LINEAR REGRESSION

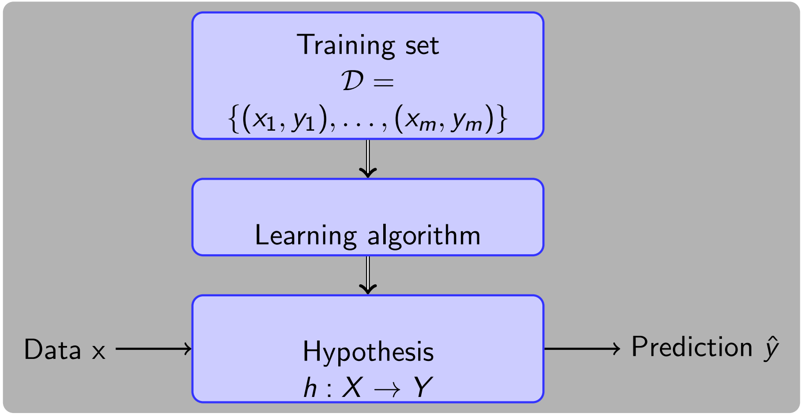
Machine Learning for Autonomous Robots

Dr. Alexander Fabisch
DFKI, Robotics Innovation Center

22 November, 2022 – Bremen, Deutschland

Problem statement

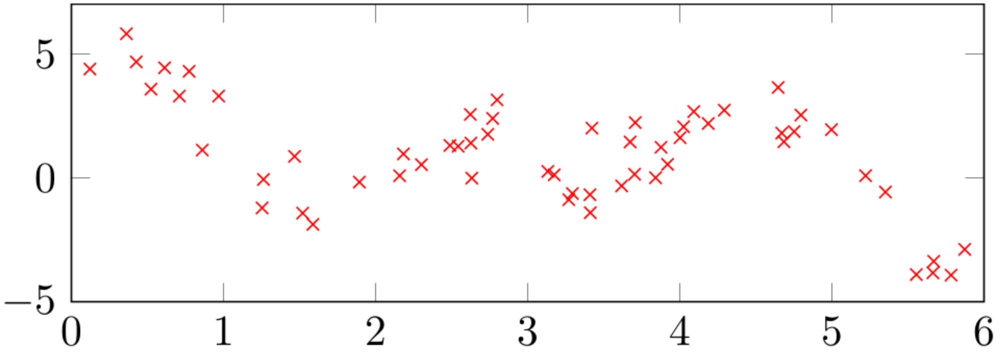
Recap: Supervised Learning



- If Y is a discrete domain: classification
- If Y is a continuous domain: regression

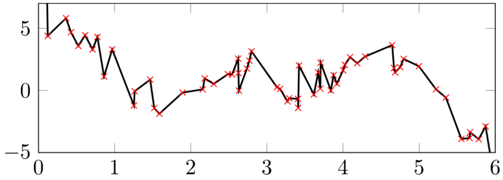
An example: Robot trajectory

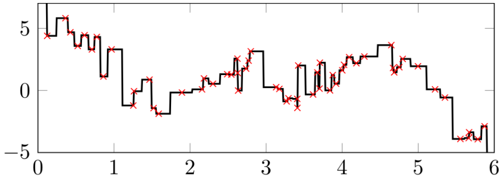
- Consider the task of estimating (a posteriori) the actual state of a robot at time t when only noisy measurements exist
- When no model of the system's dynamics is known: One could try to infer the true trajectory from the data
- Example data:

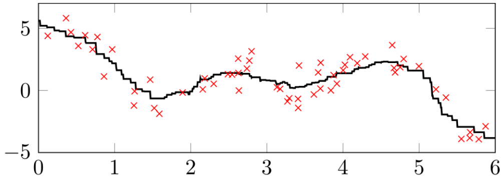


Simple approaches

- Interpolation: Cannot reduce influence of noise
- Nearest neighbor: Cannot reduce influence of noise
- Moving window average
  - How large should moving window be chosen?
  - Can this size be chosen globally?
  - How can this be extended to $X = \mathbb{R}^n$?







Linear Regression

Model: Linear function plus gaussian noise

- Assumption: True, unknown function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is linear in x: $f(x) = \mathbf{w}^T x + \mathbf{b}$, where $\mathbf{w}$ is a n-dimensional vector and $\mathbf{b}$ is a scalar (bias).
- This can be more conveniently written as $f(x) = \tilde{\mathbf{w}}^T \tilde{x}$ with $\tilde{\mathbf{w}} = (b, w_0, \ldots, w_n)^T$ and $\tilde{x} = (1, x_0, \ldots, x_n)^T$
- The measured data is then given by $y^{(i)} = \mathbf{w}^T x^{(i)} + \epsilon^{(i)}$ where $\epsilon^{(i)}$ is the noise in the i-th measurement.
- It is often assumed that the $\epsilon^{(i)}$ are normally distributed with mean 0 and unknown variance σ^2: $\epsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$

Maximum likelihood estimate (1/2)

- We would like to find a parameter vector $\hat{w}$ that is as close as possible to the true, unknown $\mathbf{w}$.
- We have two sources of information regarding $\mathbf{w}$:
  - Our prior belief about $\mathbf{w}$: $p(w)$.
  - The observed data $\mathcal{D} = \left\{\left(x^{(0)}, y^{(0)}\right), \dots, \left(x^{(m)}, y^{(m)}\right)\right\}$ and the likelihood of a given w for this data: $p(\mathcal{D}|w)$.
- For uniform priors we can choose $\hat{w}$ so that it maximizes the likelihood: $\hat{w} = \arg\max_{w \in \mathbb{R}^n} p(\mathcal{D}|w)$
- When assuming that the elements of $\mathcal{D}$ are independent: $\hat{w} = \arg\max_{w \in \mathbb{R}^n} \prod_{i=0}^{m} p\left(\left(x^{(i)}, y^{(i)}\right) \middle| w\right)$

