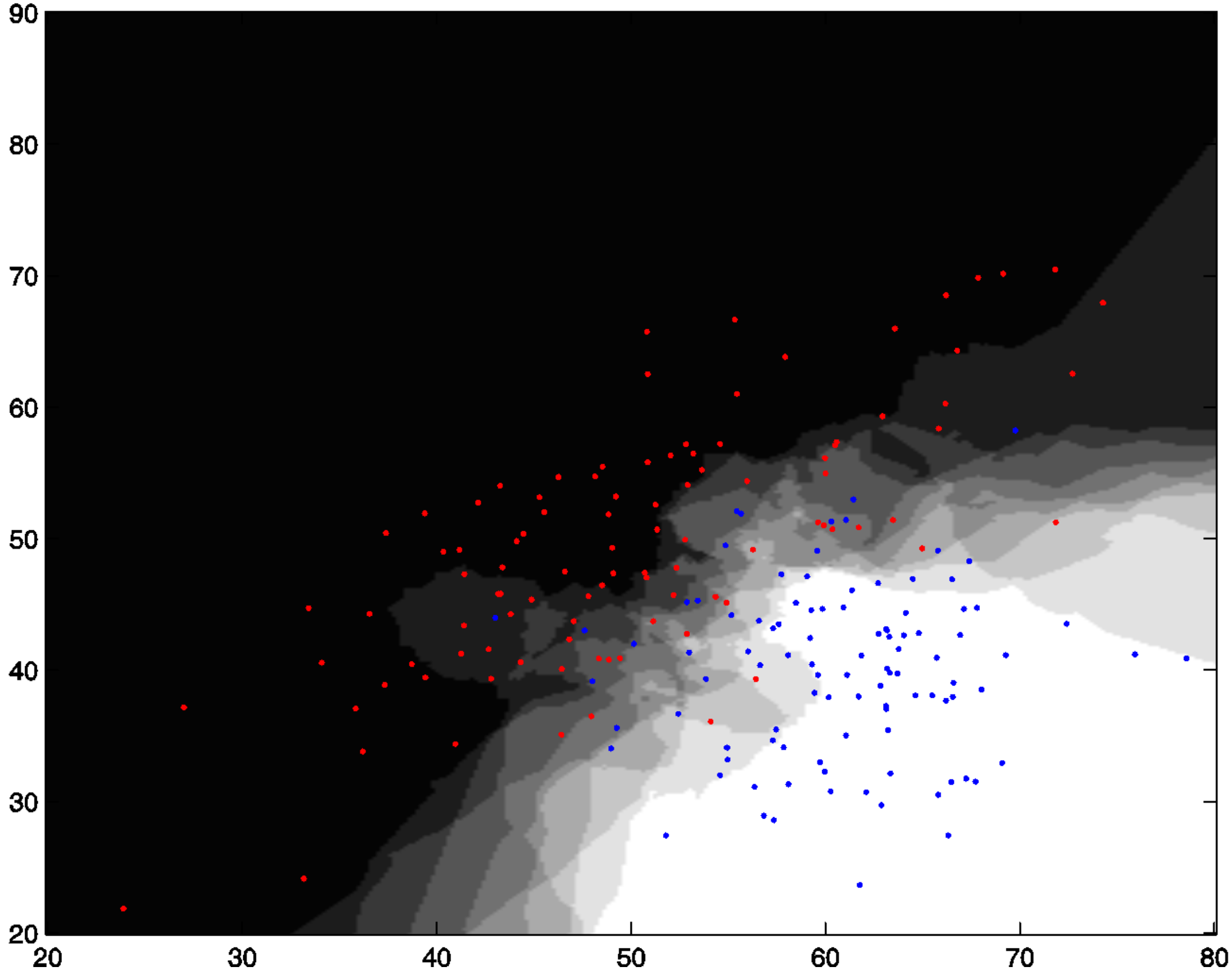


9NN

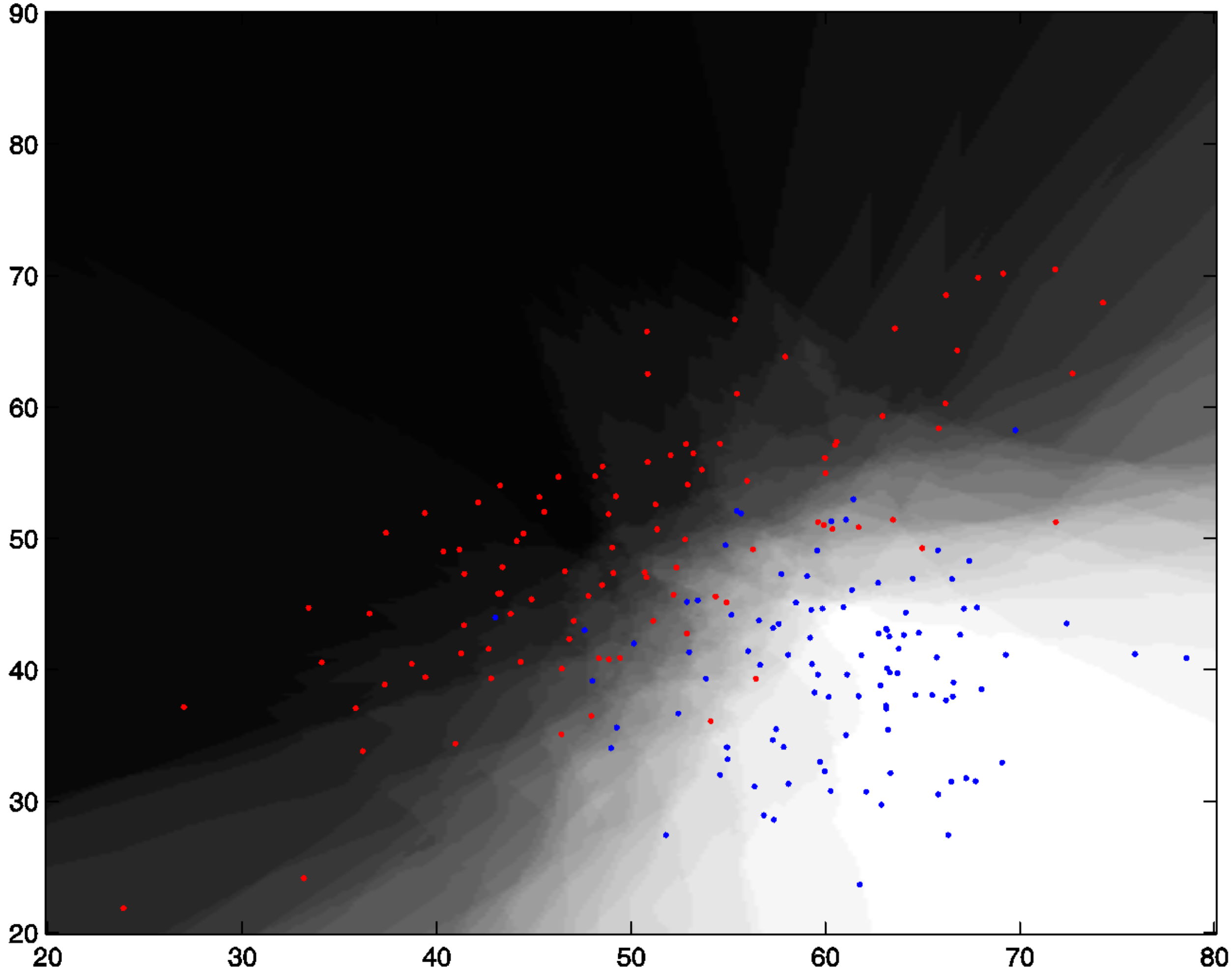


