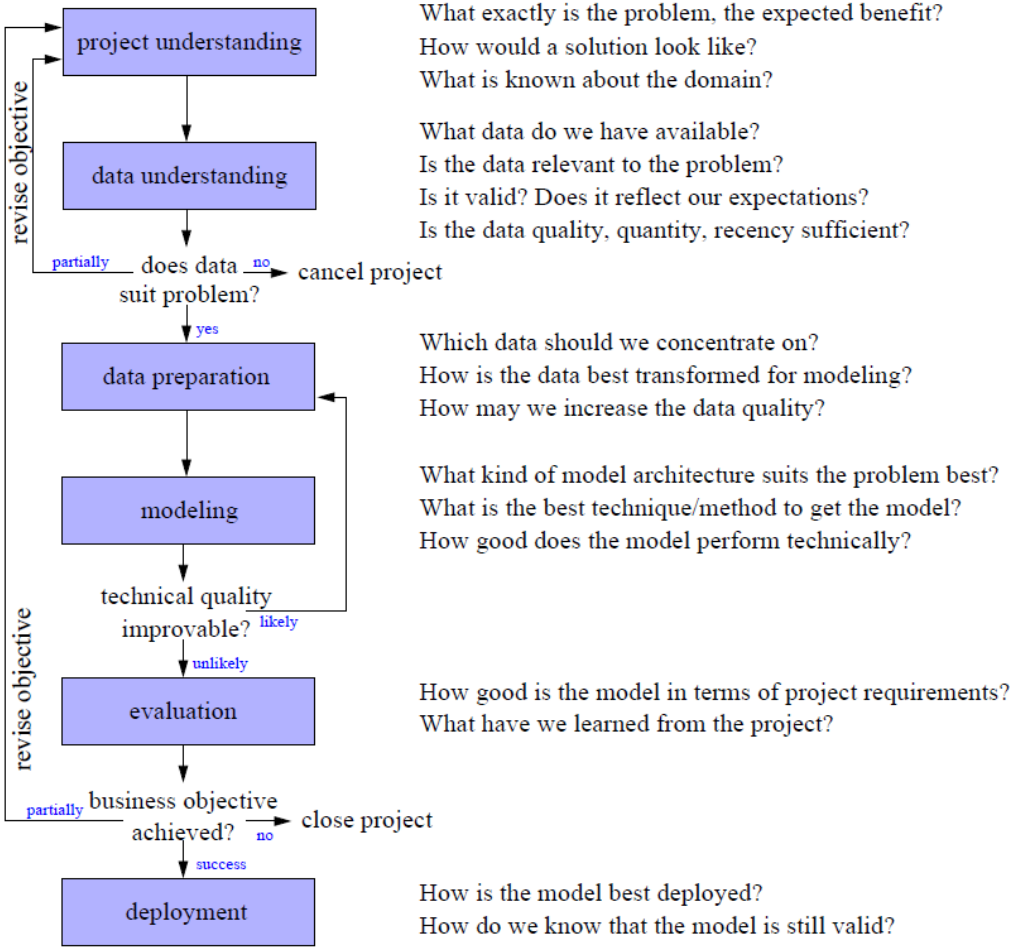
Data Analysis and Knowledge Discovery

Principles of Modelling

Antti Airola

University of Turku
Department of Computing

Where are we now?



Outline

Principles of modelling

- The general data mining task (classification, regression, clustering, associations, deviation analysis) for solving the problem should already be identified within project understanding.
- For each data mining task there are various models that are suitable for solving the given problem.
- Which one is the appropriate model?
- What are the underlying principles of all models?
- How can models be evaluated?

Machine learning

- *machine learning*: field of artificial intelligence concerned with algorithms that can learn from data
- computational statistics, data mining, pattern recognition, statistical learning... closely related
- tools for extracting knowledge out of data in form of a model
- two main branches
  - unsupervised learning
  - supervised learning

Why Machine Learning?

Machine learning can provide new types of capabilities for computers:

- Data mining: extracting new information from medical records, maintenance records, etc.
- Self-customizing programs: Web browser that learns what you like and seeks it out
- Applications one can not program by hand: For example, speech recognition, autonomous driving

The Essence of Machine Learning

1. A pattern exists.
2. We can not pin it down mathematically.
3. We have data on it.

Supervised learning

- data: (input, correct output) -pairs (features, label)
- features: real-valued or categorical
- label: real-valued (regression) or categorical (classification)
- labels available for training data, unknown for future data
- goal: model dependency between the features and the label
- model predicts labels for new instances

Supervised learning

In supervised learning we are given a set of input-output pairs

$$\{(x_1, y_1), \ldots, (x_n, y_n)\}$$

that we call a training set.

- Classification: A learning problem with output values taken from a finite unordered set $C = \{c_1, \ldots, c_k\}$. A special case is binary classification where $y_i \in \{-1, 1\}$.
- Regression: A learning problem whose output values are real $y_i \in \mathbb{R}$.

Supervised learning problems

- Junk-mail filtering
  - features: word frequencies
  - class: junk / non-junk
- Access control system
  - features: generated using feature extraction methods from camera image of face
  - class: id of a person
- Medical diagnosis
  - features: BMI, age, symptoms, prior diagnoses, text in patient documentation, blood samples...
  - class: ICD10 diagnostic code

Components Of Learning

Metaphor: Credit approval

Applicant information

age	23 years
gender	male
annual salary	30,000 €
years in residence	1 year
years in job	1 year
current debt	15,000 €
...	...

Approve credit?

Components Of Learning

Formalization:

- Input: $\mathbf{x}$ (customer application)
- Output: y (good/bad customer?)
- Target function: $f : \mathcal{X} \to \mathcal{Y}$ (ideal credit approval formula)
- Data: $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ (historical records)

$$\downarrow$$

- Hypothesis: $g : \mathcal{X} \to \mathcal{Y}$

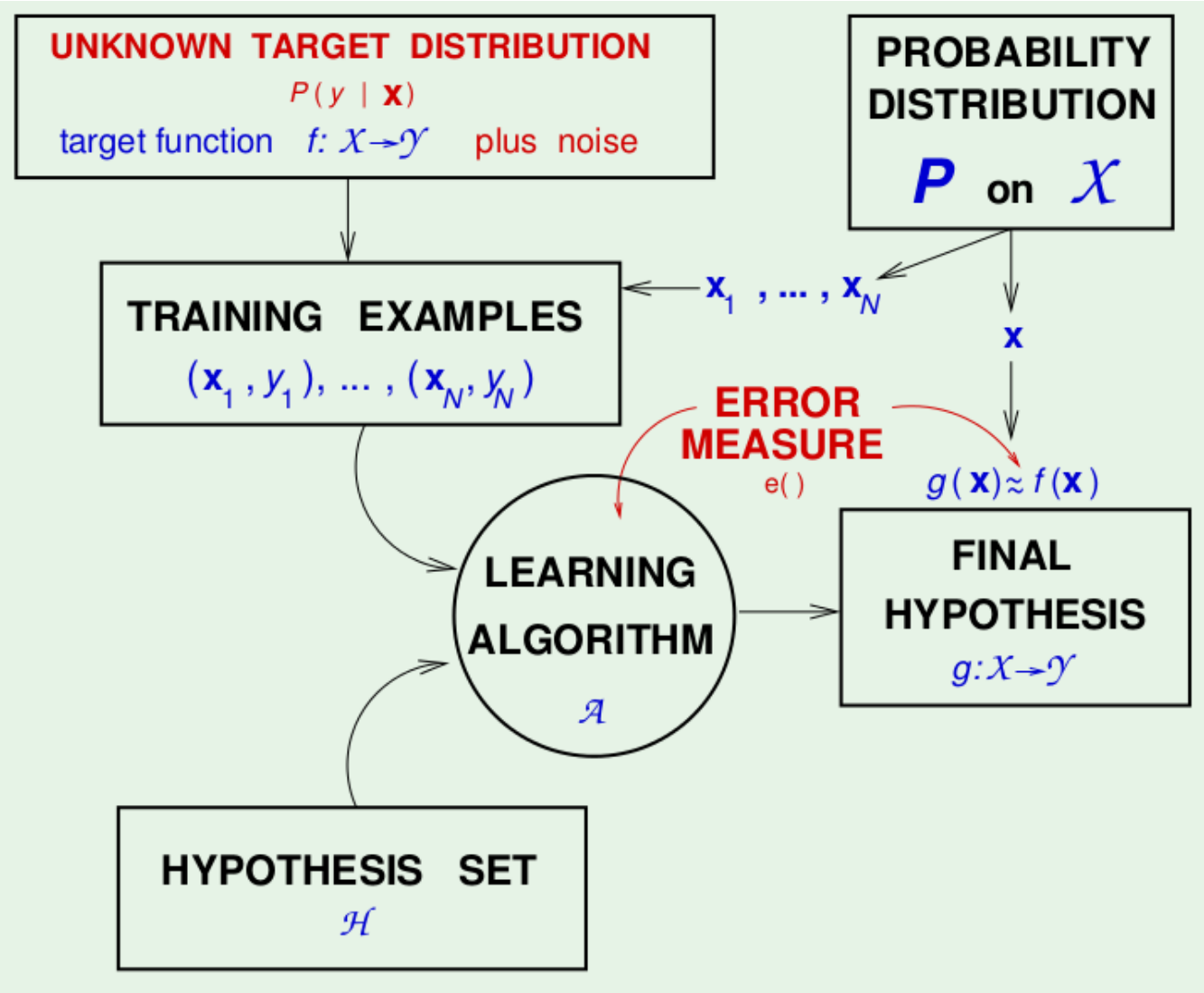
Hypothesis set

- The hypothesis set, $\mathcal{H}$, is the space of functions where we look for our solution.
- For many algorithms (such as optimization algorithms) it is the space the algorithm is allowed to search.
- Supervised learning uses the training data to learn a function

$$g : \mathcal{X} \to \mathcal{Y}$$

from $\mathcal{H}$, that can be applied to previously unseen data: $y_{pred} = g(x_{new})$

Components Of Learning



Modelling steps

Building blocks for a machine learning algorithm:

Model class: general structure of the model (hypothesis set) e.g.:

- linear or quadratic function
- decision tree, neural network
- division of data to clusters
- ...

