Nearest Neighbor Methods

Lecture series „Machine Learning“

Niels Landwehr

Research Group „Data Science“
Institute of Computer Science
University of Hildesheim

Agenda for Lecture

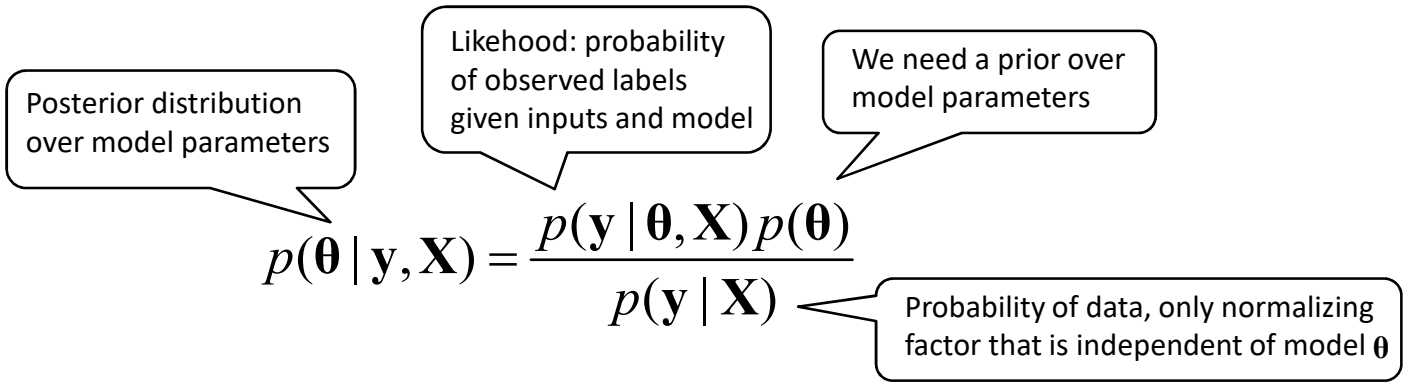
- Overflow from last lecture: Bayesian linear regression
- Distance measures
- K-Nearest Neighbor

Agenda for Lecture

- Overflow from last lecture: Bayesian linear regression
- Distance measures
- K-Nearest Neighbor

Bayes Rule for Probabilistic Linear Regression

- We use Bayes rule to compute the posterior probability:



- We need to
  - compute likelihood
  - define prior distribution
  - derive posterior distribution

Likelihood for Probabilistic Linear Regression

- Likelihood for probabilistic regression model:

$$p(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\theta}) = p(y_1, ..., y_N \mid \mathbf{x}_1, ..., \mathbf{x}_N, \boldsymbol{\theta})$$

$$= \prod_{n=1}^{N} p(y_n \mid \mathbf{x}_n, \boldsymbol{\theta})$$

Independent training instances

$$= \prod_{n=1}^{N} \mathcal{N}(y_n \mid \mathbf{x}_n^{\mathrm{T}} \boldsymbol{\theta}, \sigma^2)$$

Plugging in model

$$= \mathcal{N}(\mathbf{y} \mid \mathbf{X}\boldsymbol{\theta}, \sigma^2 \mathbf{I})$$

Product of univariate normal distributions can be written as multivariate normal distribution:

- Mean vector is vector of predictions:

$$\mathbf{X}\boldsymbol{\theta} = \begin{pmatrix} \mathbf{x}_1{}^T \boldsymbol{\theta} \\ ... \\ \mathbf{x}_n{}^T \boldsymbol{\theta} \end{pmatrix}$$

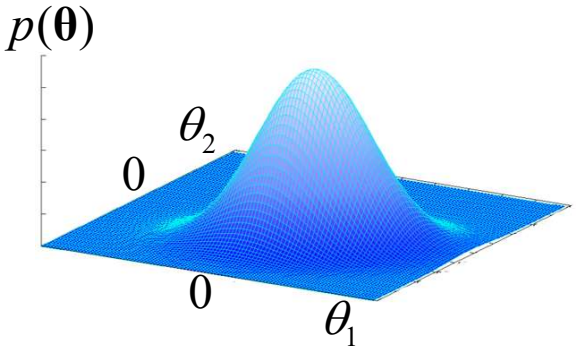
- Covariance matrix is a diagonal matrix $\sigma^2 \mathbf{I}$

Prior for Probabilistic Linear Regression

- Prior for probabilistic regression model:
  - We need to define a prior over parameter vectors $\boldsymbol{\theta} \in \mathbb{R}^M$
  - Similar idea as for regularization/shrinkage: we prefer parameter values that are not too large (in absolute value)

- As prior distribution, choose a multivariate normal distribution with mean zero and a diagonal covariance matrix:

$$p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta} \mid \mathbf{0}, \sigma_p^2 \mathbf{I})$$



- Hyperparameter σ_p^2 controls variance of prior distribution and thereby strength of preference for small parameters
  - small σ_p^2 : low variance, very „peaked“ distribution, strong regularization
  - large σ_p^2 : high variance, flat distribution, less regularization

Posterior for Probabilistic Linear Regression

- The prior distribution is a conjugate prior for the likelihood computed above: the product of prior and likelihood is again a multivariate normal distribution

$$p(\boldsymbol{\theta} \mid \mathbf{y}, \mathbf{X}) = \frac{1}{Z} p(\mathbf{y} \mid \boldsymbol{\theta}, \mathbf{X}) p(\boldsymbol{\theta}) \qquad \text{Bayes rule}$$

$$= \frac{1}{Z} \mathcal{N}(\mathbf{y} \mid \mathbf{X}\boldsymbol{\theta}, \sigma^2 \mathbf{I}) \cdot \mathcal{N}(\boldsymbol{\theta} \mid \mathbf{0}, \sigma_p^2 \mathbf{I}) \qquad \text{plugging in likelihood and prior}$$

$$= \mathcal{N}(\boldsymbol{\theta} \mid \overline{\boldsymbol{\theta}}, \mathbf{A}^{-1}) \qquad \text{compute product of normal terms - no details here}$$

$$\text{where } \mathbf{A} = \sigma^{-2} \mathbf{X}^{\mathrm{T}} \mathbf{X} + \sigma_p^{-2} \mathbf{I} \quad \text{and} \quad \overline{\boldsymbol{\theta}} = \sigma^{-2} \mathbf{A}^{-1} \mathbf{X}^{\mathrm{T}} \mathbf{y}$$

- Posterior is again normally distributed, with a new mean vector given by $\overline{\boldsymbol{\theta}}$ and a new covariance matrix $\mathbf{A}^{-1}$

Visualization: Posterior as Sequential Update

- The posterior distribution is given by

$$p(\boldsymbol{\theta} \mid \mathbf{y}, \mathbf{X}) \propto p(\boldsymbol{\theta}) p(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\theta})$$

$$= p(\boldsymbol{\theta}) \prod_{n=1}^{N} p(y_n \mid \mathbf{x}_n, \boldsymbol{\theta})$$

Symbol „$\propto$„: proportional to, means equal up to a normalizing constant that is independent of $\boldsymbol{\theta}$. Could also equivalently write

$$p(\boldsymbol{\theta} \mid \mathbf{y}, \mathbf{X}) = \frac{1}{Z} p(\boldsymbol{\theta}) p(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\theta})$$

- We can visualize the computation of the posterior as a sequential update: take the prior distribution, and successively multiply it with the likelihood terms of the individual training instances, yielding distributions $p_1(\boldsymbol{\theta}), \ldots, p_N(\boldsymbol{\theta})$

$$p(\boldsymbol{\theta} \mid \mathbf{y}, \mathbf{X}) \propto \underbrace{\underbrace{\underbrace{\underbrace{p(\boldsymbol{\theta}) p(y_1 \mid \mathbf{x}_1, \boldsymbol{\theta})}_{p_1(\boldsymbol{\theta})} p(y_2 \mid \mathbf{x}_2, \boldsymbol{\theta})}_{p_2(\boldsymbol{\theta})} p(y_3 \mid \mathbf{x}_3, \boldsymbol{\theta})}_{p_3(\boldsymbol{\theta})} \cdot \ldots \cdot p(y_N \mid \mathbf{x}_N, \boldsymbol{\theta})}_{p_N(\boldsymbol{\theta})}$$

Example: Sequential Update of Posterior

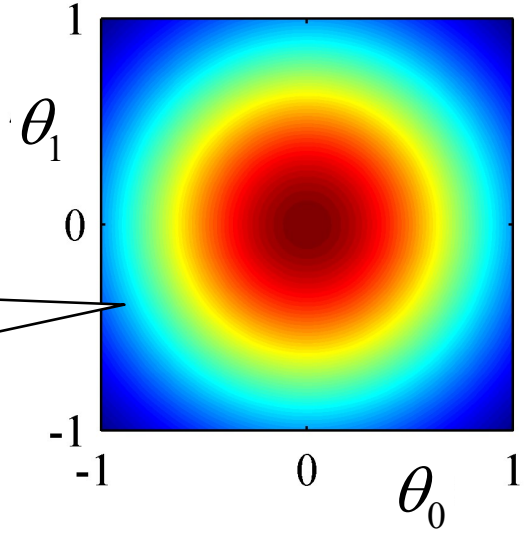
- Example: sequential update of the posterior for a Bayesian linear regression model
- One-dimensional linear regression model:

$$f_{\boldsymbol{\theta}}(x) = \theta_0 + \theta_1 x = \mathbf{x}^{\mathrm{T}}\boldsymbol{\theta} \qquad \boldsymbol{\theta} = (\theta_0, \theta_1)^{\mathrm{T}}$$