9NN – soft decision

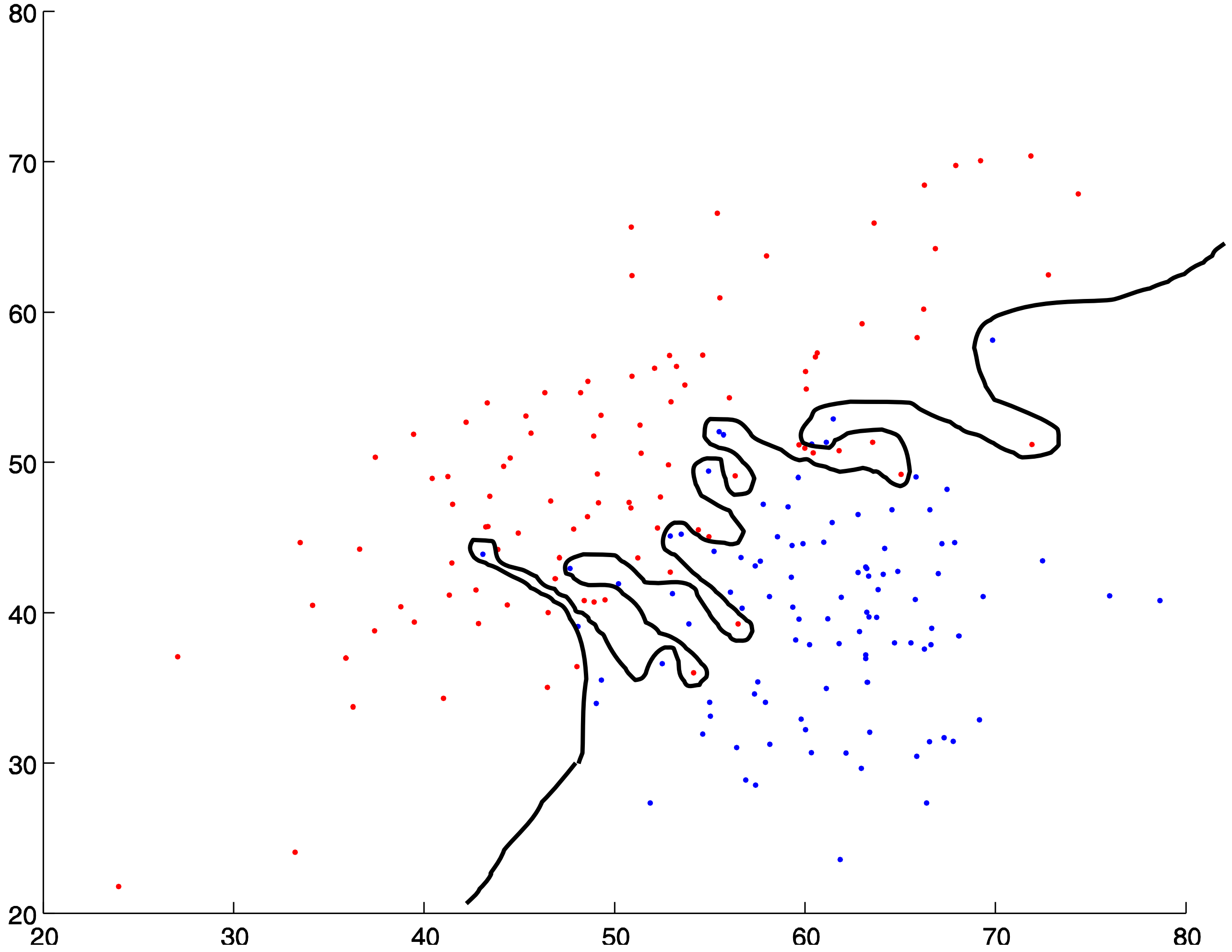
(ratio between the number of neighbors of different classes)