Error measure (Score function): evaluates quality of different models

- does the model fit data?
  - squared error, cluster variance...
- how complex is the model?
  - e.g. minimum description length

Algorithm: find good model, as defined by score function

- mathematical optimization

Modelling steps

Final step: Validation (supervised learning)

- finding best fit to training data does not guarantee accurate predictions on new data
- danger of overfitting:
  - worst case: model just memorizes training data, does not generalize beyond it
  - e.g. a very complex function predicts whether you pay back your loan or not based on social security number
  - 100% accuracy on training data, fails to outperform random guessing on new instances
- alternatively underfitting:
  - model class not expressive enough, e.g. linear functions on non-linear problems

Approximation-generalization trade-off

- The aim of learning is to approximate the target function f as closely as possible.
- Small prediction error: good approximation of the target function f outside of the training set.
- More complex hypothesis set $\mathcal{H} \Rightarrow$ better change of approximating f.
- Less complex hypothesis set $\mathcal{H} \Rightarrow$ better change of generalizing f outside of the training set.
- Having the ideal hypothesis set $\mathcal{H} = \{f\}$ means we already know the answer and there is no need for machine learning.

Grue Emerald Paradox

What if we allow $\mathcal{H}$ to contain all possible functions in the hypothesis set?

Grue Emerald Paradox

Suppose we have seen a large set of emeralds and they are all green. Consider the followig two alternative hypotheses:

- Hypothesis 1: All emeralds are green.
- Hypothesis 2: All emeralds are blue except the ones we have seen so far.

Based on the training set alone, there is no means of choosing which one is better. On the test data, however, they give completely opposite results.

Occam's Razor

- Occam's razor is a principle that favors the simplest hypothesis that can well explain a given set of observations
- Given a simple hypothesis A, and a much more complex hypothesis B which explains the existing data only slightly better than A, A is likely to explain future data better than B.
- Works also on hypothesis sets $\mathcal{H}$

Another example: Polynomial Curve Fitting

M^{th} Order Polynomial

$$f_{\mathbf{w}}(x) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M$$

Denoting with

$$\mathbf{w} = (w_0, w_1, \ldots, w_M)$$

the $M + 1$ dimensional vector of polynomial weights, the hypothesis set of all Mth oder polynomials corresponds to the set

$$\mathbb{R}^{M+1}$$

of $M + 1$ dimensional real valued vectors.

Recap: Polynomial Curve Fitting

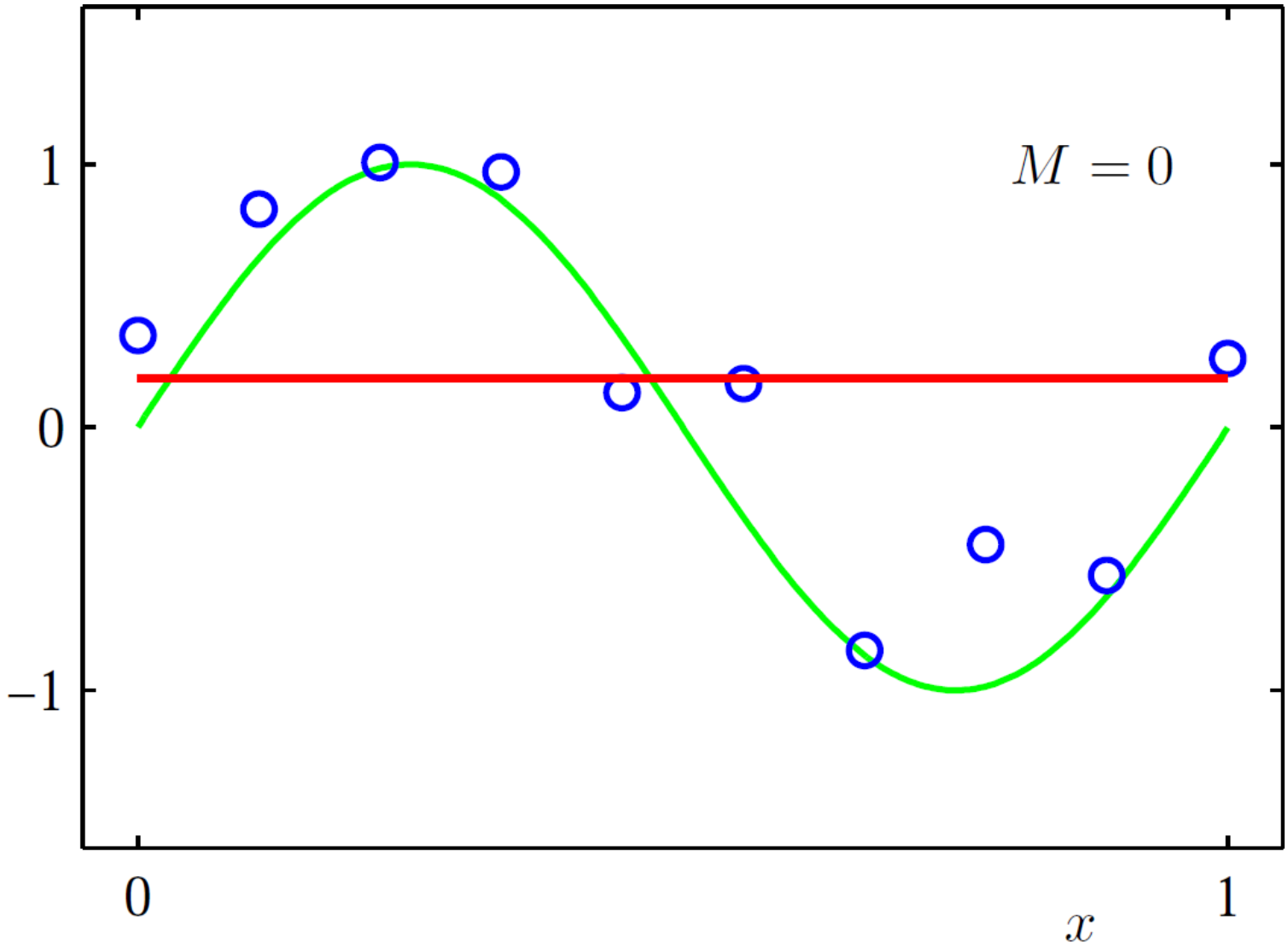
Error on the training set with the mean squared error measure

$$R(f_{\mathbf{w}}) = \frac{1}{N} \sum_{i=1}^{N} \left(f\left(x_i\right) - y_i\right)^2$$

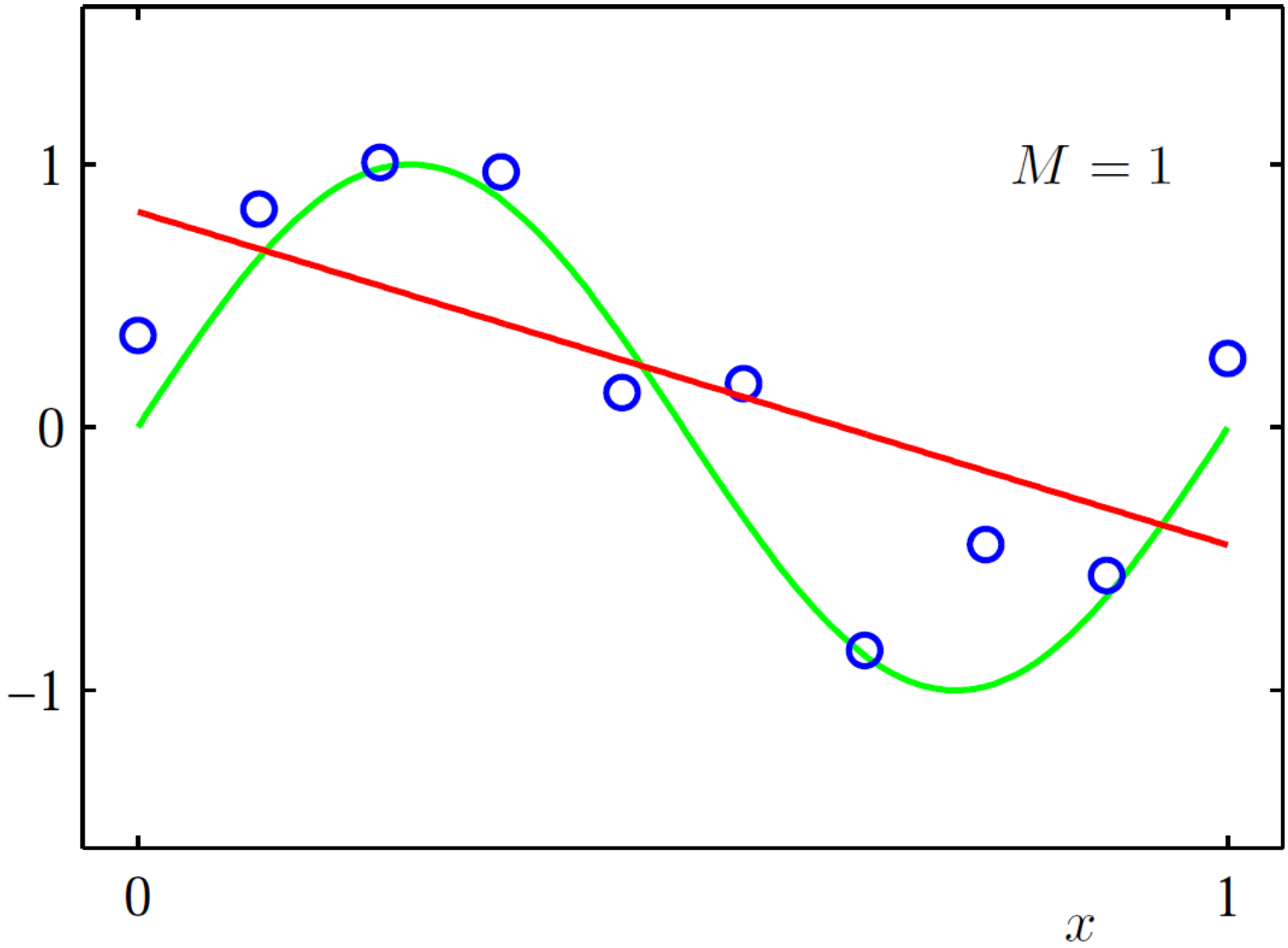
Select the hypothesis (e.g. the polynomial of degree M) with the smallest error:

$$\arg\min_{\mathbf{w} \in \mathbb{R}^{M+1}} R(f_{\mathbf{w}})$$

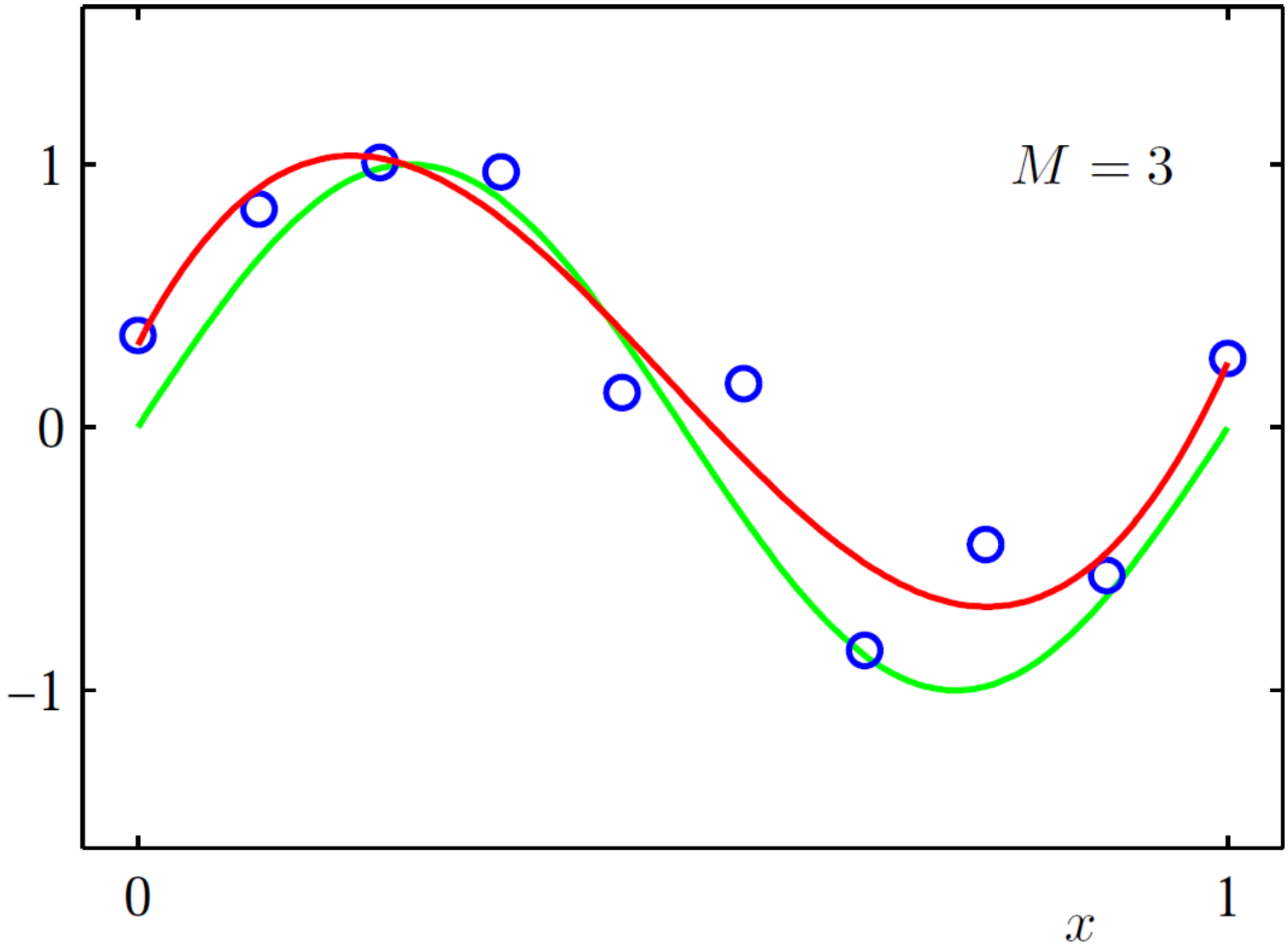
0^{th} Order Polynomial



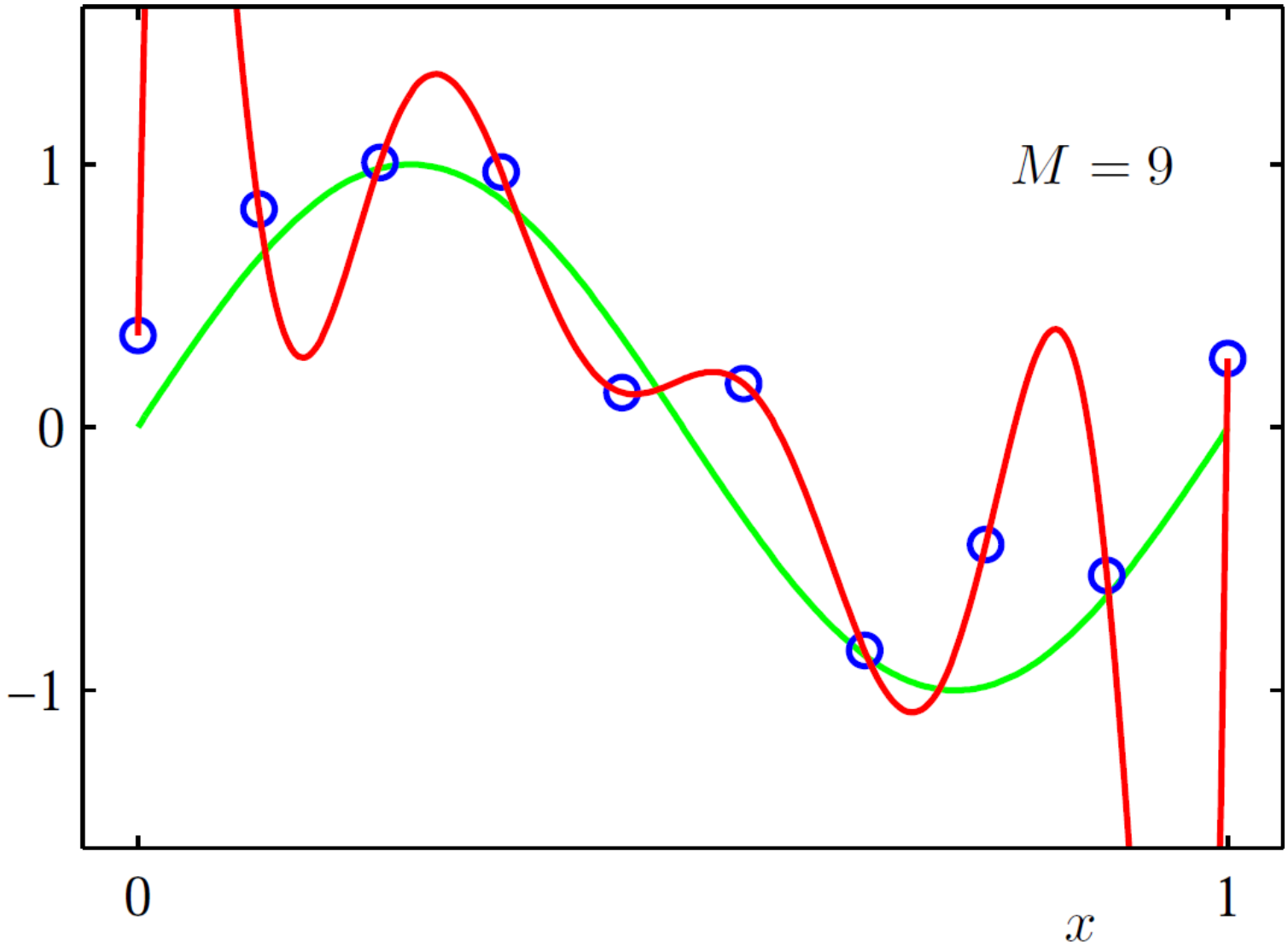
1st Order Polynomial



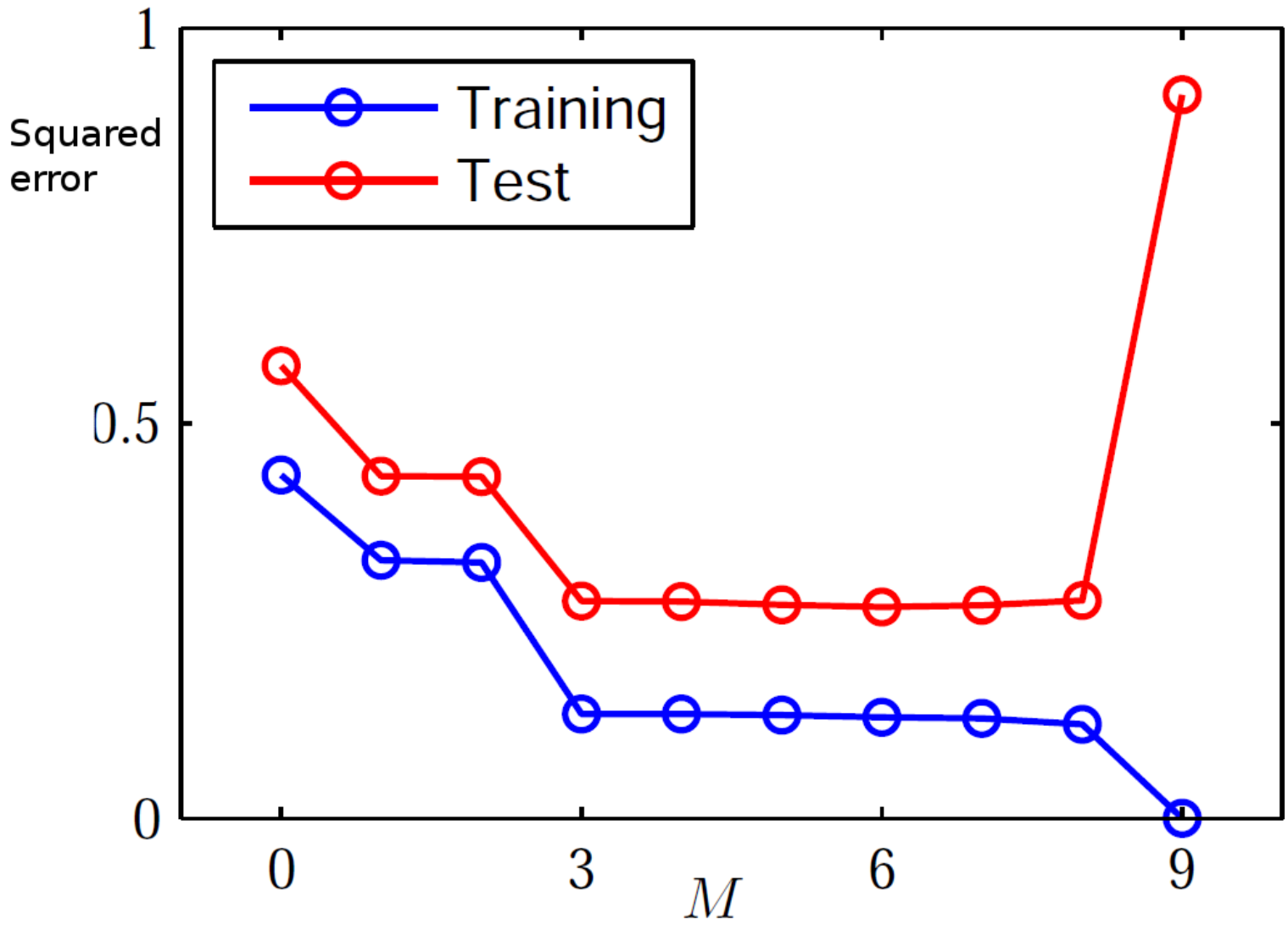
3rd Order Polynomial



9^{th} Order Polynomial



Over-fitting



Model classes: Simple examples

(a) Linear models. The simplest case of a linear model is a regression line

$$y = ax + b,$$

describing the idealized relationship between attributes x and y.

- The parameters a and b still need to be determined.
- They should be chosen in such a way that the regression line fits best to the given data.
- A perfect fit, i.e. the data points lie exactly on the regression line, cannot be expected.
- To fit the regression line to the data, a criterion or error measures is required, defining how well a line with given parameters a and b fits to the data.

Model classes: Simple examples

(a) Linear models. More generally, the prediction of a linear model is based on several features

$$y = w_1 x_1 + w_2 x_2 + ... + w_d x_d + b,$$

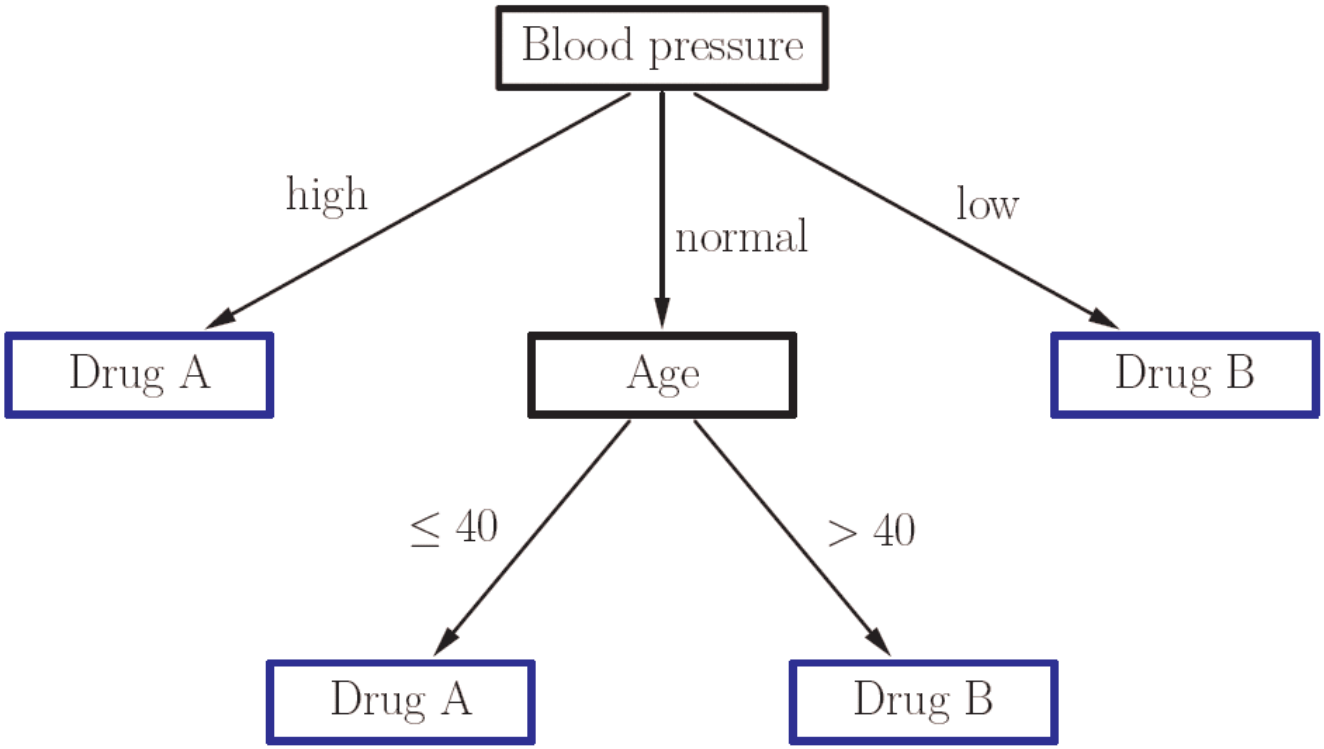