- Start: prior distribution („ $p_0(\boldsymbol{\theta})$ “)

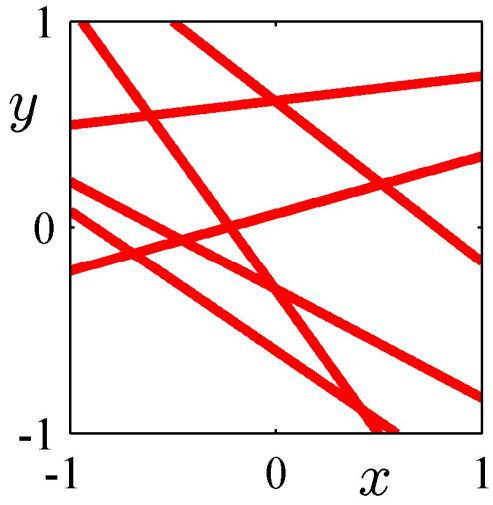
Prior is multivariate normal:
$p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta} \mid \mathbf{0}, \sigma_p^2 \mathbf{I})$

$p_0(\boldsymbol{\theta}) = p(\boldsymbol{\theta})$



Heat map visualization of distribution: red = high probability density function, blue = low pdf

Sample from $p_0(\boldsymbol{\theta})$



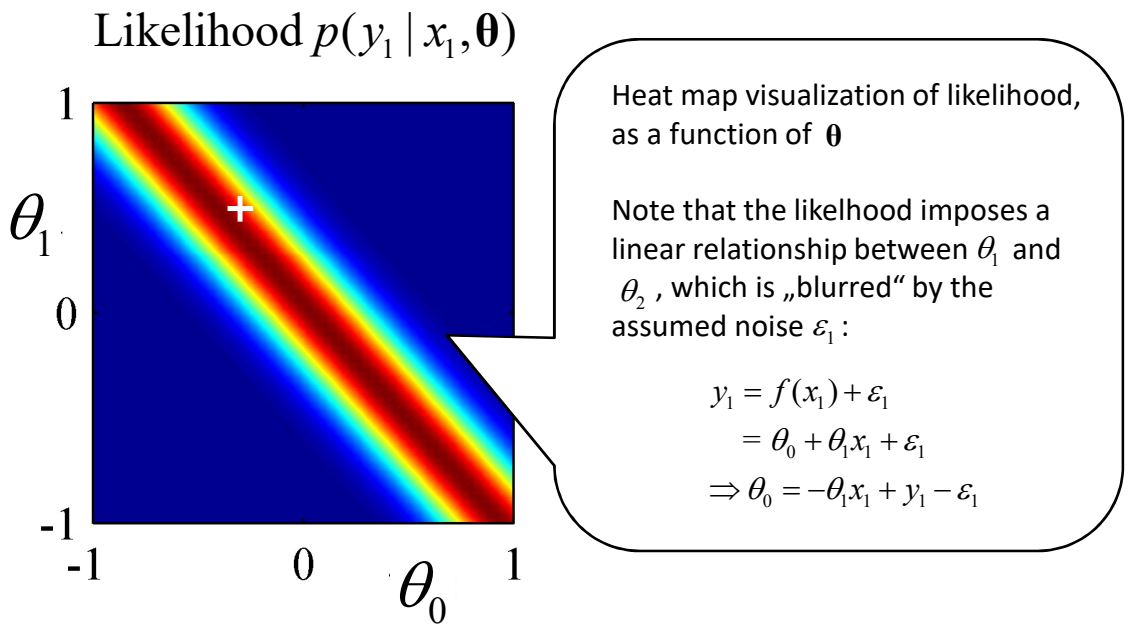
Figures: C. M. Bishop, "*Pattern recognition and Machine learning*". Springer (2006).

Example: Sequential Update of Posterior

- Example: sequential update of the posterior for a Bayesian linear regression model
- One-dimensional linear regression model:

$$f_{\boldsymbol{\theta}}(x) = \theta_0 + \theta_1 x = \mathbf{x}^{\mathrm{T}}\boldsymbol{\theta} \qquad \boldsymbol{\theta} = (\theta_0, \theta_1)^{\mathrm{T}}$$

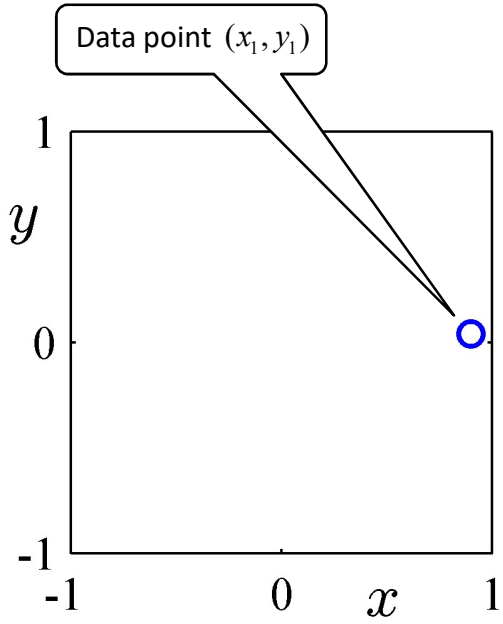
- Likelihood of first training instance (x_1, y_1)



Heat map visualization of likelihood, as a function of $\boldsymbol{\theta}$

Note that the likelhood imposes a linear relationship between θ_1 and θ_2, which is „blurred“ by the assumed noise ε_1:

$$\begin{aligned} y_1 &= f(x_1) + \varepsilon_1 \\ &= \theta_0 + \theta_1 x_1 + \varepsilon_1 \\ \Rightarrow \theta_0 &= -\theta_1 x_1 + y_1 - \varepsilon_1 \end{aligned}$$



Figures: C. M. Bishop, "*Pattern recognition and Machine learning*". Springer (2006).

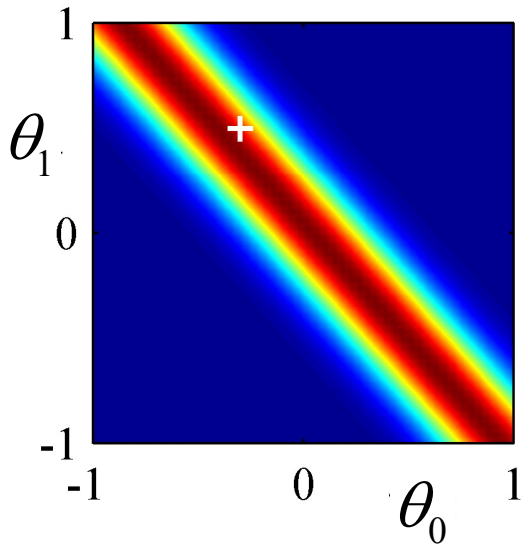
Example: Sequential Update of Posterior

- Example: sequential update of the posterior for a Bayesian linear regression model
- One-dimensional linear regression model:

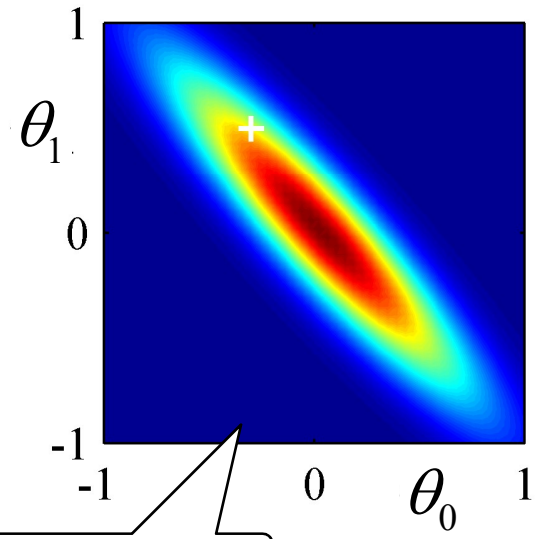
$$f_{\boldsymbol{\theta}}(x) = \theta_0 + \theta_1 x = \mathbf{x}^{\mathrm{T}}\boldsymbol{\theta} \qquad \boldsymbol{\theta} = (\theta_0, \theta_1)^{\mathrm{T}}$$

- Posterior after seeing first training instance (x_1, y_1)

Likelihood $p(y_1 \mid x_1, \boldsymbol{\theta})$

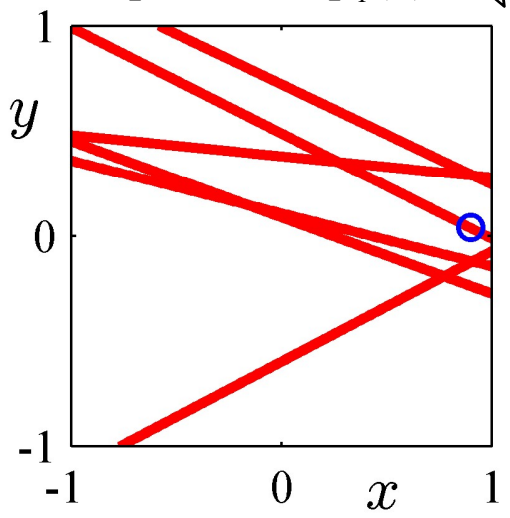


Posterior $p_1(\boldsymbol{\theta})$



$p_1(\boldsymbol{\theta}) \propto p_0(\boldsymbol{\theta}) p(y_1 \mid x_1, \boldsymbol{\theta})$

Samples from $p_1(\boldsymbol{\theta})$



Samples from posterior all pass (approximately) through the first training instance (x_1, y_1)

Figures: C. M. Bishop, "*Pattern recognition and Machine learning*". Springer (2006).