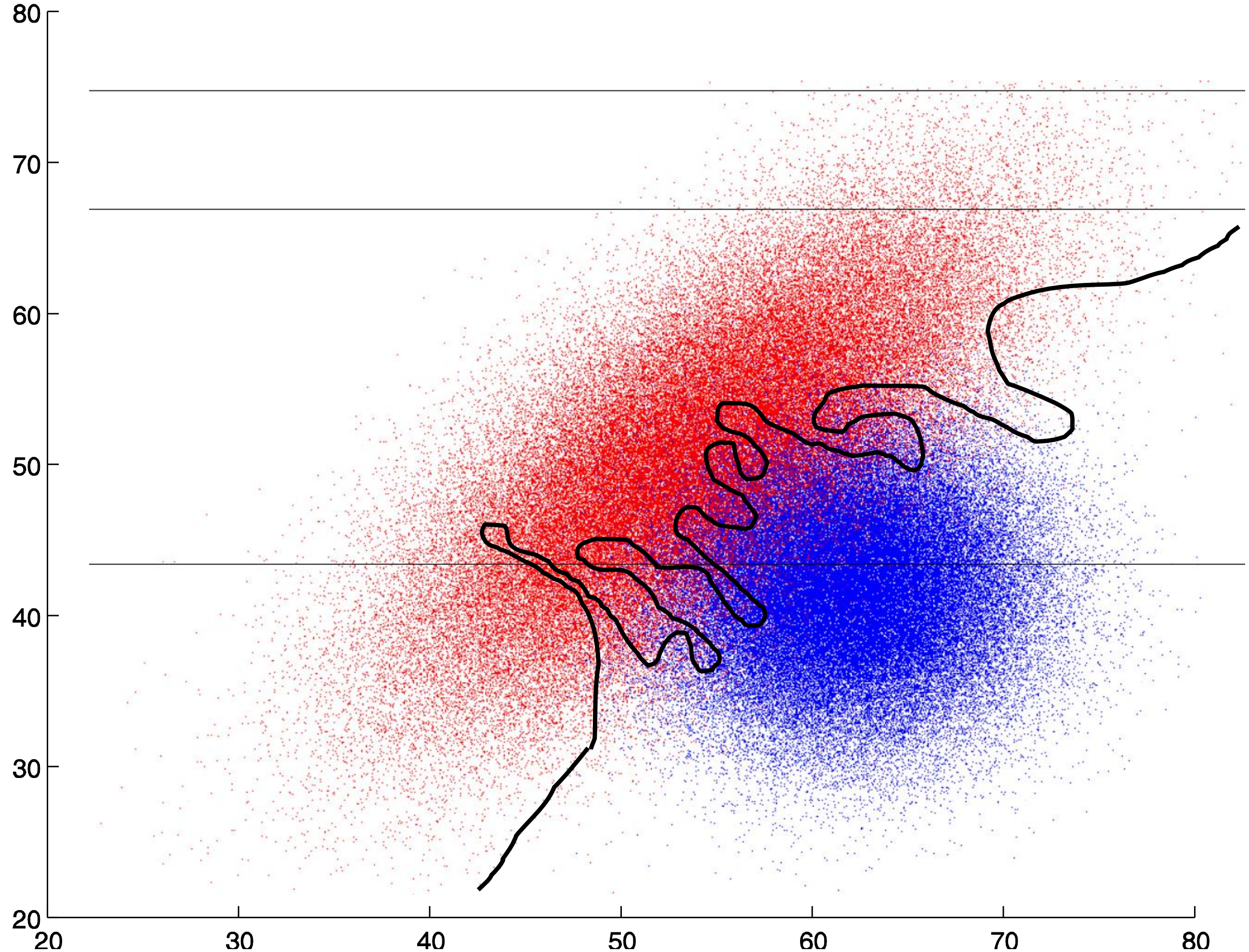
31NN – soft decision



Overtraining



Overtraining

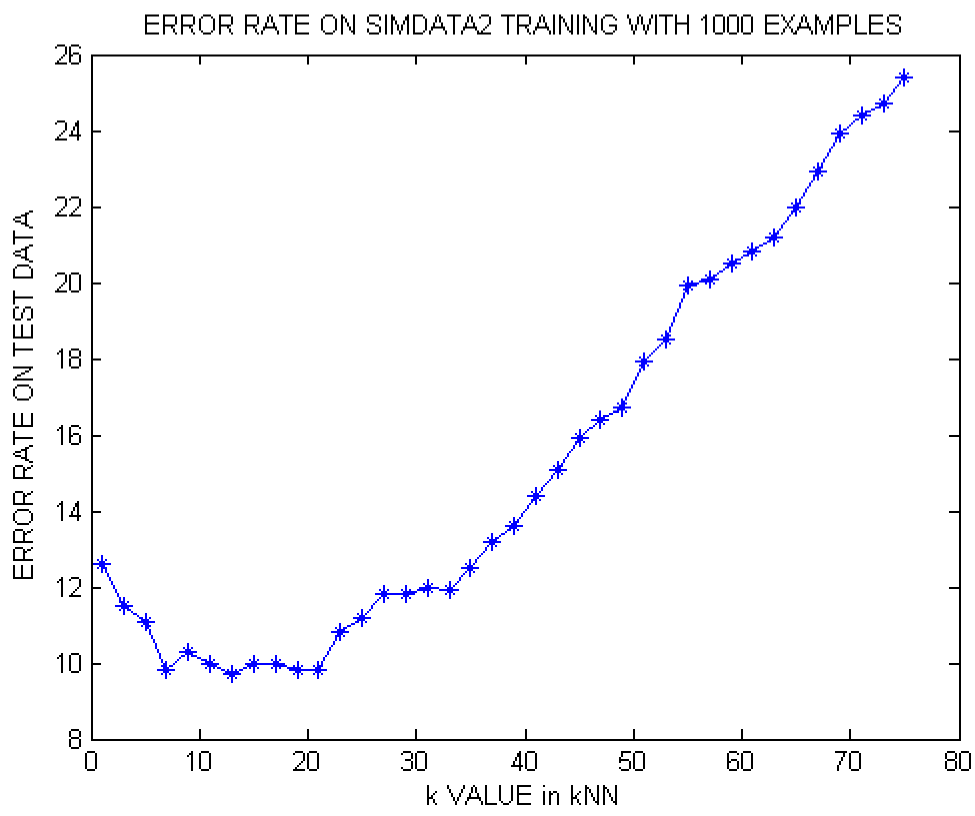


How do I identify overtraining?

- Can the overtraining be identified with the help of new data? How?

- Can the overtraining be already identified during the learning? How?

Evaluation on new (test) data: Dependence of the k-NN algorithm error on k



Source: University of California, Irvine

Variants of k-NN

$$\hat{f}(x_q) \leftarrow \frac{\Sigma_{i=1}^{k} w_i f(x_i)}{\Sigma_{i=1}^{k} w_i}$$

$$w_i \equiv \frac{1}{d(x_q, x_i)^2}$$