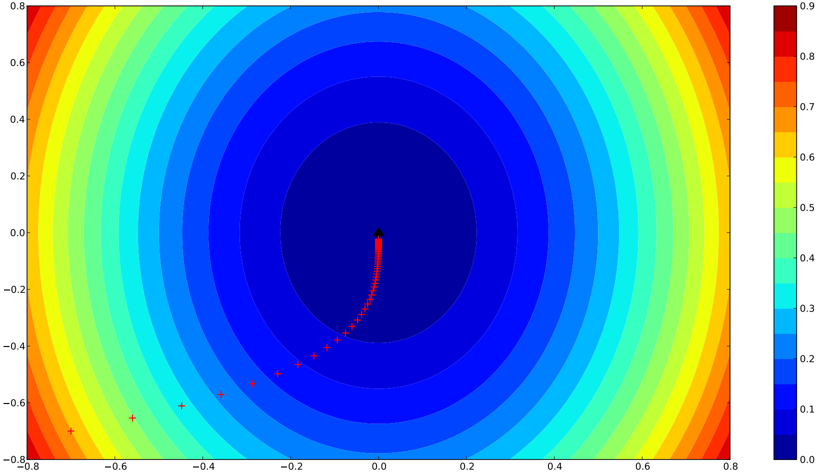
Maximum likelihood estimate (2/2)

- We assumed $y^{(i)} = \mathbf{w}^T x^{(i)} + \epsilon^{(i)}$ with $\epsilon^{(i)} \sim \mathcal{N}\left(0, \sigma^2\right)$.
- It can be shown that finding w that maximizes $\prod_{i=0}^{m} p\left(y^{(i)} | x^{(i)}, w\right) \quad \text{(MLE)}$, is equivalent to finding w that minimizes $\sum_{i=0}^{m} \left(y^{(i)} - w^T x^{(i)}\right)^2 \quad \text{(SSE)}$.
- This is called sum of squared errors (SSE)
- Linear **least squares**: $\hat{w} = \arg\min_{w \in \mathbb{R}^n} SSE(w)$
- How can one determine $\hat{w}$? $\Rightarrow$ Approach: *gradient descent*

Gradient descent

- Gradient descent is an **iterative** method that is based on the intuition that one finally reaches the bottom of a valley when always going downwards on a hill.
- Mathematical background: Negative gradient is direction of largest decrease of function value!
- Given objective function $f : \mathbb{R}^n \to \mathbb{R}$ that is differentiable. Objective: Find minimum (or maximum).
- Start with some arbitrary $x_0 \in \mathbb{R}^n$.
- Iteratively update: $x \leftarrow x - \alpha \nabla_x f(x)$
  - $\nabla_x f(x) = \left(\frac{\partial}{\partial x_0} f(x), \ldots, \frac{\partial}{\partial x_{n-1}} f(x) \right)$
  - α is a meta-parameter called “learning rate”
- Until convergence, i.e. until x remains (nearly) constant

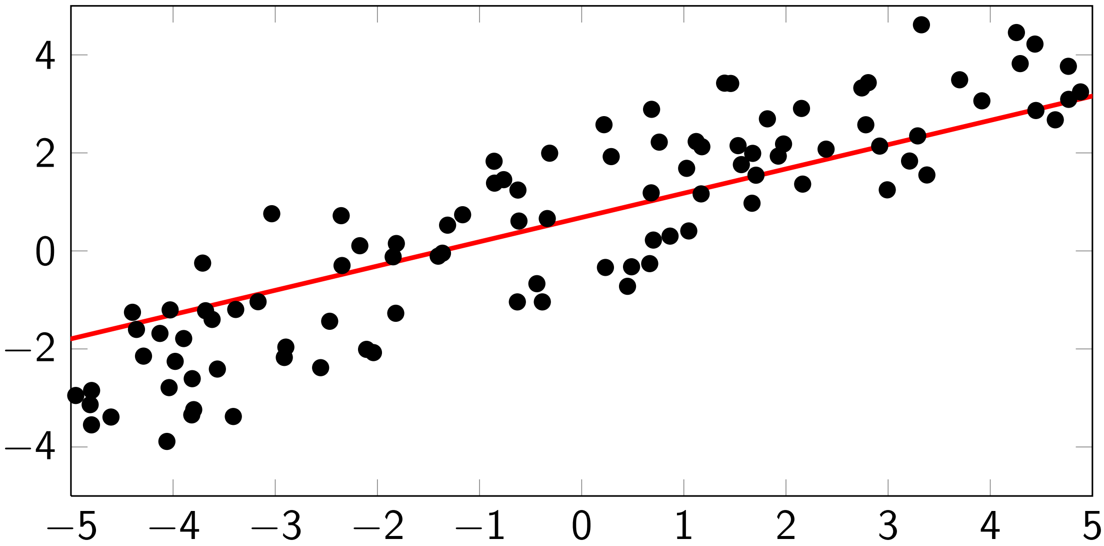
Example: Gradient descent



Applying gradient descent to linear regression

- Objective: Minimize function $SSE(w) = \frac{1}{2} \sum_{i=0}^{m} \left(y^{(i)} - w^T x^{(i)}\right)^2$
- Partial derivatives: $\frac{\partial}{\partial w_j} f(w) = - \sum_{i=0}^{m} x_j^{(i)} \left(y^{(i)} - w^T x^{(i)}\right)$
- Batch update rule:
$$w_j \leftarrow w_j + \alpha \left(\sum_{i=0}^{m} x_j^{(i)} \left(y^{(i)} - w^T x^{(i)}\right) \right) \text{ (for every } j\text{)}$$
- Disadvantage: Requires to loop over the whole training set before anything is improved
- Alternative: "Stochastic gradient descent"
- for $i = 0, \ldots, m$:
$$w_j \leftarrow w_j + \alpha \left(x_j^{(i)} \left(y^{(i)} - w^T x^{(i)}\right) \right) \text{ (for every } j\text{)}$$