describing the idealized relationship between the feature vector $[x_1, ..., x_d]$ and y.

Model classes: Simple examples

- Example models for nominal data: association rules like
  - *"If temperature=cold and precipitation=rain then action=read book".*
  - *"If a customer buys product A, then he also buys product B with a probability of 30%".*

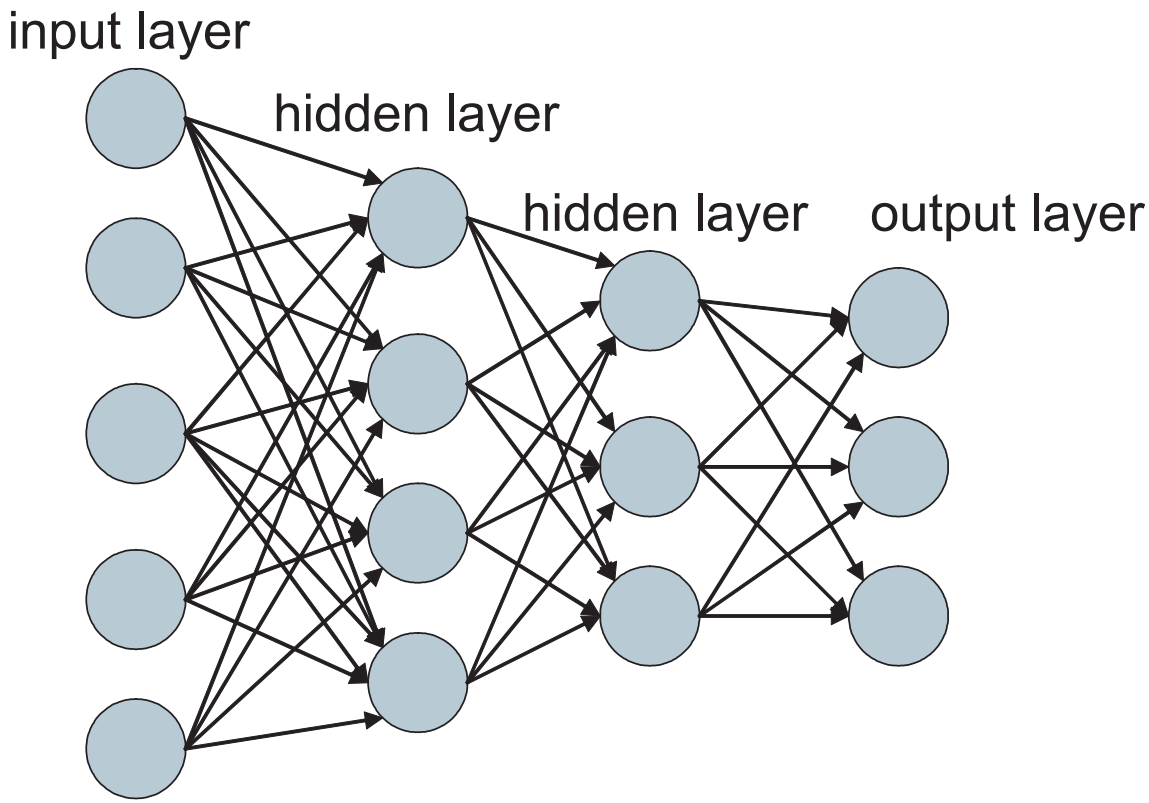
Model classes: decision trees

Assignment of a drug to a patient:



Model classes: neural networks

A feedforward neural network



How to find the best model?

- Project understanding will restrict the model class, but will usually not provide strict definition of the model class.
  - Example: If the data analysis task is identified as a regression problem, clustering models are not suitable.
  - But it is not specified which kind of regression function should be chosen.
- The model class must still be selected.
  - The choice of the wrong model class can spoil the whole data analysis process.
- It must be formalised what we mean by a "good" model.

Criteria for "good" models

Fitting criterion: How well does the model fit the data?