- The contribution of a neighbor x_i is weighted by the distance from the classified instance (query point) x_q
- Classification using etalons - select the appropriate subset of the training set

Discussion

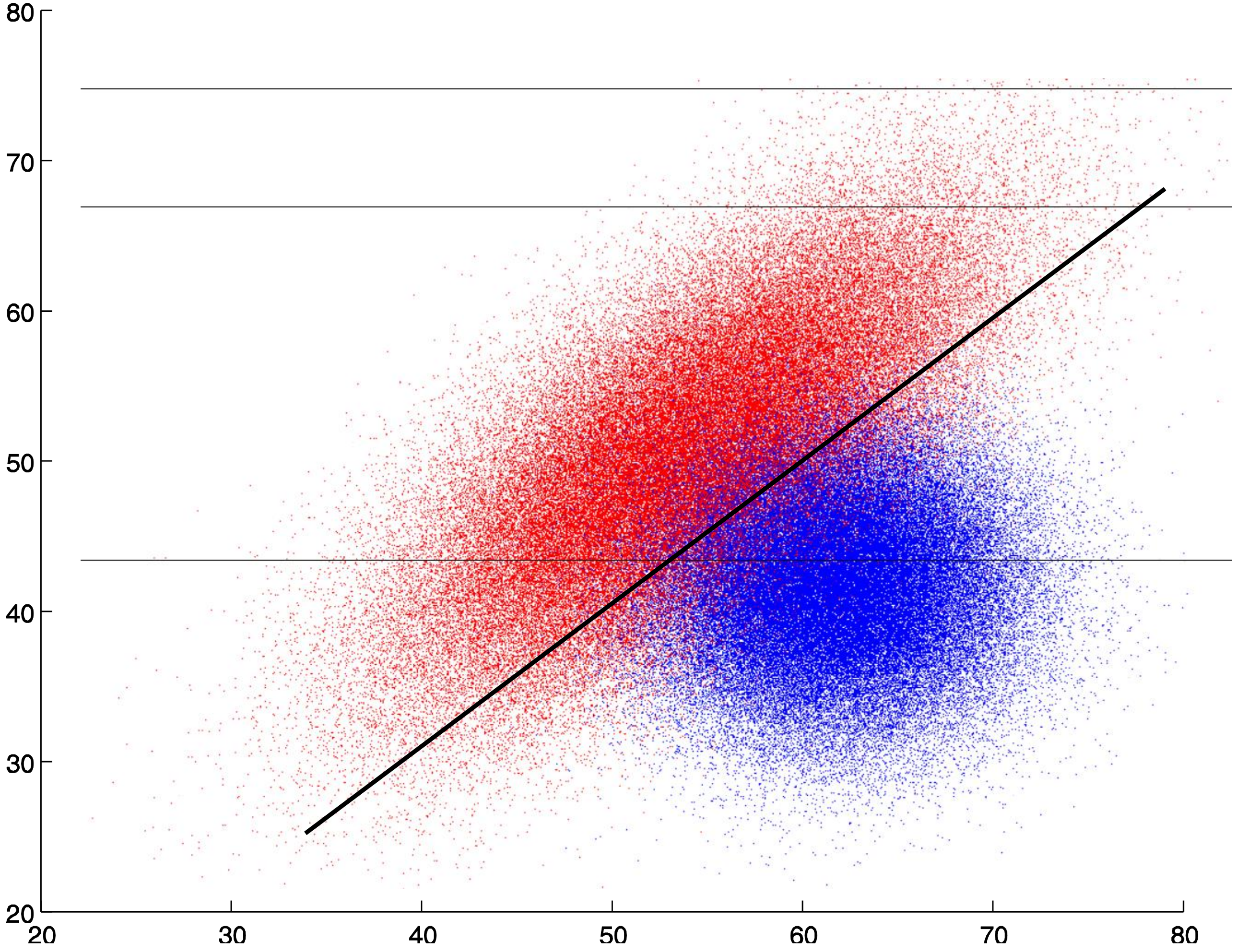
- A very popular method of classification and often very successful
- Immediate creation of a model
- But slow use
  - When classifying, the algorithm must go through the entire training set
- Model is memory-consuming
  - Necessary to remember the whole training set
- Attention to weights of attributes
  - The solution is the data normalization
- It is important to find suitable k
  - To minimize errors on test data