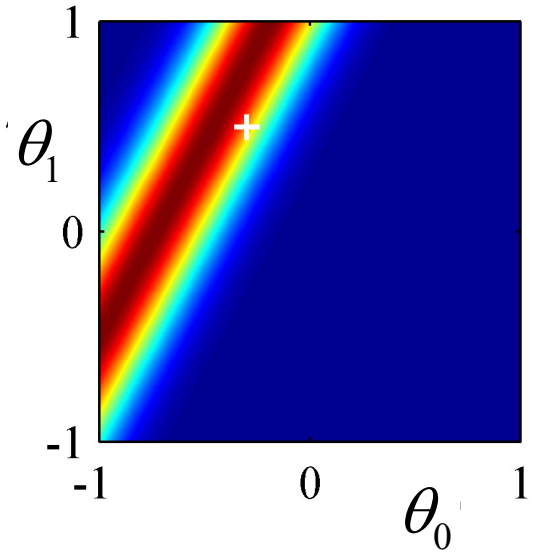
Example: Sequential Update of Posterior

- Example: sequential update of the posterior for a Bayesian linear regression model
- One-dimensional linear regression model:

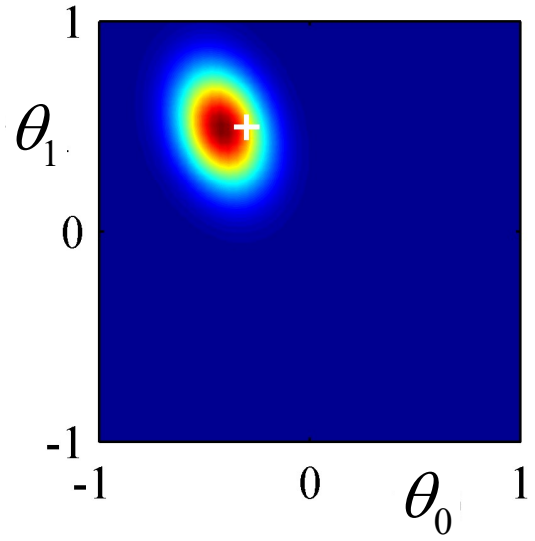
$$f_{\boldsymbol{\theta}}(x) = \theta_0 + \theta_1 x = \mathbf{x}^{\mathrm{T}}\boldsymbol{\theta} \qquad \boldsymbol{\theta} = (\theta_0, \theta_1)^{\mathrm{T}}$$

- Posterior after seeing second training instance (x_2, y_2)

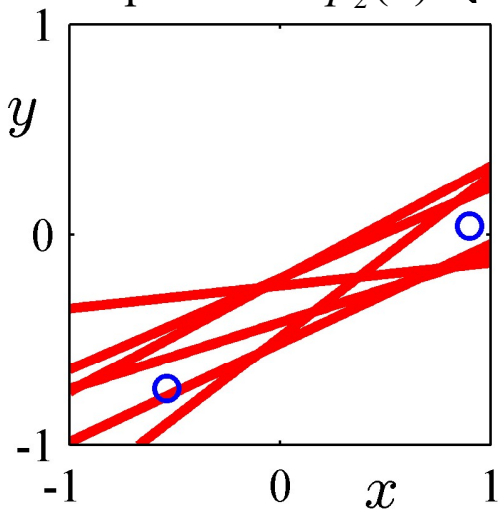
Likelihood $p(y_2 \mid x_2, \boldsymbol{\theta})$



Posterior $p_2(\boldsymbol{\theta})$



Samples from $p_2(\boldsymbol{\theta})$



Samples from posterior all pass (approximately) through both training instances

Figures: C. M. Bishop, "*Pattern recognition and Machine learning*". Springer (2006).

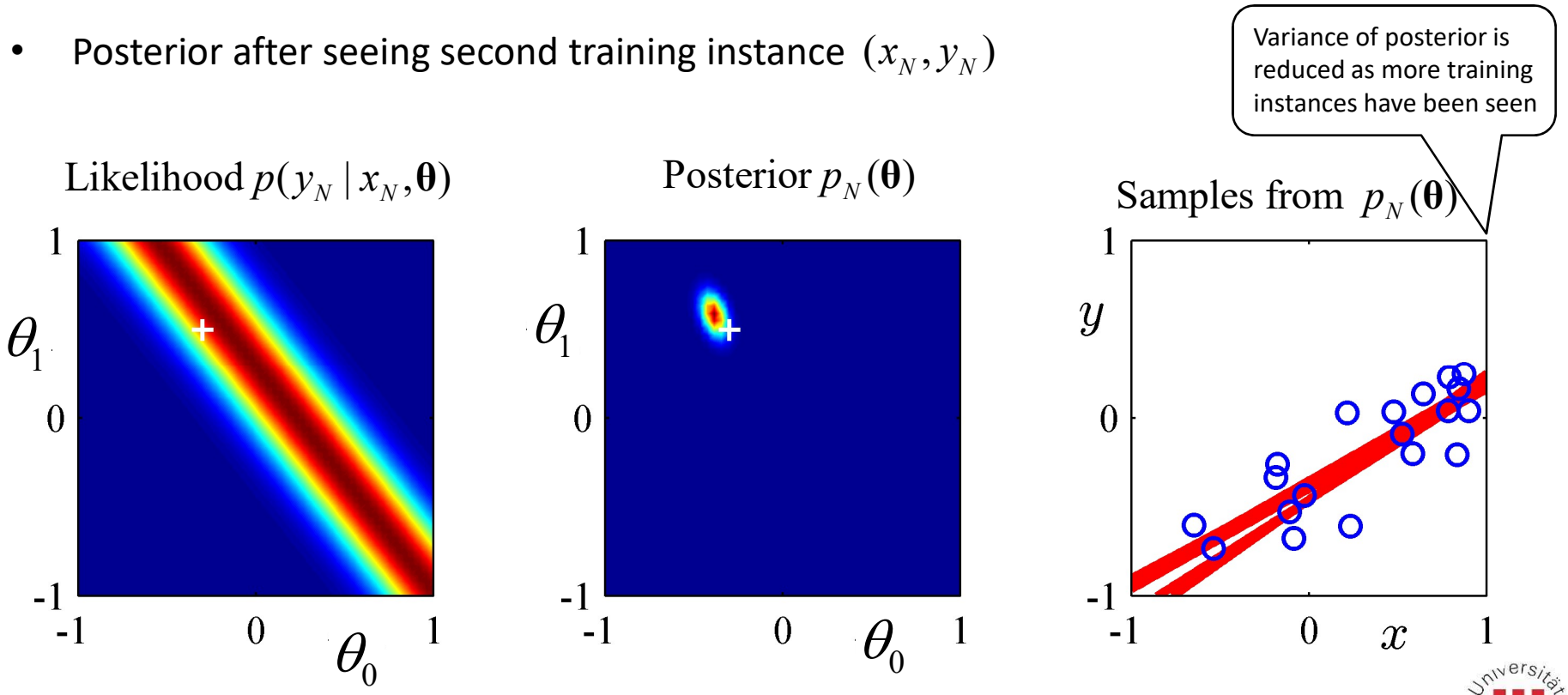
Example: Sequential Update of Posterior

- Example: sequential update of the posterior for a Bayesian linear regression model
- One-dimensional linear regression model:

$$f_{\boldsymbol{\theta}}(x) = \theta_0 + \theta_1 x = \mathbf{x}^{\mathrm{T}}\boldsymbol{\theta} \qquad \boldsymbol{\theta} = (\theta_0, \theta_1)^{\mathrm{T}}$$

- Posterior after seeing second training instance (x_N, y_N)

Variance of posterior is reduced as more training instances have been seen



Figures: C. M. Bishop, "*Pattern recognition and Machine learning*". Springer (2006).