Example: Linear regression with gradient descent



Remarks

- The SSE function is quadratic convex. There are no local, non-global optima, no plateaus, and no side constraints.
  - In other cases, there is no guarantee that gradient descent converges to the global minimum (can converge to local minima).
  - Gradient descent can also get stuck in plateaus.
- Learning rate is often critical:
  - Too small: Slow convergence
  - Too large: Jump over minima and potential divergence

Normal Equations (1/2)

- Gradient descent can be slow
  (require many (expensive) iterations)
- Can the optimal $\hat{w}$ be computed directly?
- For linear regression: Yes.

$$
\begin{aligned}
SSE(w) &= \frac{1}{2}\sum_{i=0}^{m}\left(y^{(i)} - w^T x^{(i)}\right)^2 \\
&= \frac{1}{2}\left\|\begin{pmatrix} y^{(0)} \\ \vdots \\ y^{(m)} \end{pmatrix} - \begin{pmatrix} 1 & x_0^{(0)} & \dots & x_{n-1}^{(0)} \\ \vdots & \vdots & & \vdots \\ 1 & x_0^{(m)} & \dots & x_{n-1}^{(m)} \end{pmatrix}\begin{pmatrix} w_0 \\ \vdots \\ w_n \end{pmatrix}\right\|_2^2 \\
&= \frac{1}{2}\|y - Xw\|_2^2 = \frac{1}{2}(y - Xw)^T(y - Xw)
\end{aligned}
$$

Normal Equations (2/2)

$SSE(w)$ is quadratic and ≥ 0 for all w. Thus, its global minimum is the unique root of its derivative:

$$\begin{aligned}
0 &= \nabla_w SSE(\hat{w}) \\
&= \frac{1}{2}\nabla_w \left[(y - Xw)^T(y - Xw)\right]\Big|_{w=\hat{w}} \\
&= \frac{1}{2}\nabla_w \left[y^T y - w^T X^T y - y^T X w + w^T X^T X w\right]\Big|_{w=\hat{w}} \\
&= -X^T y + X^T X \hat{w}
\end{aligned}$$

Since X (usually) has full column rank:

$$\hat{w} = (X^T X)^{-1} X^T y$$

Online Learning

- $\hat{w} = (X^T X)^{-1} X^T y$
- Can this be calculated on a mobile system?
- Memory/processing effort with new incoming (Big) data?
- Sherman-Morrison-Woodbury formula
  - $\rightarrow$ Fast online update for $\left(X(m+1)^T X(m+1)\right)^{-1}$
  - $X(m+1)^T X(m+1) = X(m)^T X(m) + \left(x^{(m+1)}\right)^T x^{(m+1)}$
  - Fixed dimensions of $\left(X(m)^T X(m)\right)^{-1}$ and $X(m)^T y(m)$
  - $X(m+1)^T y(m+1) = X(m)^T y(m) + \left(x^{(m+1)}\right)^T y^{(m+1)}$

Other target functions

- The “linear function plus gaussian noise” model is often too simple.
- More general classes of functions have a better chance of fitting the data well
- Classes of functions can be defined by a projection function ϕ
- Examples of classes of functions (for $X = \mathbb{R}$):
  - Linear functions: $\phi(x) = (1, x)^T$
  - Polynomials: $\phi(x) = (x^0, x^1, \ldots, x^k)^T$, k = degree of polynomial
  - Linear combination of a set of sinusoidal functions:
    $\phi(x) = (sin(0x/k), sin(1x/k), sin(2x/k), \ldots, sin(x))^T$

Other target functions

Objective: Minimize function

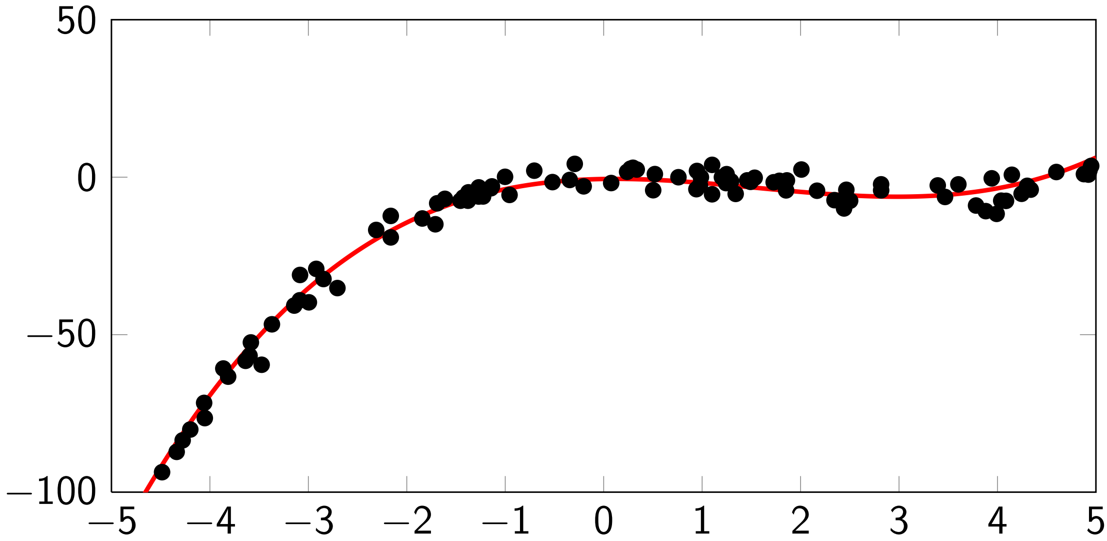
$$
\begin{aligned}
SSE(w) &= \frac{1}{2}\sum_{i=0}^{m}\left(y^{(i)} - w^T\phi\left(x^{(i)}\right)\right)^2 \\
&= \frac{1}{2}(y - X_\phi w)^T(y - X_\phi w)
\end{aligned}
$$

$$
\text{with } X_\phi = \begin{pmatrix} \phi(x^{(0)})_0 & \phi(x^{(0)})_1 & \dots & \phi(x^{(0)})_k \\ \vdots & \vdots & & \vdots \\ \phi(x^{(m)})_0 & \phi(x^{(m)})_1 & \dots & \phi(x^{(m)})_k \end{pmatrix}
$$