Model complexity: Usually, simpler models are preferred as they

- are easier to understand,
- easier to fit,
- avoid overfitting.
  Overfitting refers to the problem that complex models might be flexible enough to fit the given data almost perfectly – learning them by heart – without representing the general structure in the data.

Criteria for "good" models

Interpretability is often desired.

- In some cases, a black box model suffices.
  A black box model is function – a black box – that computes suitable outputs for inputs, but the structure of the function or its coefficients do not have any specific meaning.

  Interpretability is context dependent.
  Example. A quadratic function (in the parameter time) might be perfectly interpretable in the context of a mechanical process where acceleration, velocity and distance are involved, whereas in a different setting the coefficients of a quadratic function might have no meaning.

Criteria for "good" models

Computational aspects: Finding the best, or at least a good, model w.r.t. the criteria data fitting, model complexity and interpretability is also a matter of computational complexity.

- In some cases, it might be very easy to find the best model w.r.t. to a given criterion.
- For certain methods such as least-squares linear or logistic regression, simple and fast algorithms that are guaranteed to find globally optimal solutions (convex optimization).
- Complex methods such as deep neural networks can require more complex optimization methods, GPU-hardware and only locally optimal solutions guaranteed (non-convex optimization).

Controlling the complexity of $\mathcal{H}$

The complexity of the hypothesis set $\mathcal{H}$ is usually controlled with hyper-parameters.

- The max degree of polynomials in the above example
- The number of neighbours in the k-nearest neighbour method
- The regularization parameter value for many methods (ridge regression, SVM, regularized logistic regression...)
- Depth of decision tree
- The number of layers and the number of neurons per hidden layer with neural networks
- You might also be comparing models produced by different algorithms altogether
- ...

Practical advice

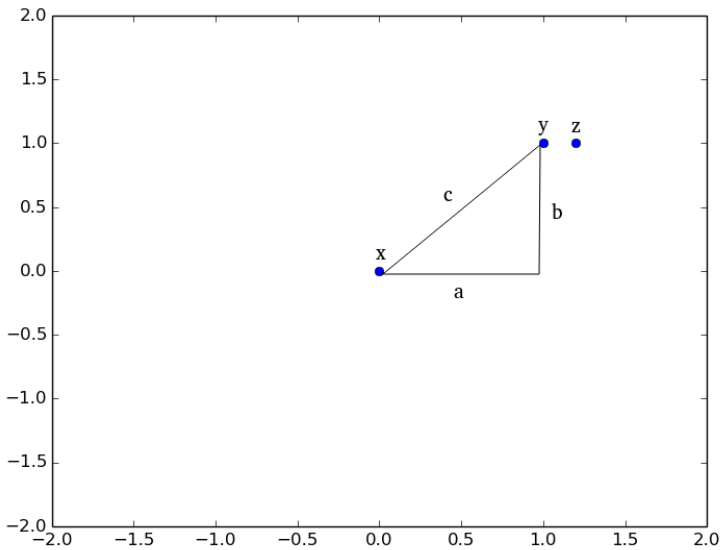
- figuring out the right features often much more important than the choice of learning method
- start with simple methods (linear regression, k-nearest neihgbour, Naive Bayes...) rather than complex ones (support vector machines with kernels, deep neural networks, Bayesian networks...)
  - easier to understand
  - easy to find or program yourself good implementations
  - need less tuning, less risk of overfitting
  - in many practical problems often give just as good results as more advanced methods
- once you have done your best using a simple approach, you can check if you can do better with more advanced ones
- much more useful to know a few basic methods well, than 20 methods badly

Outline

Method of k nearest neighbors

- k nearest neighbors (k-NN)
- Classical method (e.g. Fix & Hodges 1951)
- Simple: classification based on k most similar training instances
- Parameter k tunes model complexity
- Can learn complex non-linear functions
- Also obvious limitations

Distance



- $\mathbf{x} = [0, 0]$, $\mathbf{y} = [1, 1]$, $\mathbf{z} = [1.2, 1]$
- $c^2 = a^2 + b^2$ (Pythagoraean theorem)
- Euclidian distance: $d(\mathbf{x}, \mathbf{y}) = \sqrt{(1-0)^2 + (1-0)^2} = \sqrt{2}$
- Similarly $d(\mathbf{y}, \mathbf{z}) = \sqrt{(1-1.2)^2 + (1-1)^2} = 0.2$
- $d(\mathbf{y}, \mathbf{z}) < d(\mathbf{x}, \mathbf{y})$, meaning that $\mathbf{z}$ is closer to $\mathbf{y}$ than $\mathbf{x}$

Distance

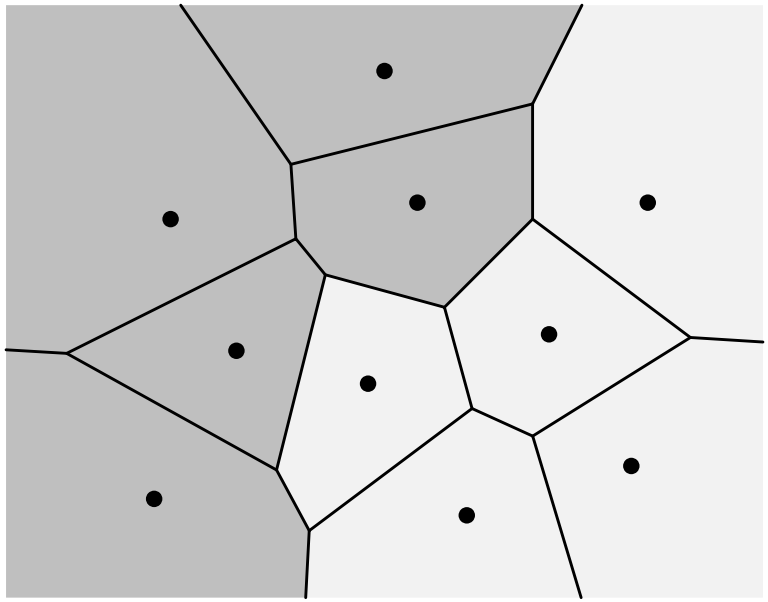
- Basic geometric concepts generalize to higher dimensions
- (Euclidean) distance: $d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{d} (\mathbf{x}_i - \mathbf{y}_i)^2}$
- Inner (dot) product: $\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{d} \mathbf{x}_i \mathbf{y}_i$
- When $\mathbf{x}$ and $\mathbf{y}$ considered column vectors, may equivalently be written as the matrix product $\mathbf{x}^T \mathbf{y}$
- In addition to standard Euclidean distance measure, also many others can be defined

Nearest neighbour predictors

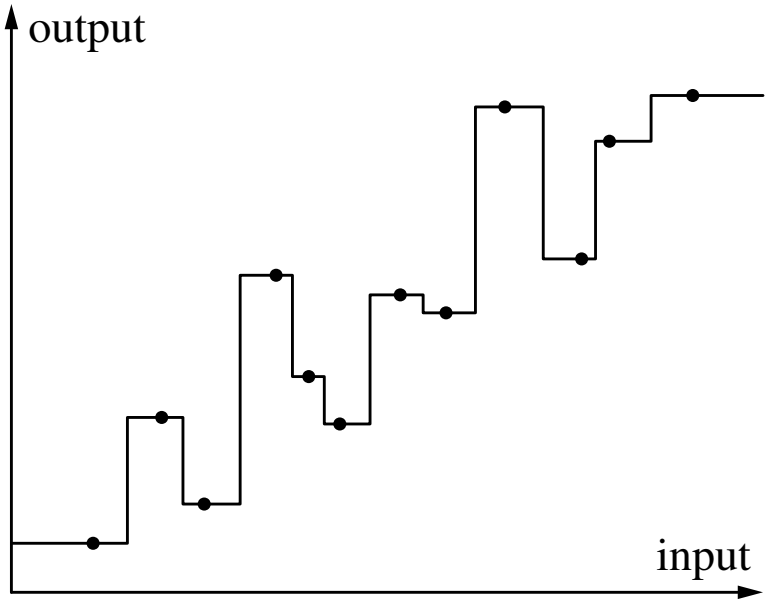
- Nearest neighbour predictors are special case of instance-based learning.
- Basic idea: similar instances should have similar class-labels or real-valued labels (classification / regression)
- Similarity defined in terms of distance: the smaller the distance between two feature vectors the more similar the corresponding instances are considered

Simple nearest neighbour predictor

For a new instance, use the target value of the closest neighbour in the training set.