Bayesian Linear Regression: Predictive Distribution

- Can also compute Bayesian prediction for probabilistic linear regression
- Recap: Bayesian prediction is

$$\hat{y} = \arg\max_y p(y \mid \mathbf{x}_{new}, \mathbf{X}, \mathbf{y})$$
$$= \arg\max_y \int p(y \mid \mathbf{x}_{new}, \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid \mathbf{X}, \mathbf{y}) \, \mathrm{d}\boldsymbol{\theta}$$

Average over all models $\boldsymbol{\theta}$

Prediction of model $\boldsymbol{\theta}$

- For probabilistic linear regression:

$$p(y \mid \mathbf{x}_{new}, \mathbf{X}, \mathbf{y}) = \int p(y \mid \mathbf{x}_{new}, \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid \mathbf{X}, \mathbf{y}) \, \mathrm{d}\boldsymbol{\theta}$$
$$= \int \mathcal{N}(y \mid \mathbf{x}_{new}^T \boldsymbol{\theta}, \sigma^2) \mathcal{N}(\boldsymbol{\theta} \mid \overline{\boldsymbol{\theta}}, A^{-1}) d\boldsymbol{\theta}$$
$$= \mathcal{N}\left(y \mid \mathbf{x}_{new}^T \overline{\boldsymbol{\theta}}, \sigma^2 + \mathbf{x}_{new}^T A^{-1} \mathbf{x}_{new}\right)$$

Plug in model definition and posterior from above

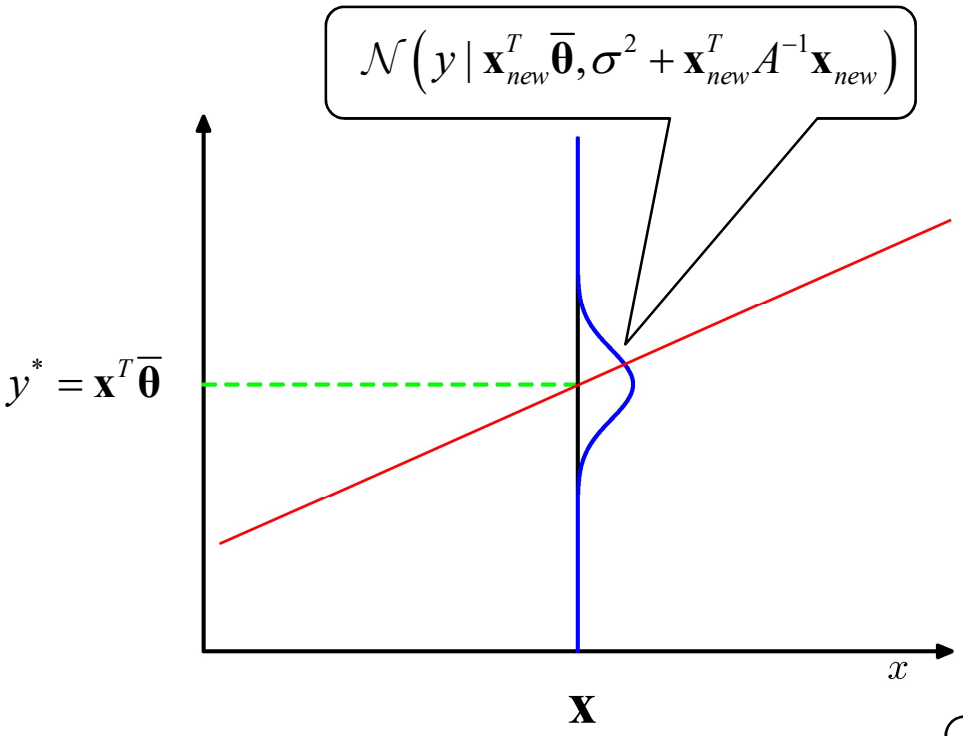
No proof here

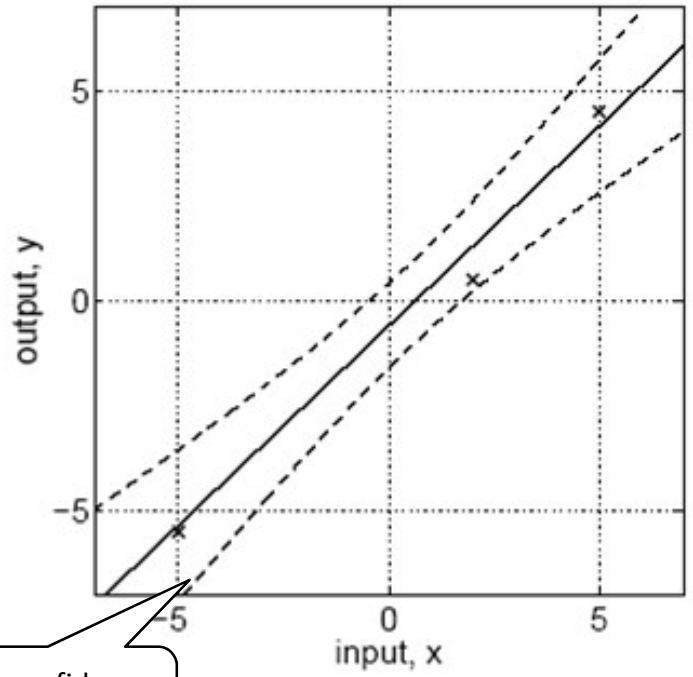
For Bayesian linear regression, the Bayesian prediction is the same as the prediction of MAP model $\overline{\boldsymbol{\theta}}$

where as before $\mathbf{A} = \sigma^{-2}\mathbf{X}^{\mathrm{T}}\mathbf{X} + \sigma_p^{-2}\mathbf{I}$ and $\overline{\boldsymbol{\theta}} = \sigma^{-2}\mathbf{A}^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$

Bayesian Linear Regression: Predictive Distribution

- For probabilistic linear regression model, Bayesian prediction is the same as prediction of MAP model
- However, the Bayesian prediction gives us an uncertainty estimate that takes into account the remaining uncertainty after having seen the data





E.g. 95% confidence for prediction

Bayesian Regression With Nonlinear Feature Map

- Of course, Bayesian linear regression can also directly be applied with non-linear (for example: polynomial) feature maps
- E.g. one-dimensional feature map:

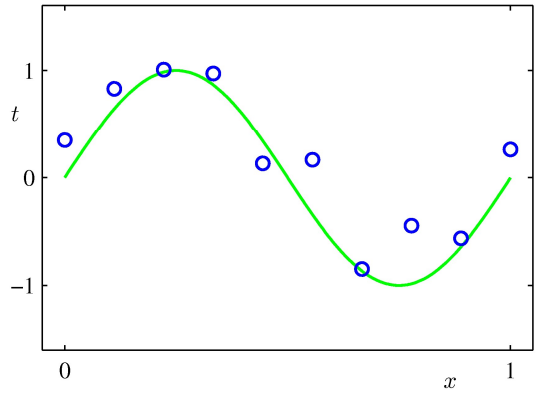
$$x \mapsto \underbrace{(1, x, x^2, ..., x^d)^{\mathrm{T}}}_{\mathbf{x}} \in \mathbb{R}^{d+1}$$

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = \theta_0 \cdot 1 + \theta_1 x + \theta_2 x^2 + ... + \theta_d x^d$$

- Running Bayesian linear regression on the transformed features $\mathbf{x}$ gives us a probabilistic nonlinear regression model
- Example: toy sine data set
  - generate data points by