Other algorithms for construction of classifiers

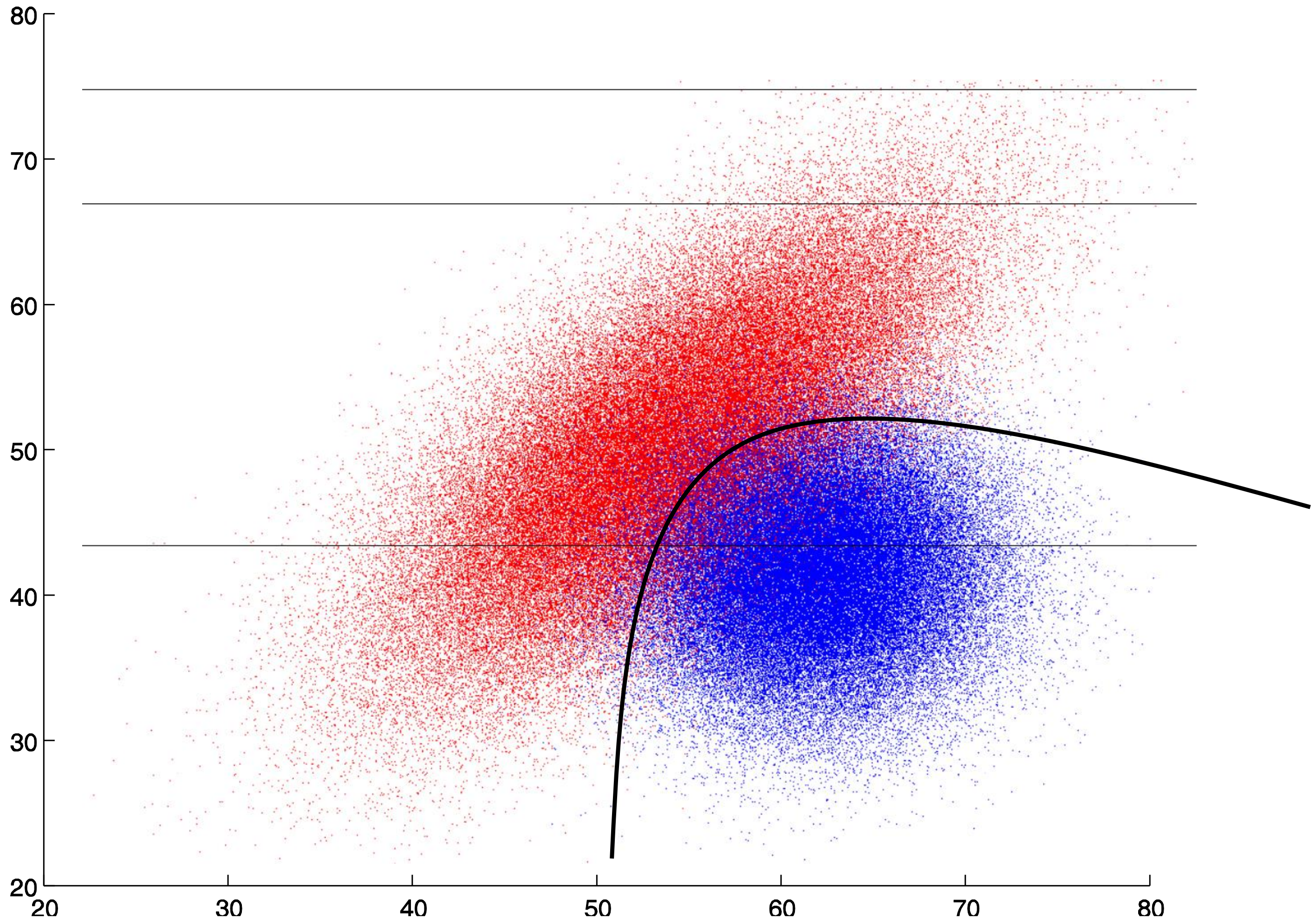
- Linear separation
- Polynomial separation