Classification



Regression

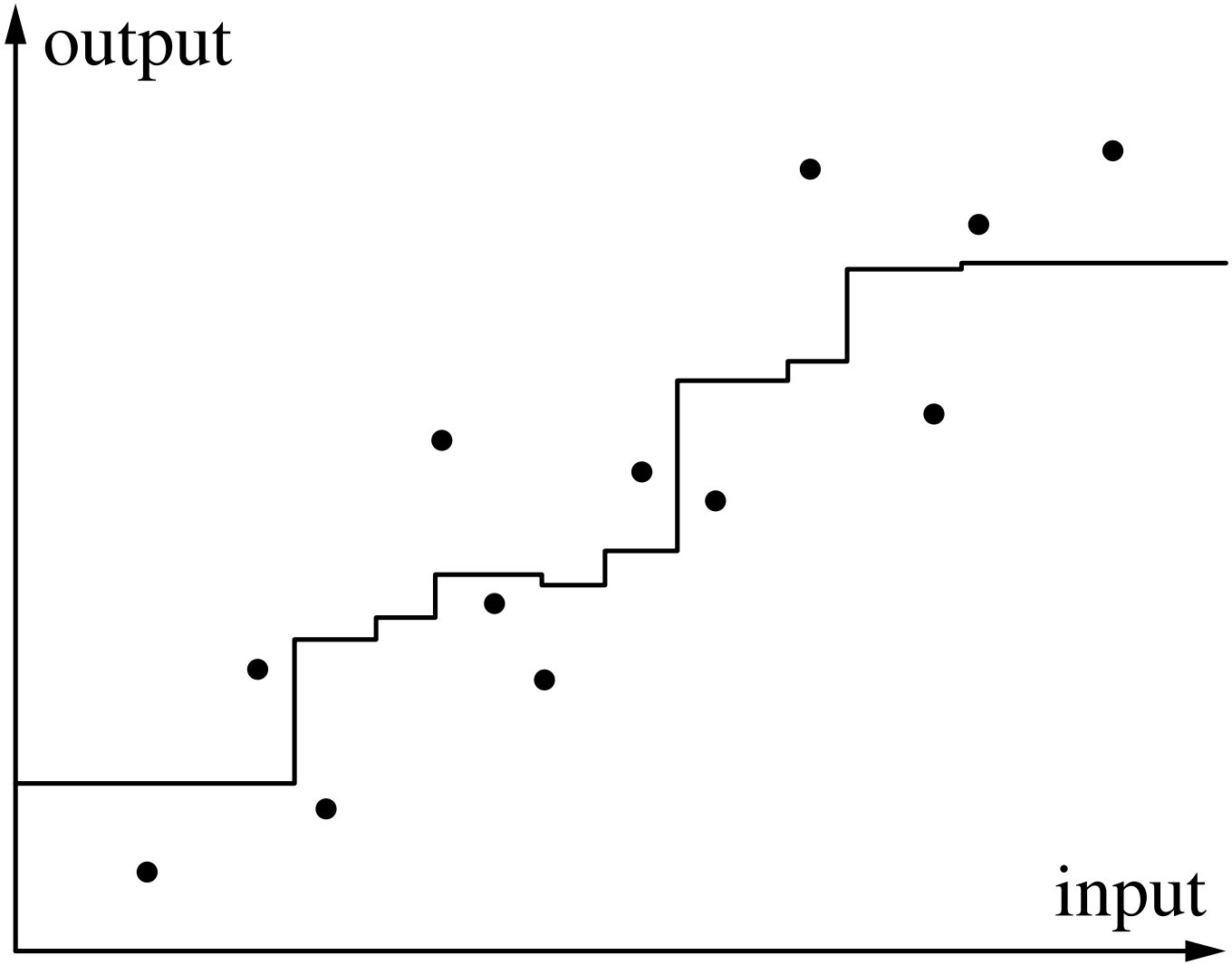
k-nearest neighbour predictor

- Instead of relying for the prediction on only one instance, the (single) nearest neighbour, usually the k ($k > 1$) are taken into account, leading to the k-nearest neighbour predictor.
- Classification: Choose the majority class among the k nearest neighbours for prediction.
- Regression: Take the mean value of the k nearest neighbours for prediction.

Disadvantage:

- All k nearest neighbours have the same influence on the prediction.
- Closer nearest neighbours should (perhaps) have a higher influence on the prediction.

Nearest neighbour predictor

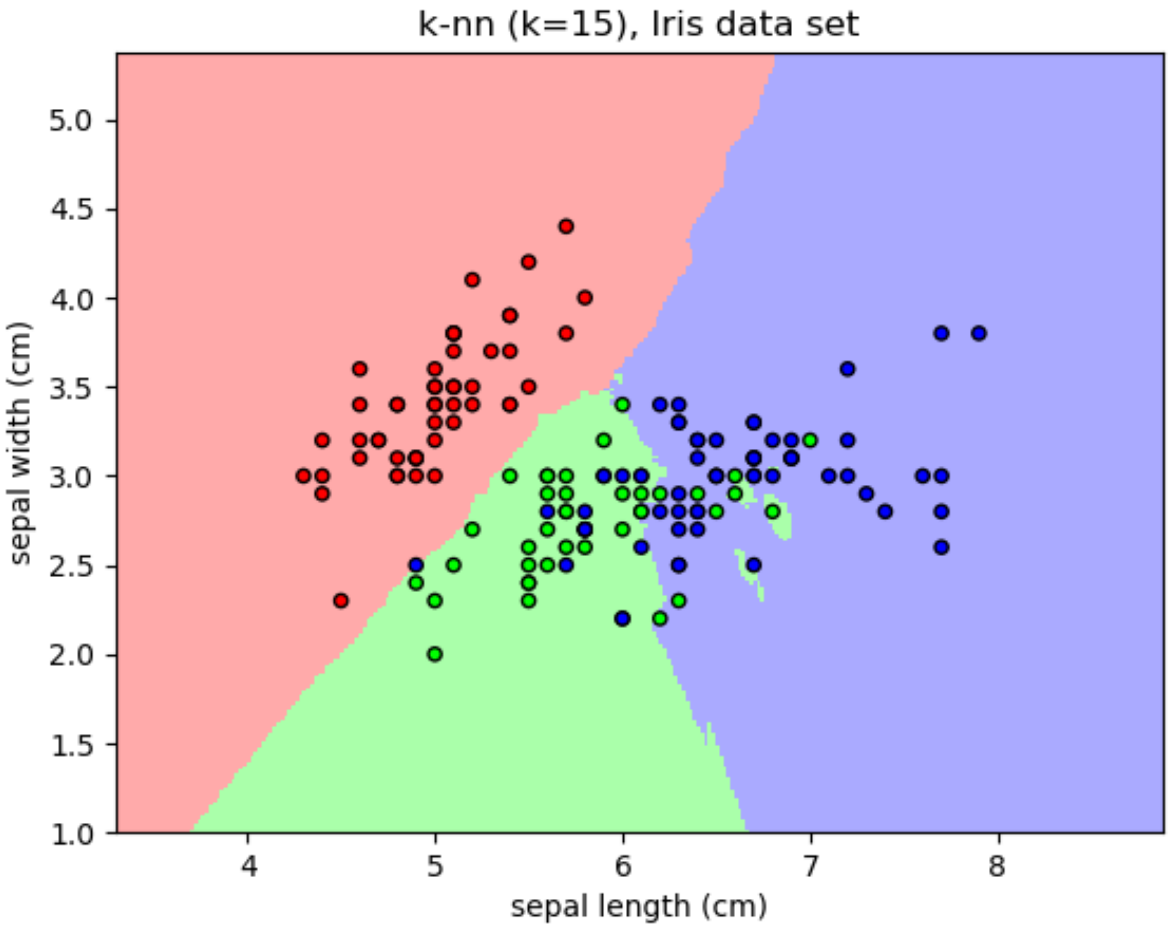


k nearest neighbor algorithm

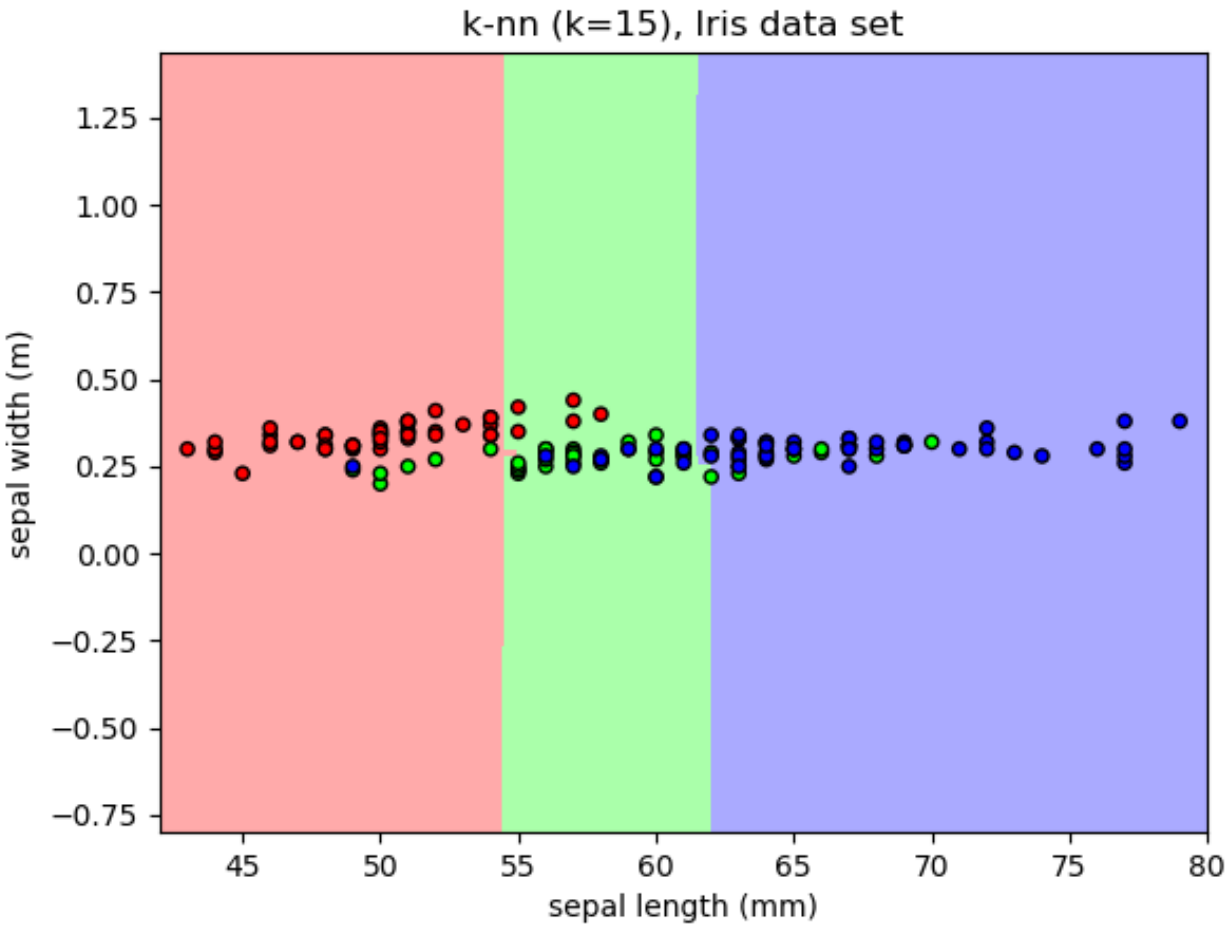
Algorithm 1 k nearest neighbors classifier

1: **procedure** K-NN(**z**, training set, k)
2: Compute distance $d(\mathbf{z}, \mathbf{x}_i)$ between **z** and all instances in training set
3: Let $(\mathbf{x}_{i_1}, y_{i_1}), ..., (\mathbf{x}_{i_k}, y_{i_k})$ be the k nearest training instances
4: Return the most common class among these k neighbours
5: **end procedure**

k-NN is sensitive to scaling of features



k-NN is sensitive to scaling of features



Ingredients for the k-nearest neighbour predictor

- **distance metric**
  The distance metric, together with a possible task-specific scaling or weighting of the attributes, determines which of the training examples are nearest to a query data point and thus selects the training example(s) used to compute a prediction.
- **number of neighbours**
  The number of neighbours of the query point that are considered can range from only one (the basic nearest neighbour approach) through a few (like k-nearest neighbour approaches) to, in principle, all data points as an extreme case.