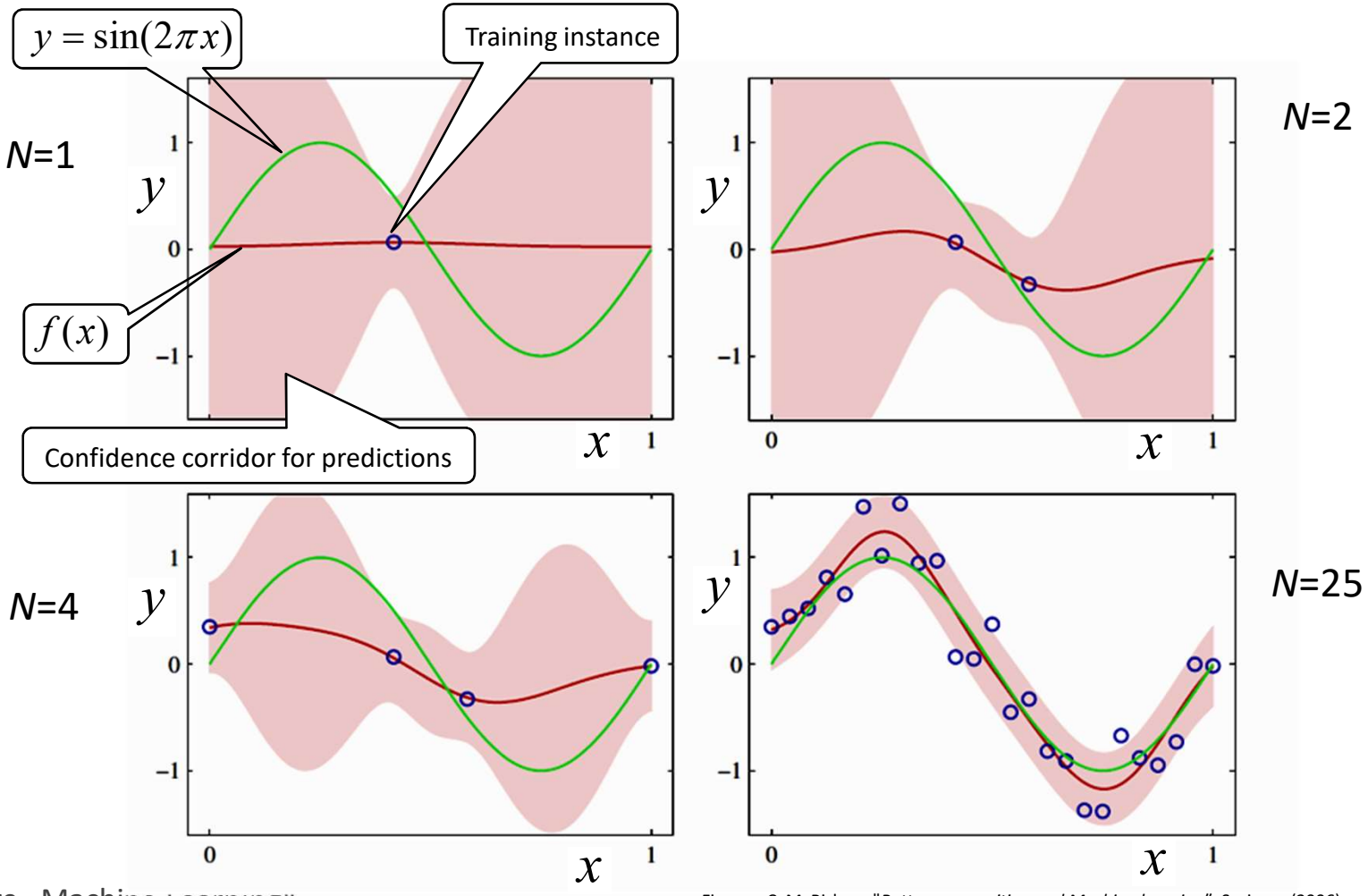
$$y = \sin(2\pi x) + \varepsilon$$
$$\varepsilon \sim \mathcal{N}(\varepsilon \mid 0, \sigma^2), \qquad x \in [0,1]$$

  - fit a linear regression on polynomial feature map



Example: Polynomial Bayesian Regression

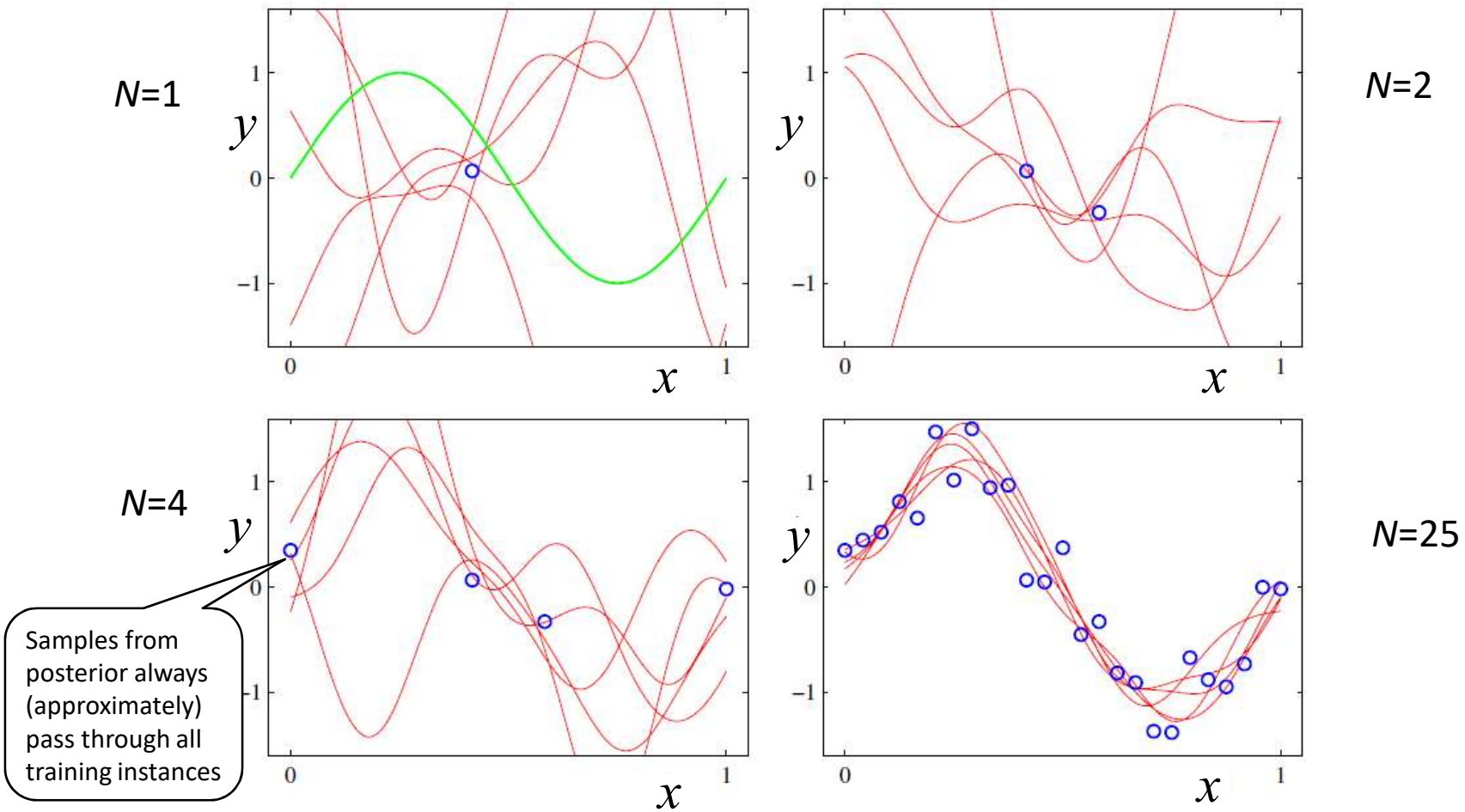
- Example: Polynomial regression on toy sine data set, prediction with confidence



Figures: C. M. Bishop, "*Pattern recognition and Machine learning*". Springer (2006).

Example: Polynomial Bayesian Regression

- Example: Polynomial regression on toy sine data set, samples from posterior



Figures: C. M. Bishop, "*Pattern recognition and Machine learning*". Springer (2006).

Summary: Bayesian Learning

- Bayesian learning takes a probabilistic approach and treats data and models as random variables
  - prior distribution over models
  - likelihood of data given models
  - posterior distribution over models
  - Bayesian prediction (average over models weighted by posterior)

- Conjugate priors are preferred for computational convenience: posterior distribution is in closed form and of the same family as prior distribution

- Bayesian linear regression is a simple and tractable Bayesian model: all distributions involved are normal distributions (prior, likelihood, posterior, Bayesian predictive distribution)

- Reference for further reading: C. M. Bishop, "*Pattern recognition and Machine learning",* Chapter 3.

Agenda for Lecture

- Overflow from last lecture: Bayesian linear regression
- Distance measures
- K-Nearest Neighbor

Slides in this section adapted from ISMLL group (Prof. Schmidt-Thieme)

Limits of Feature Representations

- A central part of the machine learning approaches we discussed so far were **feature representations**: we have always assumed an object can be described by a feature vector

$$\mathbf{x} = (x_1, ..., x_M)^{\mathrm{T}} \in \mathbb{R}^M$$

- Which kind of attributes for objects can be described by such a numerical vector?
  - Continuous attributes: straightforward, $x_i \in \mathbb{R}$
  - Binary attributes: also easily encoded as $x_i \in \{0,1\} \subset \mathbb{R}$
  - Categorical (also called nominal or multi-valued) attributes, e.g. color $x_i \in \{red, green, blue\}$: use a so-called **one-hot encoding**:

Assume a categorical attribute x_i with $x_i \in \{c_1, ..., c_T\}$

Transform $x_i \mapsto \overline{\mathbf{x}}_i$ with $\overline{\mathbf{x}}_i = \begin{pmatrix} I(x_i = c_1) \\ I(x_i = c_2) \\ ... \\ I(x_i = c_T) \end{pmatrix} \in \mathbb{R}^T$ $\qquad I(x = c_t) = \begin{cases} 1 : x_i = c_t \\ 0 : x_i \neq c_t \end{cases}$ (binary indicator)

The whole vector $\overline{\mathbf{x}}_i$ is included in the feature representation $\mathbf{x}$. That is, the original feature x_i transforms into T new features in $\mathbf{x}$

Structured Variables

- Sometimes we are also interested in more complex types of attributes
  - set-valued attributes,
  - sequence-valued attributes (e.g. strings)
  - trees, graphs, ...