This can still be solved with the normal equations:

$$
\hat{w} = (X_\phi^T X_\phi)^{-1} X_\phi^T y
$$

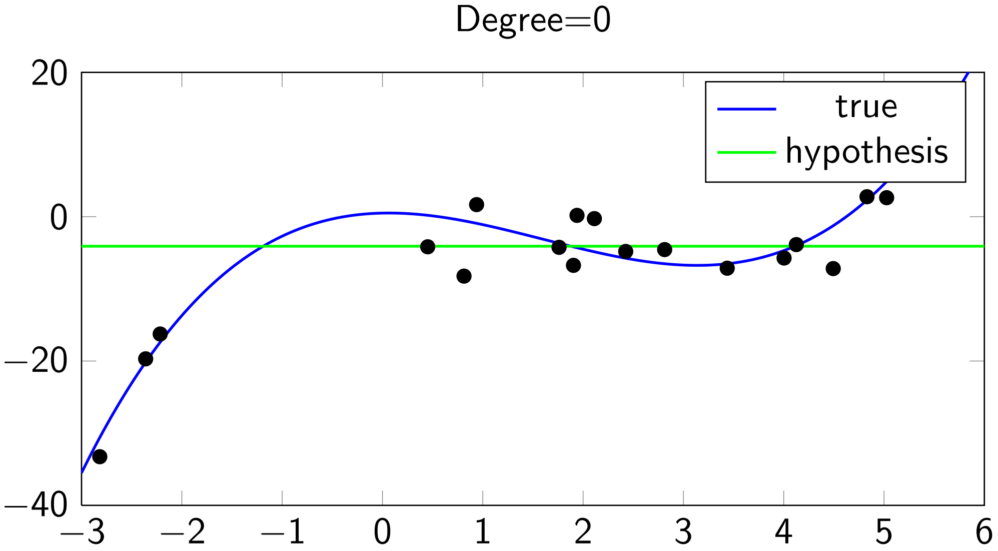
Gradient descent for $\phi(x) = (x^0, x^1, x^2, x^3)$



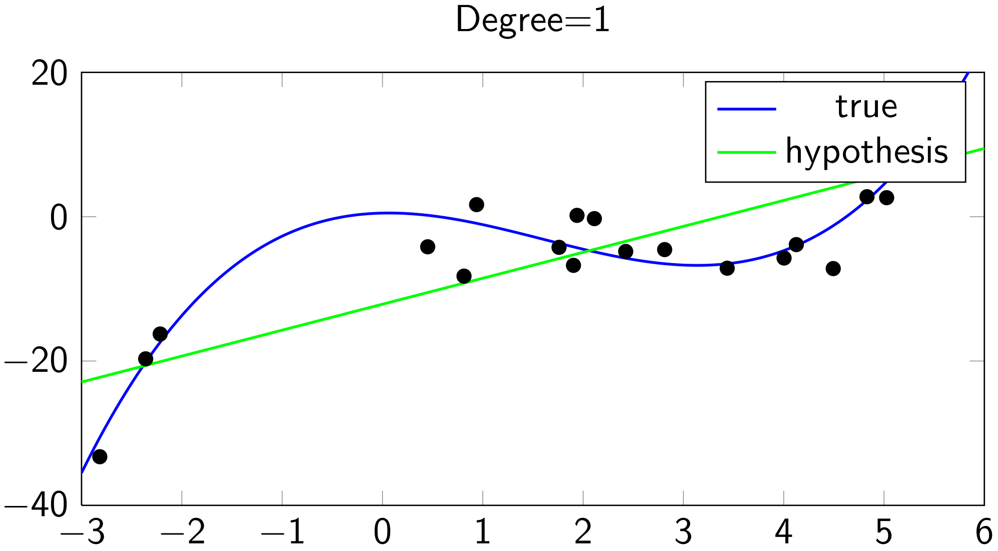
How to select ϕ

- The function ϕ defines which functions can be represented. It is also called "the model"
- Intuitively, ϕ should somehow reflect the properties of the unknown target function.
- But even when it is known that the true function is a polynomial: How should k be chosen?
- First idea:
  - Choose k as large as possible since the class of learnable functions becomes bigger.
  - Even linear functions remain learnable
    (set all higher-order w_i to 0)
- Is this really a good idea?