Ingredients for the k-nearest neighbour predictor

- **weighting function for the neighbours**
  Weighting function defined on the distance of a neighbour from the query point, which yields higher values for smaller distances.
- **prediction function**
  For multiple neighbours, one needs a procedure to compute the prediction from the (generally differing) classes or target values of these neighbours, since they may differ and thus may not yield a unique prediction directly.

Distance

Choosing the "ingredients"

- **distance metric**
  Problem dependent. Often Euclidean distance (after normalisation).
- **number of neighbours**
  Very often chosen on the basis of cross-validation (cross-validation will be introduced later).
- **weighting function for the neighbours**
  Simplest choice uniform weighting. May also use (a function of) inverse of the distance for weighting.
- **prediction function**

Nearest neighbour predictor

Choosing the "ingredients"

- **prediction function**

  Regression. Compute the weighted average of the target values of the nearest neighbours.

  Classification. Sum up the weights for each class among the nearest neighbours. Choose the class with the highest value. If tie among several classes, choose one for example randomly.

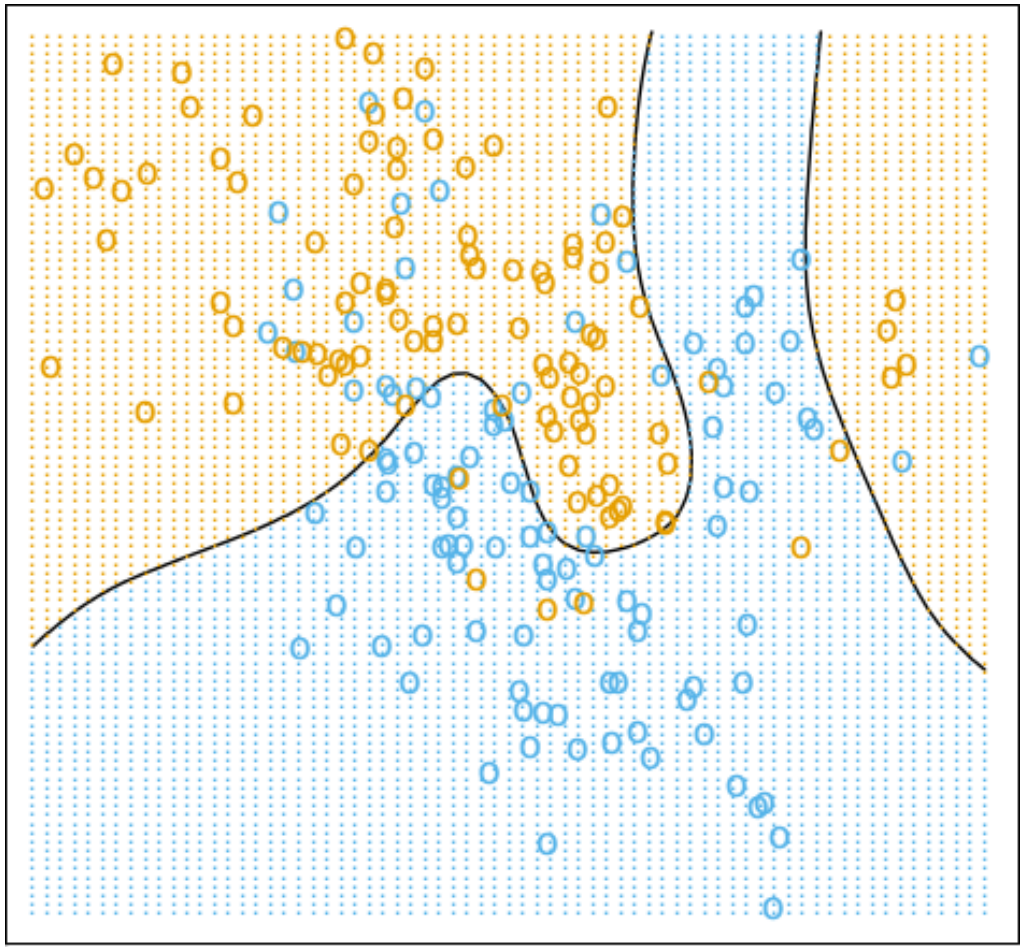
Adjusting the distance function

- The choice of the distance function is crucial for the success of a nearest neighbour approach.
- In addition to standard Euclidean distance a large number of other possible distance measures (Manhattan, Mahalanobis, Chebyshev, Hamming...)
- If your data is not numerical, also distances defined between strings, graphs, sets exist
- One can also try do adapt the distance function.
- One way to adapt the distance function is feature weights to put a stronger emphasis on those features that are considered more important, either based on prior knowledge or experiments.

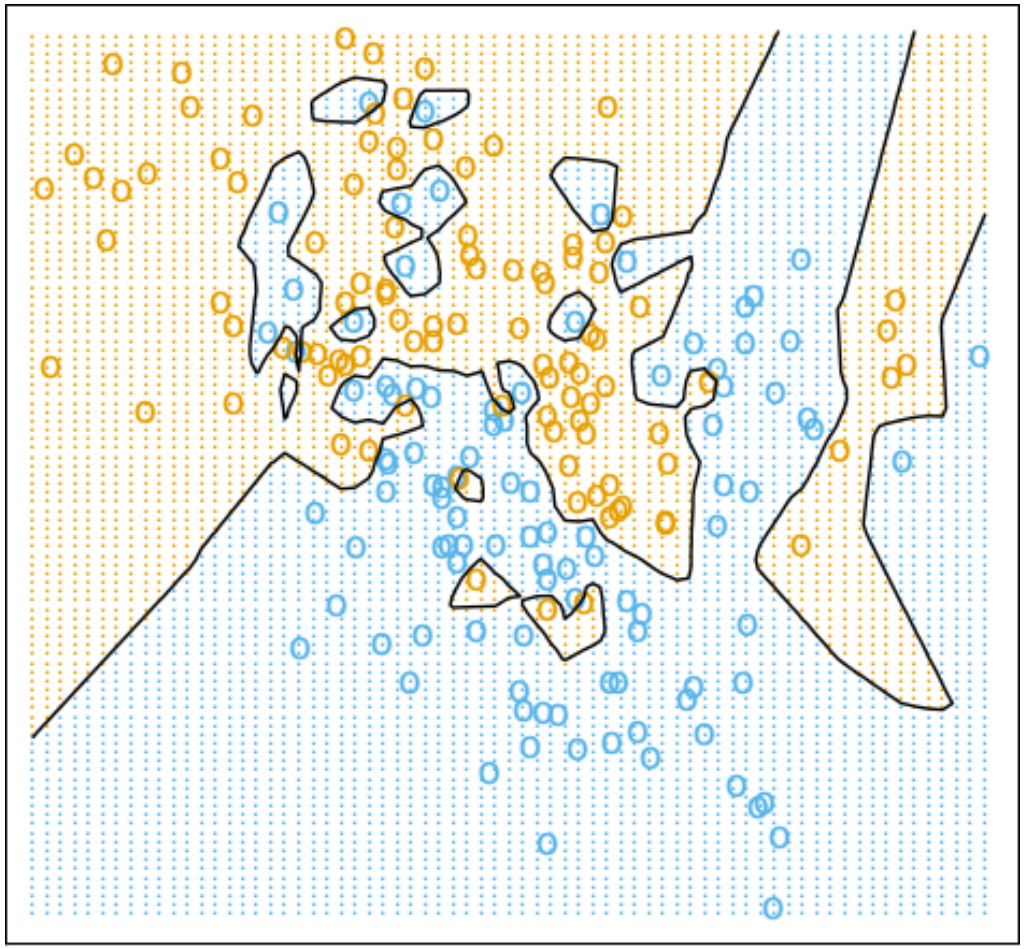
Model selection

- Parameter k (number of neighbors)
- Controls model complexity
- Small values: local complex model, depends only on a handful of instances
- Large values: global simpler model, averaged over a large set of instances
- Balancing act
  - overfitting: too complex model may fit to noise in the data
  - underfitting: too simple model cannot properly describe the real target function