- Two possible approaches for dealing with such complex variables
  1. **Feature extraction**:
     1. Manually (or potentially automatically) derive informative numerical, binary, or categorical attributes that can be encoded as a feature vector $\mathbf{x} \in \mathbb{R}^M$ as discussed above
     2. Run a machine learning algorithm using the resulting feature vector
  2. **Kernel or distance-based methods**:
     1. Manually (or potentially automatically) derive an informative distance metric that can be used to measure the distance between two instances
     2. Then use machine learning methods that only need distances between objects, without an explicit feature representation $\mathbf{x} \in \mathbb{R}^M$

Distance Measure and Metric

- A **distance measure** on a space $\mathcal{X}$ is a function

$$d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$

that quantifies the distance $d(\mathbf{x}, \mathbf{x}')$ between two instances $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$

- The distance measure is called a **metric** if the following three conditions are satisfied:
  - d is **positive definite**: $d(\mathbf{x}, \mathbf{x}') \geq 0$ and $d(\mathbf{x}, \mathbf{x}') = 0 \Leftrightarrow \mathbf{x} = \mathbf{x}'$
  - d is **symmetric**: $d(\mathbf{x}, \mathbf{x}') = d(\mathbf{x}', \mathbf{x})$
  - d is **subadditive** (or fulfils the **triangle inequality**): $d(\mathbf{x}, \mathbf{x}'') \leq d(\mathbf{x}, \mathbf{x}') + d(\mathbf{x}', \mathbf{x}'')$

  for all $\mathbf{x}, \mathbf{x}', \mathbf{x}'' \in \mathcal{X}$

- Example: Euclidian metric on $\mathcal{X} = \mathbb{R}^M$:

$$d(\mathbf{x}, \mathbf{x}') = \left\| \mathbf{x} - \mathbf{x}' \right\|_2 = \sqrt{\sum_{m=1}^{M} (x_m - x'_m)^2}$$

$$\mathbf{x} = (x_1, \ldots, x_M)^{\mathrm{T}} \qquad \mathbf{x}' = (x'_1, \ldots, x'_M)^{\mathrm{T}}$$

Minkowski Metric

- A widely used family of metric for the space $\mathcal{X} = \mathbb{R}^M$ is the Minkowski metric or Minkowski distance, also called the L_p metric

$$d(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|_p = \left(\sum_{m=1}^{M} | x_m - x'_m |^p \right)^{\frac{1}{p}}$$

- Examples:
  - $p = 1$ (taxicab distance, Manhattan metric)

$$d(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|_1 = \sum_{m=1}^{M} | x_m - x'_m |$$

  - $p = 2$ (Euclidian distance)

$$d(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|_2 = \sqrt{\sum_{m=1}^{M} (x_m - x'_m)^2}$$

  - $p = \infty$ (maximum distance, Chebyshev distance)

$$d(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|_\infty = \max_{m \in \{1,\ldots,M\}} | x_m - x'_m |$$

Example: Minkowski Metric

- Example: Minkowski metric

$$\mathbf{x} = \begin{pmatrix} 1 \\ 3 \\ 4 \end{pmatrix} \qquad \mathbf{x}' = \begin{pmatrix} 2 \\ 4 \\ 1 \end{pmatrix}$$

- Let's compute the distance between $\mathbf{x}$ and $\mathbf{x}'$, using different p

$$\|\mathbf{x} - \mathbf{x}'\|_1 = |1-2| + |3-4| + |4-1| = 1+1+3 = 5$$

$$\|\mathbf{x} - \mathbf{x}'\|_2 = \sqrt{(1-2)^2 + (3-4)^2 + (4-1)^2} = \sqrt{1+1+9} = \sqrt{11} \approx 3.32$$

$$\|\mathbf{x} - \mathbf{x}'\|_\infty = \max\{|1-2|, |3-4|, |4-1|\} = 3$$

Different Metrics, Different Decisions

- Depending on the metric that we choose, a machine learning algorithm using that metric will get different results

- Consider the following three data points:

$$\mathbf{x}_1 = \begin{pmatrix} 0.1 \\ 2.8 \end{pmatrix} \qquad \mathbf{x}_2 = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \qquad \mathbf{x}_3 = \begin{pmatrix} 1.9 \\ 1.9 \end{pmatrix}$$

- Question: which of the three points is closest to the origin $\mathbf{x}_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$?

Metric	$d(\mathbf{x}_0, \mathbf{x}_1)$	$d(\mathbf{x}_0, \mathbf{x}_2)$	$d(\mathbf{x}_0, \mathbf{x}_3)$
$\|\ldots\|_1$	2.9	3	3.8
$\|\ldots\|_2$	≈ 2.801	≈ 2.236	≈ 2.687
$\|\ldots\|_\infty$	2.8	2	1.9

Similarity Measures

- Instead of a distance measure (metric) sometimes **similarity measures** are used,

$$s : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$

- For a similarity measures we typically assume that it is symmetric: $s(\mathbf{x}, \mathbf{x}') = s(\mathbf{x}', \mathbf{x})$

- An example for a similarity measure would be cosine similarity: For instances $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^M$, their cosine similarity is defined as

$$s(\mathbf{x}, \mathbf{x}') = \frac{\langle \mathbf{x}, \mathbf{x}' \rangle}{\|\mathbf{x}\|_2 \|\mathbf{x}'\|_2} \in [-1, 1]$$

Dot product

- The cosine similarity is the cosine of the angle between the vectors $\mathbf{x}, \mathbf{x}'$:

$$\cos(\text{angle}(\mathbf{x}, \mathbf{x}')) = \frac{\langle \mathbf{x}, \mathbf{x}' \rangle}{\|\mathbf{x}\|_2 \|\mathbf{x}'\|_2}$$

Distance for Set-valued Features

- Distances can also be defined for feature spaces $\mathcal{X}$ that are not simply vector spaces
- For example, for set-valued features, possible values of a feature are subsets of a set of elements $\mathcal{A}$, for example, $x_i = \{a,c,d,f\}$ for the set $\mathcal{A} = \{a,b,c,d,e,f,g\}$

- A possible distance measure for set-valued features is the **Hamming distance**: the number of elements contained in only one of the sets

$$d(x,x') = |\, x \setminus x' \;\cup\; x' \setminus x \,|$$

Distance for single feature, would have to combine with distance over other features in instance

$$d(\{a,c,d,f\},\{a,b,c,f,g\}) = |\, \{d\} \;\cup\; \{\mathrm{b,g}\} \,| \;= 3$$

- A possible similarity measure for set-valued features is the **Jaccard coefficient**: the ratio of common elements over the union of elements

$$s(x,x') = \frac{|\, x \cap x' \,|}{|\, x \cup x' \,|} \qquad s(\{a,c,d,f\},\{a,b,c,f,g\}) = \frac{|\, \{a,c,f\} \,|}{|\, \{a,b,c,d,f,g\} \,|} = \frac{3}{6}$$

Levenshtein Distance for Sequences

- A natural distance measure for sequences is the so-called **edit distance** or **Levenshtein distance**
- Let $x_1,...,x_L$ and $y_1,...,y_K$ denote sequences over a fixed alphabet $\mathcal{A}$, that is $x_i, y_j \in \mathcal{A}$
- The edit distance $d((x_1,...,x_L),(y_1,...,y_K))$ is the minimum number of insertions, deletions, or substitutions needed to transform $x_1,...,x_L$ into $y_1,...,y_K$

Examples:

d(room,door)=2	two substitutions: r $\rightarrow$ d, m $\rightarrow$ r
d(house,spouse)=2	h $\rightarrow$ p, insert s
d(man,women)=3	a $\rightarrow$ e, insert w, insert o

- Note that due to the symmetry of the allowed operations, Levenshtein distance is symmetric

Definition: Levenshtein Distance

- More formally, the Levenshtein distance can be defined as follows:

$$d((x_1,...,x_L),(y_1,...,y_K)) = \begin{cases} K & \text{if } L = 0 \\ L & \text{if } K = 0 \\ d((x_1,...,x_{L-1}),(y_1,...,y_{K-1})) & \text{if } x_L == y_K \\ 1 + \min \begin{cases} d((x_1,...,x_{L-1}),(y_1,...,y_K)) \\ d((x_1,...,x_L),(y_1,...,y_{K-1})) \\ d((x_1,...,x_{L-1}),(y_1,...,y_{K-1})) \end{cases} & \text{otherwise} \end{cases}$$

If $x_1,...,x_L$ is the empty string, we need *K* insertions

If $y_1,...,x_K$ is the empty string, we need *L* deletions

If the last character is the same,compute distance between rest

Costs are 1 for the deletion, insertion or substitution plus cost of remaining transform

If the last character is different, there are three options for transforming the first string into the second:

- delete last character and transform $x_1,...,x_{L-1}$ into $y_1,...,y_K$
- insert character y_K at the end and transform $x_1,...,x_L$ into $y_1,...,y_{K-1}$
- substitute character x_L with y_K and transform $x_1,...,x_{L-1}$ into $y_1,...,y_{K-1}$