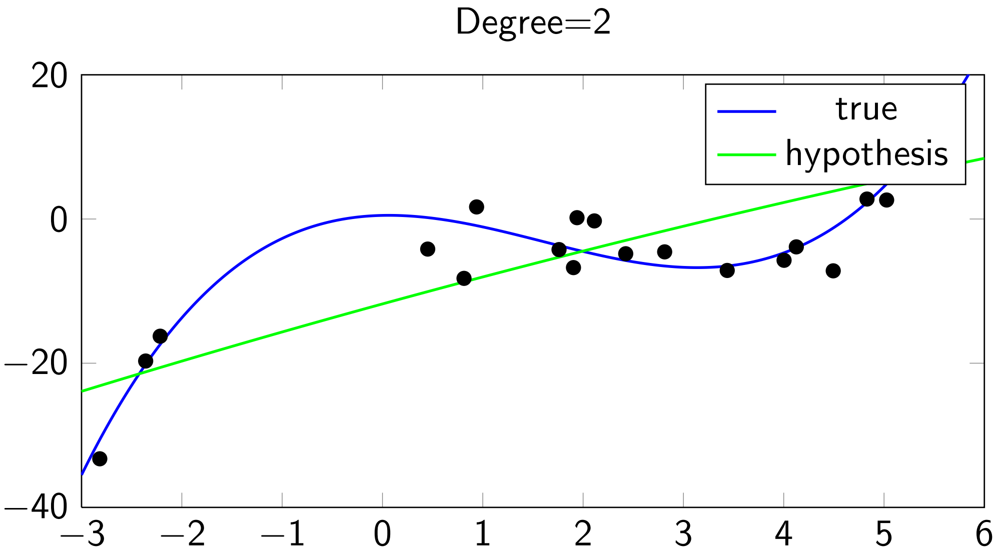
Polynomial model



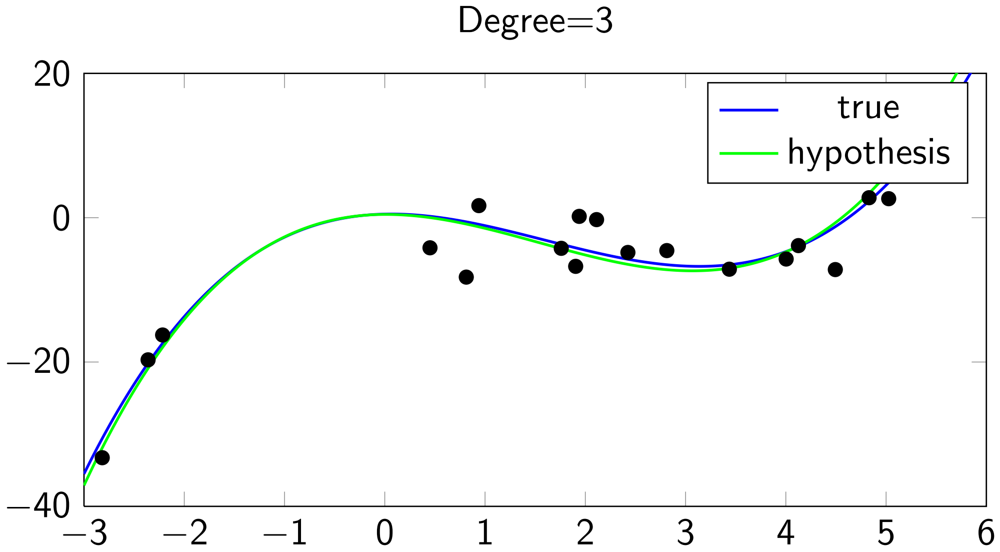
Polynomial model



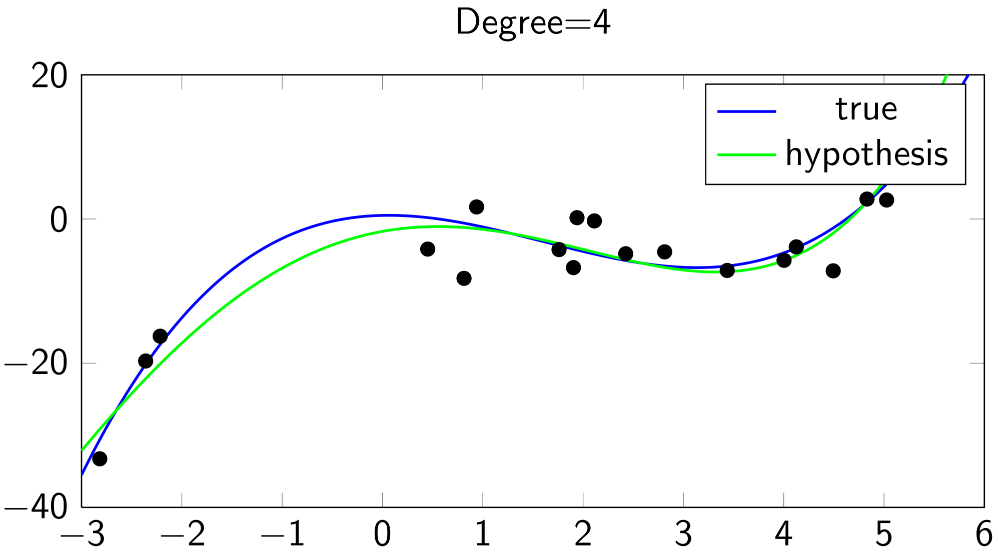
Polynomial model



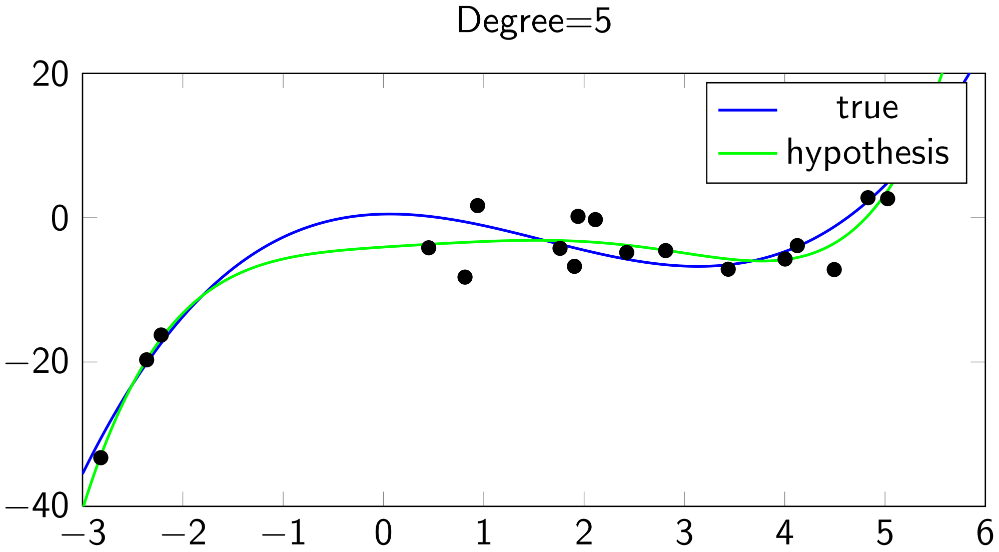
Polynomial model



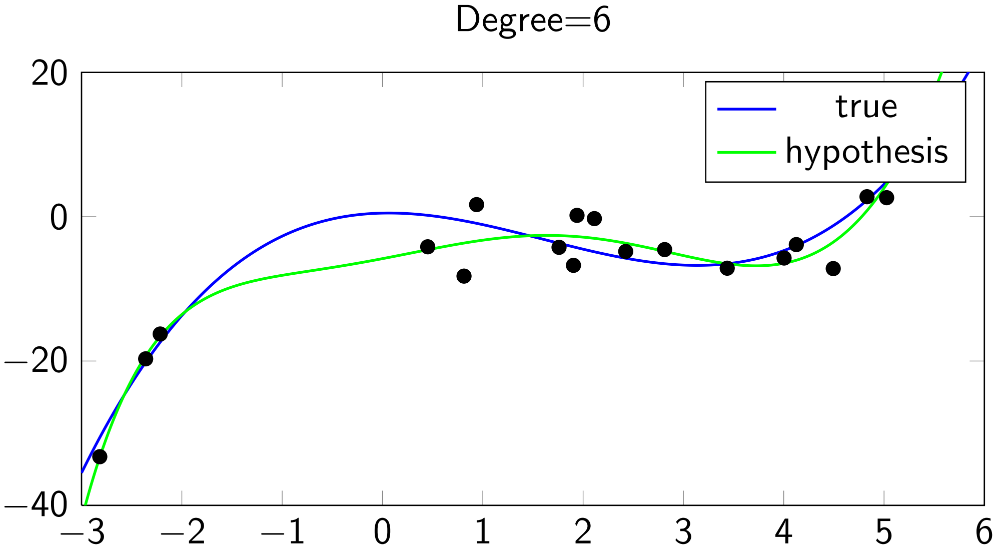
Polynomial model



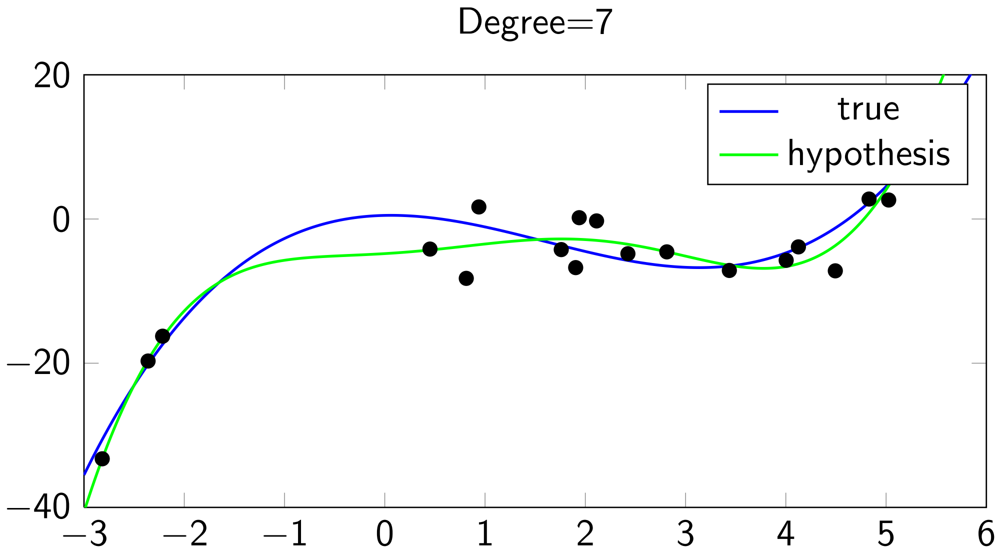
Polynomial model



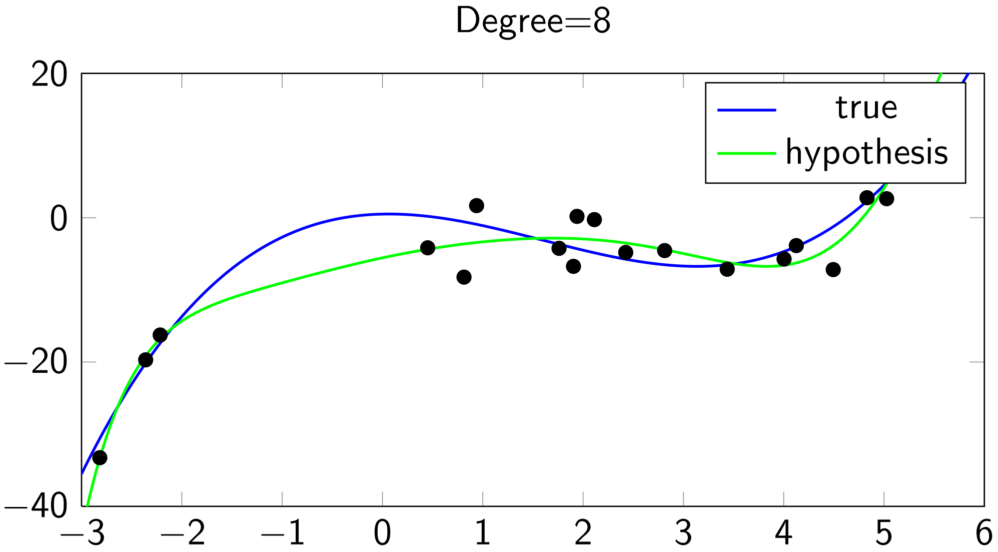
Polynomial model



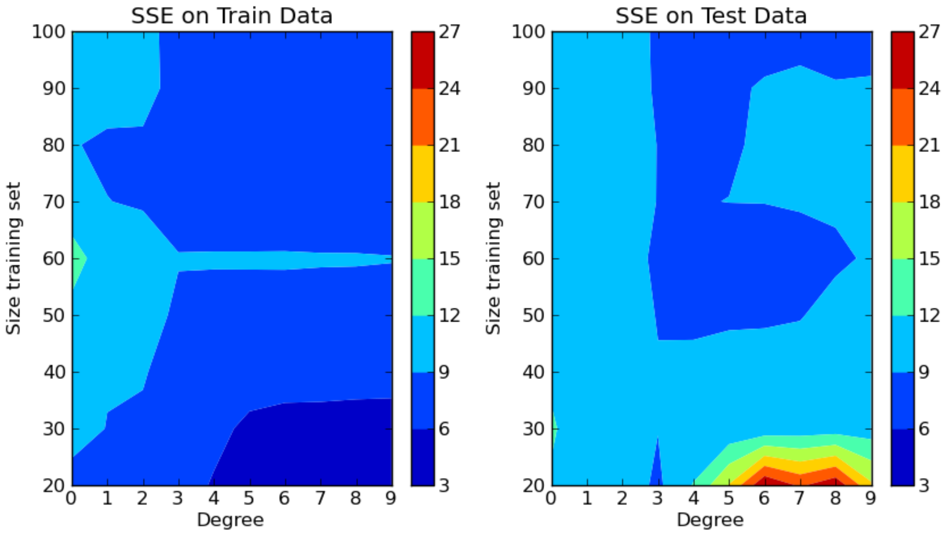
Polynomial model



Polynomial model



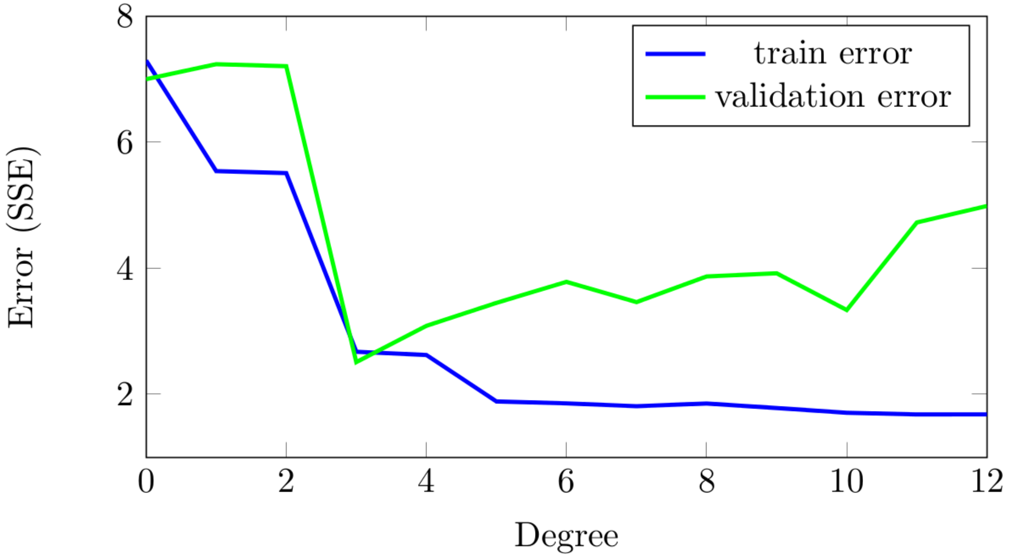
Train- and Testerror



Overfitting and Underfitting

- Larger k allows to fit the training data points better
- But: The hypothesis was maximally similar to true function for $k = 3$.
- Explanation: Large values of k (an "increased complexity") allows to fit to the noise in the training data ("Overfitting")
- Choosing k too small results in a overly simple hypothesis ("Underfitting")
- How can k be chosen when true function is unknown?
- One approach: Test learned hypothesis on hold-out validation data ($\rightarrow$ cross-validation)

Train- and Validation-Error



Model Selection

- This approach can be (partially) automated!
- Split labeled data $\mathcal{D}$ into $\mathcal{D}_{train}$ and $\mathcal{D}_{validation}$
- For each value of k:
  - Determine a hypothesis on $\mathcal{D}_{train}$ (using normal equations)
  - Evaluate learned hypothesis on $\mathcal{D}_{validation}$
- Select k with lowest SSE on validation data
- Train a polynomial with degree k on whole $\mathcal{D}$
- Can be made more accurate (cross-validation) and more general (not restricted to polynomial models)