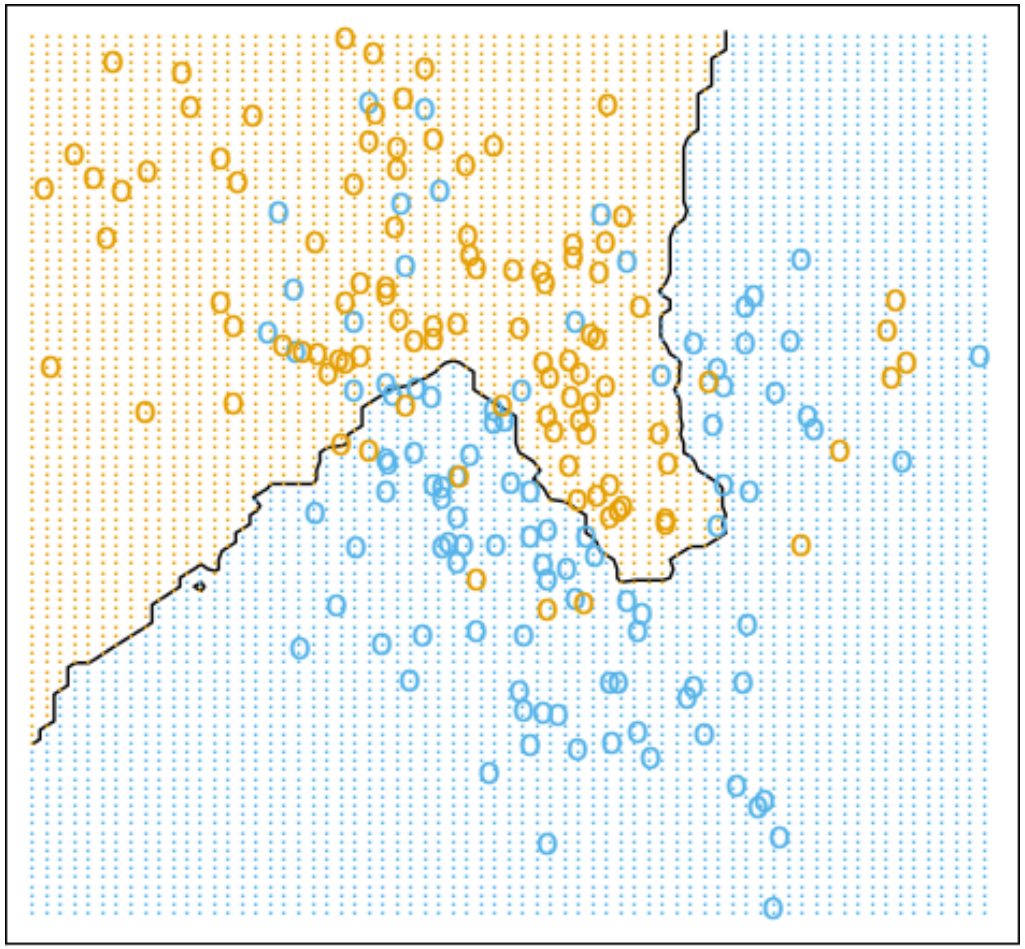
Ideal classifier



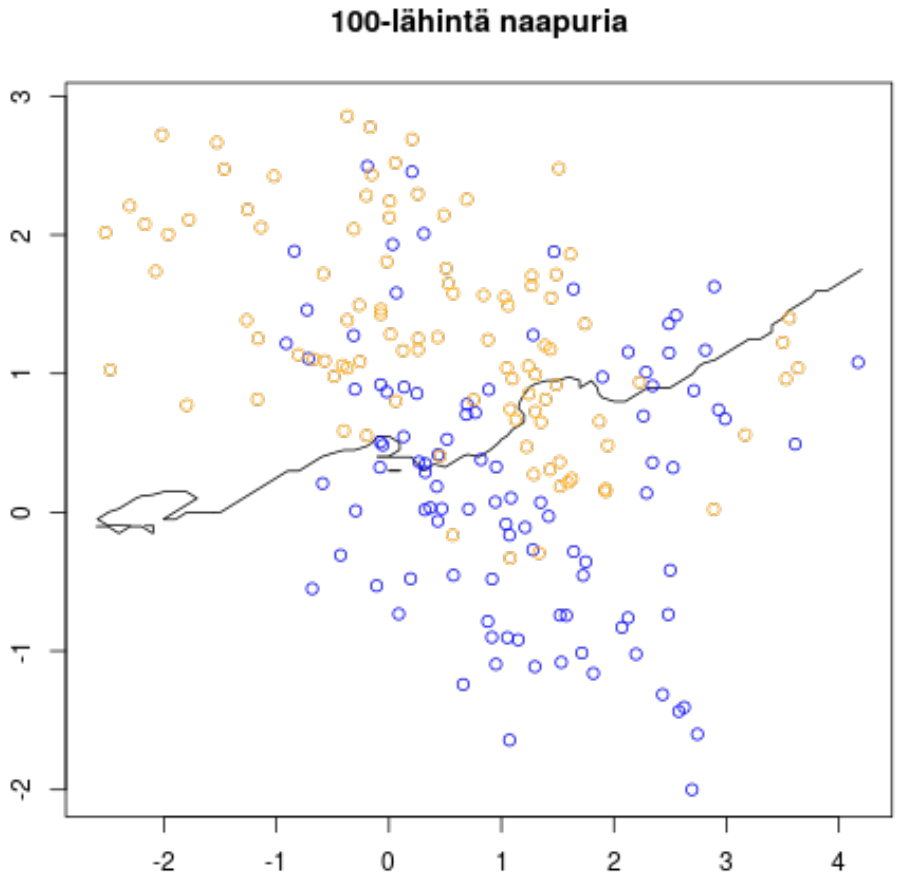
Overfitting (1-nearest neighbour)



Suitable complexity (15-nearest neighbour)



Underfitting (100-nearest neighbour)



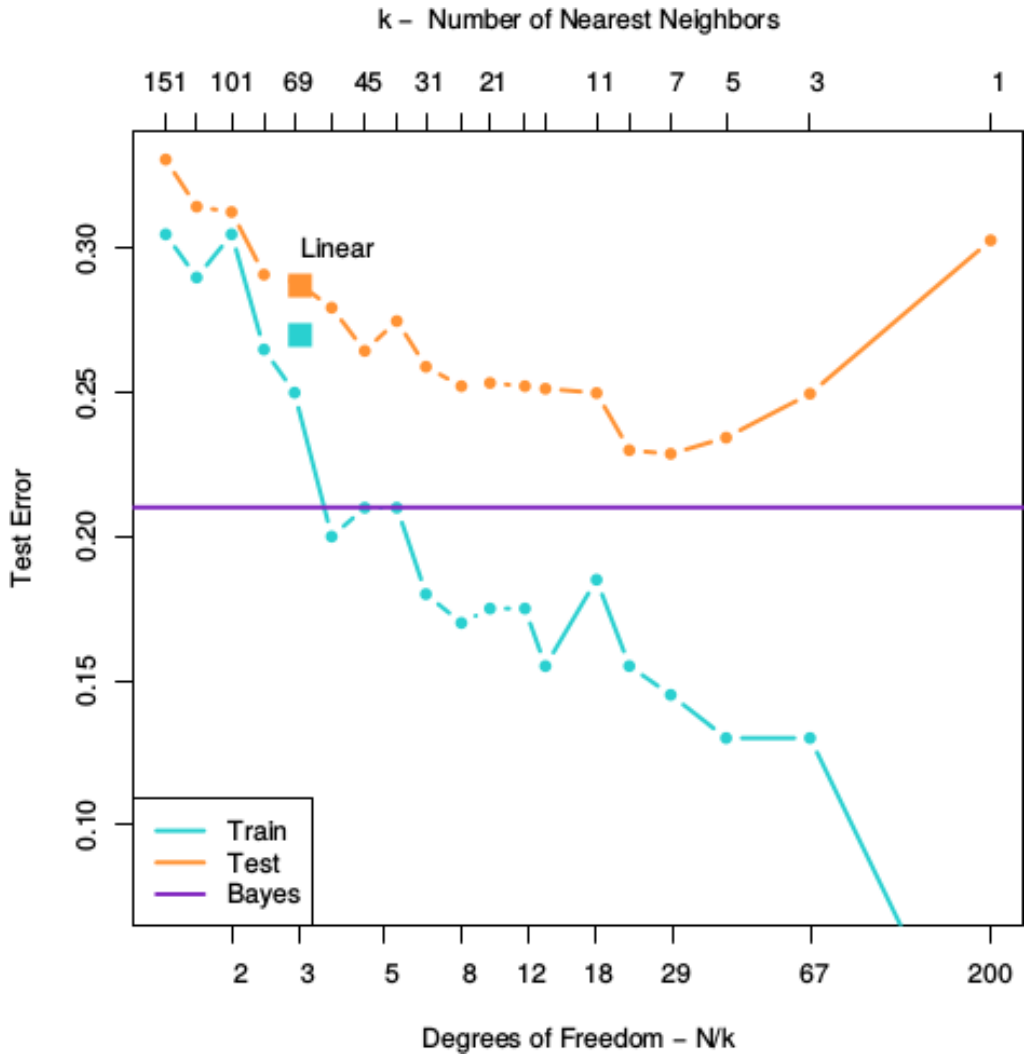
Model selection

- How to choose k?
- Optimal value depends on the learning problem
- Can we choose the value leading to smallest amount of errors on training data?

Model selection

- No! We would always choose k=1 (overfitting)
- need separate data set for choosing k, in order to check how well the model generalizes to new data

k-nearest neighbour: choosing k



Advantages of k-NN

- Extremely simple
- Non-linear modeling
- Simple mechanism for tuning model complexity (parameter k)
- Can be customized by using different distance measures, feature or neighbor weighting etc.
- Gives good results in many applications

Weaknesses of k-NN

- Large computational and memory complexity
  - For low-dimensional data neighbor search can be speeded up with suitable data structures
- Sensitive to feature scaling
  - Normalization
- Irrelevant features problematic, all features contribute equally to distance
  - Feature selection or dimensionality reduction
- Black box: does not produce explicit model that could be analyzed
- Not state-of-the-art method, sometimes can do better with more advanced ones

Applying the k-nearest neighbor method

- Few lines of code to implement, if you use standard choices like Euclidean distance, uniform weighting
- Implement function predict(x, k), that finds k-nearest neighbors of x, then returns the majority class (or mean value for regression) among them
- More complicated to implement, if you want to optimize the search, allow different distance measures etc.
- sklearn: see `http://scikit-learn.org/stable/modules/neighbors.html`

Conclusions

- Two criteria need to be balanced in learning, fit to data (low error) and model complexity
- Underfitting: too simple models may not be able to capture the underlying concept well
- Overfitting: too complex models may simply model the noise in the data
- The more data you have, the more complex models you can afford to use
- Many theoretical approaches to defining what we mean by complex: regularization theory, minimum description length principle, Bayesian priors...

Conclusions

- K-nearest neighbors, your first classification/regression method
- Simple yet powerful method, freely available implementations
- Consider data normalization, feature selection, dimensionality reduction as pre-processing
- Open questions:
  - How to choose parameters such as the value of k, or the distance measure?
  - How to reliably evaluate, how well the model predicts?
  - Next time: answer to these questions (and not only for k-nn)