Computing the Levenshtein Distance

- The Levenshtein distance can be computed by filling a $L \times K$ matrix D
- The entry in position i, j in the matrix holds $d(x_1,...,x_i, y_1,...,y_j)$: the Levenshtein distances between prefixes $x_1,...,x_i$ and $y_1,...,y_j$
- **Initialization**:
  - transforming an empty string into a string of length j has cost *j:*

$$D(0, j) = j \qquad \text{for } j \in \{0,...K\}$$

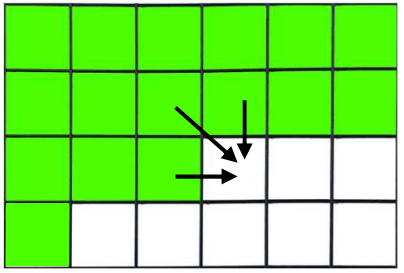
  - transforming a string of length *i* into an empty string has cost *i*

$$D(i, 0) = i \qquad \text{for } i \in \{0,...,L\}$$

- **Recursion**: for $i > 0, j > 0$ can compute

$$D(i,j) = \begin{cases} D(i-1, j-1): & \text{if } x_i = y_j \\ 1 + \min \begin{cases} D(i-1, j) \\ D(i, j-1) \\ D(i-1, j-1) \end{cases} & \text{otherwise} \end{cases}$$

„Flood filling“ the matrix: earlier values needed for recursion are already computed



- Final result $d(x_1,...,x_L, y_1,...,y_K)$ is in bottom-right cell

Example: Levenshtein Distance

- Example: compute the distance between strings „man“ and „women“

Initialization:

		w	o	m	e	n
	0	1	2	3	4	5
m	1					
a	2					
n	3					

Recursion:

		w	o	m	e	n
	0	1	2	3	4	5
m	1	1	2	2	3	4
a	2	2	2	3	3	4
n	3	3	3	3	4	3

Characters different: set current cell to minimum of cells above, left, and above-left plus 1

Characters the same: set current cell equal to cell above-left

$$d("man","women") = 3$$

- This dynamic programming algorithm is also called the Wagner-Fischer Algorithm

Agenda for Lecture

- Overflow from last lecture: Bayesian linear regression
- Distance measures
- K-Nearest Neighbor

Slides in this section adapted from ISMLL group (Prof. Schmidt-Thieme)

Neighborhoods Based On Distances

- For discussing *K*-nearest neighbor methods, we first need a definition of neighborhood

- Assume a supervised learning problem, and let

$$\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\} \qquad \mathbf{x}_n \in \mathcal{X},\ y_n \in \mathcal{Y}$$

  denote a labeled data set

- Let $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ denote a distance measure
- For an instance $\mathbf{x} \in \mathcal{X}$, let $\pi_{\mathbf{x}} : \{1, \ldots, N\} \to \{1, \ldots, N\}$ denote a permutation that sorts instances in $\mathcal{D}$ according to distance from $\mathbf{x}$ (ties broken arbitrarily). That is,

$$d(\mathbf{x}, \mathbf{x}_{\pi_{\mathbf{x}}(n)}) \leq d(\mathbf{x}, \mathbf{x}_{\pi_{\mathbf{x}}(n+1)}) \qquad 1 \leq n \leq N-1$$

- The ***K*-neighborhood** of $\mathbf{x}$ in $\mathcal{D}$ is the set of the *K* points in $\mathcal{D}$ closest to $\mathbf{x}$:

$$C_{K,\mathcal{D}}(\mathbf{x}) = \{(\mathbf{x}_{\pi_{\mathbf{x}}(1)}, y_{\pi_{\mathbf{x}}(1)}), \ldots, (\mathbf{x}_{\pi_{\mathbf{x}}(K)}, y_{\pi_{\mathbf{x}}(K)})\}$$

Neighborhoods Based on Similarities

- Equivalently, we can define a *K*-neighborhood based on a similarity function rather than a distance function as follows

- Let $s : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ denote a similarity measure
- For an instance $\mathbf{x} \in \mathcal{X}$, let $\pi_{\mathbf{x}} : \{1,...,N\} \to \{1,...,N\}$ denote a permutation that sorts instances in $\mathcal{D}$ according to similarity from $\mathbf{x}$ (ties broken arbitrarily). That is,

$$s(\mathbf{x}, \mathbf{x}_{\pi_{\mathbf{x}}(n)}) \geq s(\mathbf{x}, \mathbf{x}_{\pi_{\mathbf{x}}(n+1)}) \qquad 1 \leq n \leq N-1$$

- The ***K*-neighborhood** of $\mathbf{x}$ in $\mathcal{D}$ is the set of the *K* points in $\mathcal{D}$ most similar to $\mathbf{x}$:

$$C_{K,\mathcal{D}}(\mathbf{x}) = \{(\mathbf{x}_{\pi_{\mathbf{x}}(1)}, y_{\pi_{\mathbf{x}}(1)}),...,(\mathbf{x}_{\pi_{\mathbf{x}}(K)}, y_{\pi_{\mathbf{x}}(K)})\}$$

K-Nearest Neighbor Regressor

- Assume a regression problem, that is, the label space is $\mathcal{Y} = \mathbb{R}$
- The ***K*-nearest neighbor regressor** is a function $f_{\mathcal{D}} : \mathcal{X} \to \mathcal{Y}$ given by

$$f_{\mathcal{D}}(\mathbf{x}) = \frac{1}{K} \sum_{(\overline{\mathbf{x}}, \overline{y}) \in C_{K,\mathcal{D}}(\mathbf{x})} \overline{y}$$

Average over targets seen in *K*-neighborhood

- The parameter *K* (size of the neighborhood) is a hyperparameter of the algorithm
- Note that for this regression model, there is no training stage:
  - „Training“ consists of storing the training data $\mathcal{D}$, this is computationally inexpensive but can require a lot of memory
  - For prediction, we need to find the neighborhood $C_{K,\mathcal{D}}$, which can be computationally expensive (naive implementation: linear in size of training data)

K-Nearest Neighbor Classifier

- Assume a classification problem, that is, the label space is $\mathcal{Y} = \{0,1\}$ (binary classification) or $\mathcal{Y} = \{1,...,T\}$ (multiclass classification)
- The ***K*-nearest neighbor classifier** is a function $f_{\mathcal{D}} : \mathcal{X} \to \mathcal{Y}$ given by

$$p(y \mid \mathbf{x}, f_{\mathcal{D}}) = \frac{1}{K} \sum_{(\bar{\mathbf{x}},\bar{y}) \in C_{K,\mathcal{D}}(\mathbf{x})} I(y = \bar{y})$$

Estimate class probabilities for instance **x** as fractions of class labels observed in the K-neighborhood

$$f_{\mathcal{D}}(\mathbf{x}) = \arg\max_{y} p(y \mid \mathbf{x}, f_{\mathcal{D}})$$

Return the class with highest probability (ties broken arbitrarily)

- Again, *K* is a hyperparameter of the algorithm
- As for nearest neighbor regression, there is no explicit training stage, but finding the *K*-neighborhood for predictions can be computationally expensive

K-Nearest Neighbor Regression Algorithm

- The *K*-nearest neighbor regression algorithm can be given in pseudocode as follows:

Algorithm predict-knn-regression

Input : new instance $\mathbf{x} \in \mathcal{X}$, training data $\mathcal{D} = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$

Output : predicted target $\hat{y} \in \mathbb{R}$

1. Allocate array A of size N
2. For $n \in \{1, ..., N\}$:
3. $\quad A[n] := d(\mathbf{x}, \mathbf{x}_n)$
4. $C := \text{arg-min-K}(A)$
5. $\hat{y} := \frac{1}{K} \sum_{k=1}^{K} y_{C[k]}$

Can also use similarity instead of distance. In this case, arg-min-K(A) is replaced by arg-max-K(A)

Return indices of the *K* smallest elements in the array *A*

- The key step is the computation of the distances A and the *K*-neighborhood:
  - The function $\text{arg-min-K}(A)$ returns the indices of the *K* smallest elements in the array A, that is, the indices of the *K* closest training samples
  - A naive implementation would have to go through all training examples to find the *K* closest ones

K-Nearest Neighbor Classification Algorithm

- The *K*-nearest neighbor classification algorithm can be given in pseudocode as follows:

Algorithm predict-knn-classification

Input : new instance $\mathbf{x} \in \mathcal{X}$, training data $\mathcal{D} = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$

Output : predicted class probabilities for instance $\mathbf{x}$

1. Allocate array A of size N
2. For $n \in \{1, ..., N\}$:
3. $\quad A[n] := d(\mathbf{x}, \mathbf{x}_n)$
4. $C :=$ arg-min-K(A) ← Return indices of the *K* smallest elements in the array *A*
5. Allocate array P of size $|\mathcal{Y}|$ an initialize to zero
6. For $k \in \{1, ..., K\}$:
7. $\quad P[y_{C[k]}] = P[y_{C[k]}] + \frac{1}{K}$ ← Increment probability of the class corresponding to label of instance k in neighborhood
8. Return P