- During learning we set **parameters** of the classifier
- This time, we no longer need the training data for the recall stage

Linear classifier (separator)



Non-linear classifier (e.g. polynomial classifier)

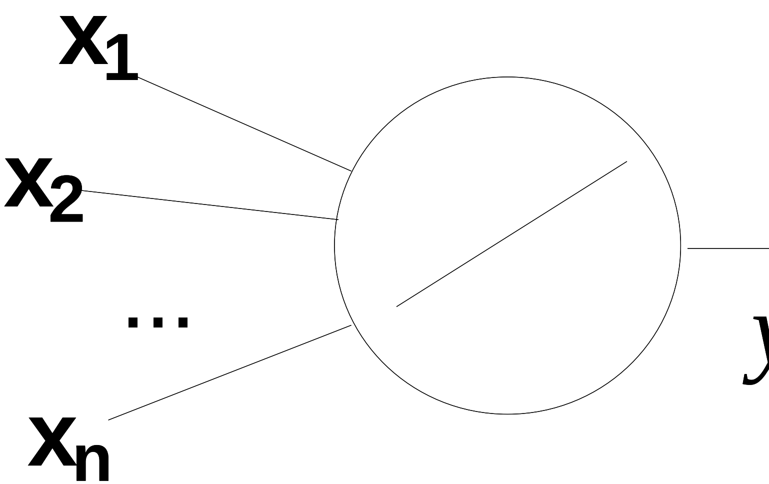


Parameters of a classifier

- How does a linear separator look like and what are its parameters?

- How to determine/learn the parameters?

Determining/learning of parameters $a_1, a_2, \ldots a_n$



Linear model

$$y = \sum_{i=1}^{n} a_i x_i + a_{n+1}$$

Parameters of the model

The generalized least squares method can be used for the calculation :

$\hat{\mathbf{a}} = (X^T X)^{-1} X^T \mathbf{y}$, where **a** is the vector of parameters, *X* is a matrix calculated from the input attributes and **y** is the vector of values of the target variable

LMS algorithm

```
for (int j = 0; j < learning_vectors; j++) x[j][coef-1] = 1;
for (int i = 0; i < coef-1; i++) {
   for (int j = 0; j < learning_vectors; j++) {
         x[j][i] = inputVal[j][i];
   }
}
for (int j = 0; j < learning_vectors; j++) y[j][0] = cValue[j];

Matrix xx = new Matrix(x);
Matrix xt = xx.transpose();
Matrix yy = new Matrix(y);
xx = xt.times(xx);                //x=xT*x
if(xx.det()==0) return false;     //matrix is singular!
xx = xx.inverse();                //power((xT*x):-1)
yy = xt.times(yy);                //// xT*y
Matrix res = xx.times(yy);        //a=(power((xT*x):-1))*xT*y
```

Prediction

- The method of a sliding window