- Again, the key step is the computation of A and the *K*-neighborhood

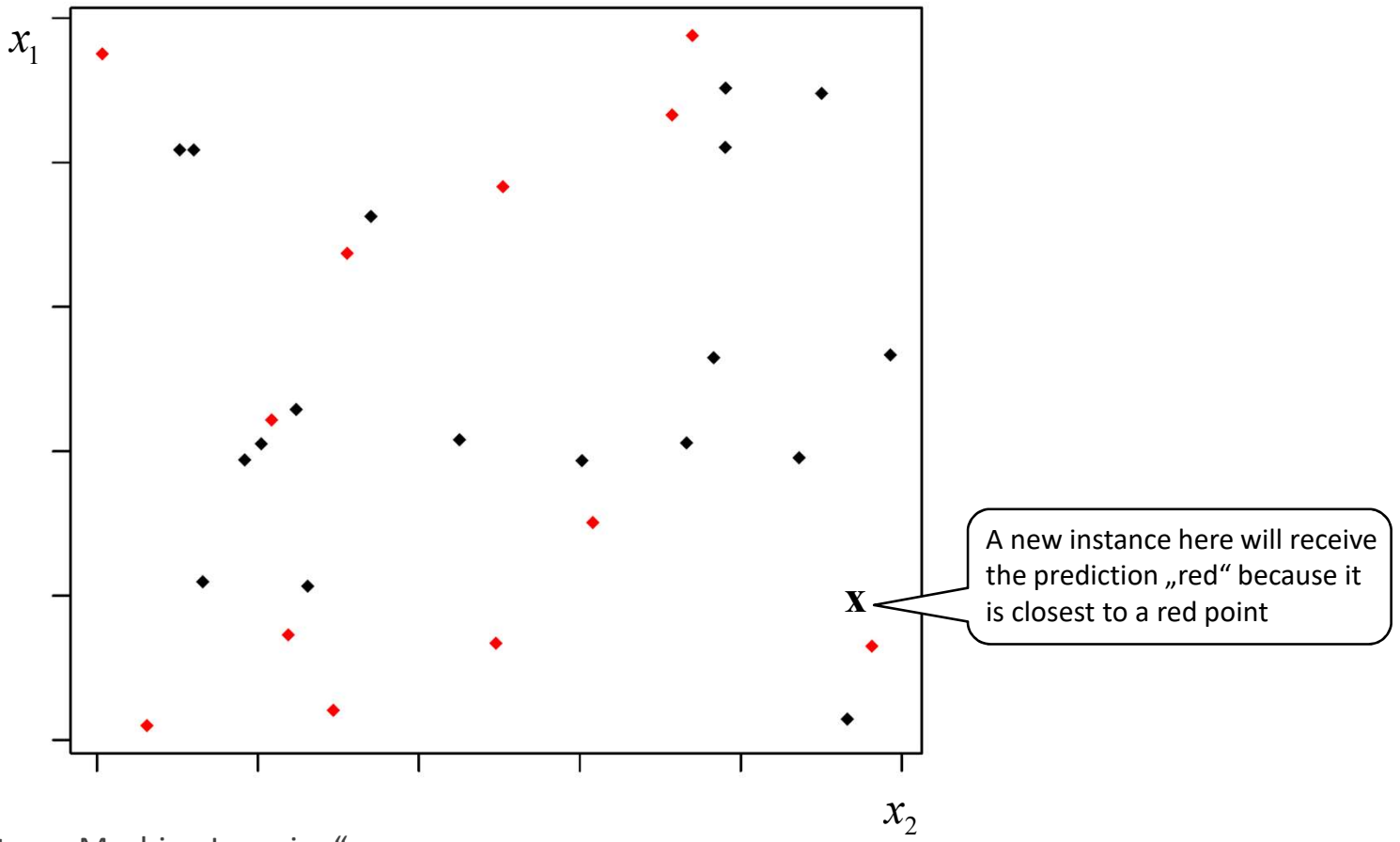
K-Nearest Neighbor Classification Algorithm

- For a naive implementation the computational complexity of K-Nearest Neighbor is relatively high. For example, the following naive implementation has complexity $O(MN + KN)$, assuming $\mathcal{X} = \mathbb{R}^M$ and that a distance computation can be done in time $O(M)$:
  - Compute the array A by looping through training data, takes time $O(MN)$
  - The smallest *K* elements in array A can be found by looping throught the array *K* times and each time picking the smallest element, in time $O(KN)$

- Better computational complexity can be obtained (in some cases) using data structures that represent the training data in such a way that for any $\mathbf{x} \in \mathcal{X}$, training data points that are close to $\mathbf{x}$ can be found quickly
- Generally challenging for high-dimensional data (high *M*), sometimes only approximate solutions possible

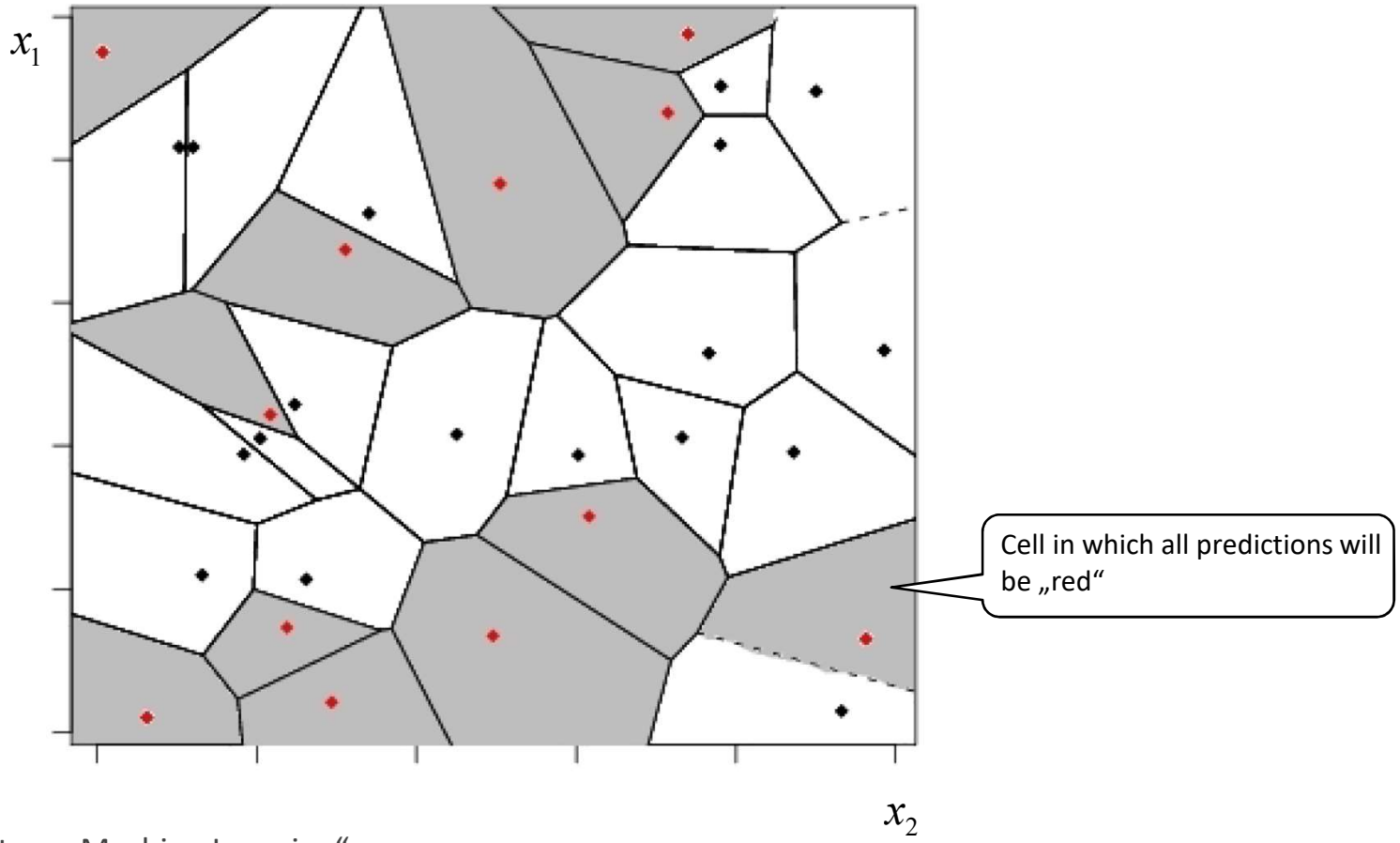
Decision Boundaries in *K*-Nearest Neighbor

- For *1*-Nearest Neighbor classification, the predictions can be visualized by looking at regions within $\mathcal{X}$ that each contain all points closest to a particular training point
- All instances in that cell will be predicted to have the same label as that training point



Decision Boundaries in *K*-Nearest Neighbor

- This leads to a partitioning of the instance space into regions (or „cells“) for which a particular class is predicted
- The partitioning is also called a **Voronoi tesselation**



Summary: Nearest Neighbor Methods

- Instead of directly relying on a vector-based feature representation such as linear models, machine learning models can also be based on distance or similarity functions between instances
- Distances or similarities can also be defined for complex instance representations such as sets or sequences
- *K*-nearest neighbor methods predict classification or regression targets based on the *K* nearest training instances (defined by a distance or similarity function)
- *K*-nearest neighbor methods do not require a training phase, but are computationally expensive at prediction time