Advanced Topics

Regularization

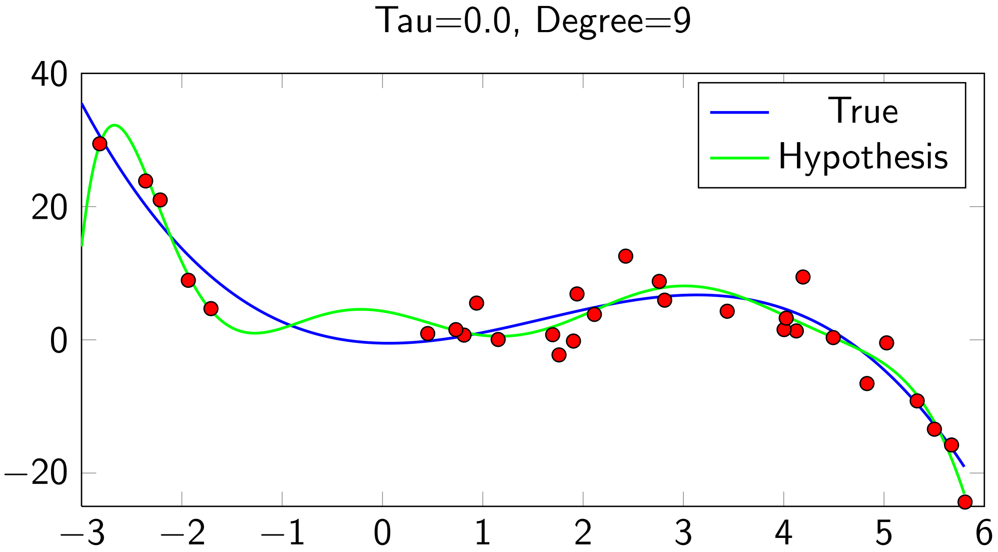
- Techniques like cross-validation allow to choose appropriate model complexity
- But: CV is computationally costly (besides some other minor drawbacks)
- Alternative: Regularization
- Idea behind regularization: Use model with high complexity but enforce that solution is somehow “smooth”
- Regularization is also recommendable from a computational point of view: It reduces problems that occur due to numerical instability.

Tikhonov Regularization

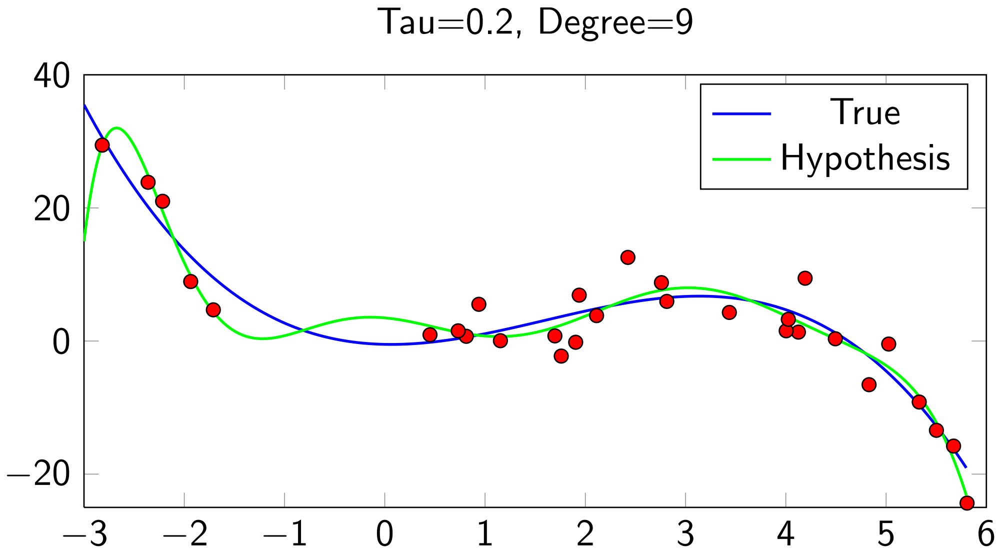
- Linear Least Squares: $\hat{w} = \arg\min_{w} \frac{1}{2} \|y - Xw\|_2^2$
- Linear Least Squares + Tikhonov Regularization: $\hat{w} = \arg\min_{w} \frac{1}{2} \|y - Xw\|_2^2 + \|\Gamma w\|_2^2$
- In many cases, Γ is chosen to be diagonal, e.g. $\Gamma = diag(\tau)$ for some $\tau \in \mathbb{R}$ Could set bias to certain weights.
- For $\tau = 0$: Equal to (unregularized) Linear Least Squares
- For $\tau > 0$: Giving preference to solutions w with smaller norms (the stronger the larger τ)
- Normal equation solution:

$$\hat{w} = (X^T X + \Gamma^T \Gamma)^{-1} X^T y$$

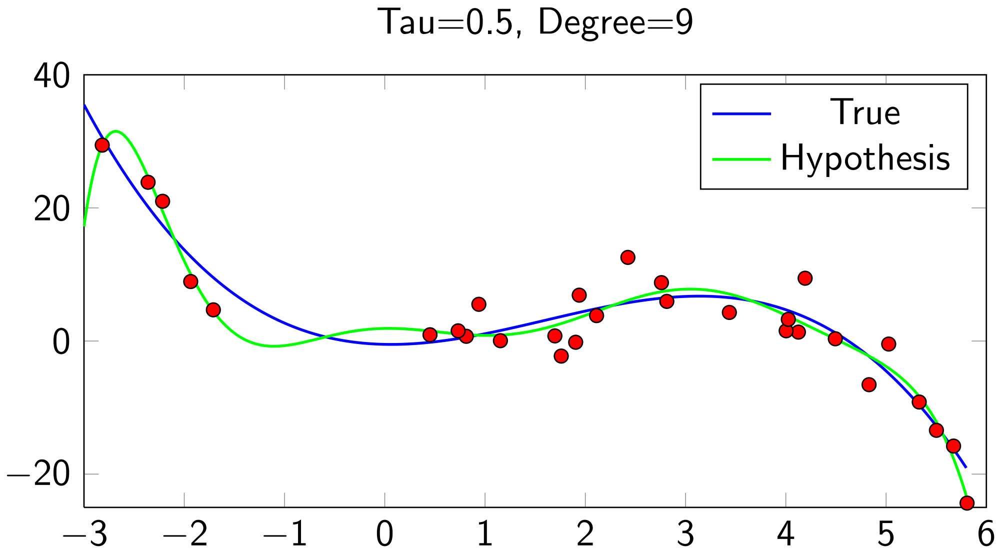
Ridge regression example



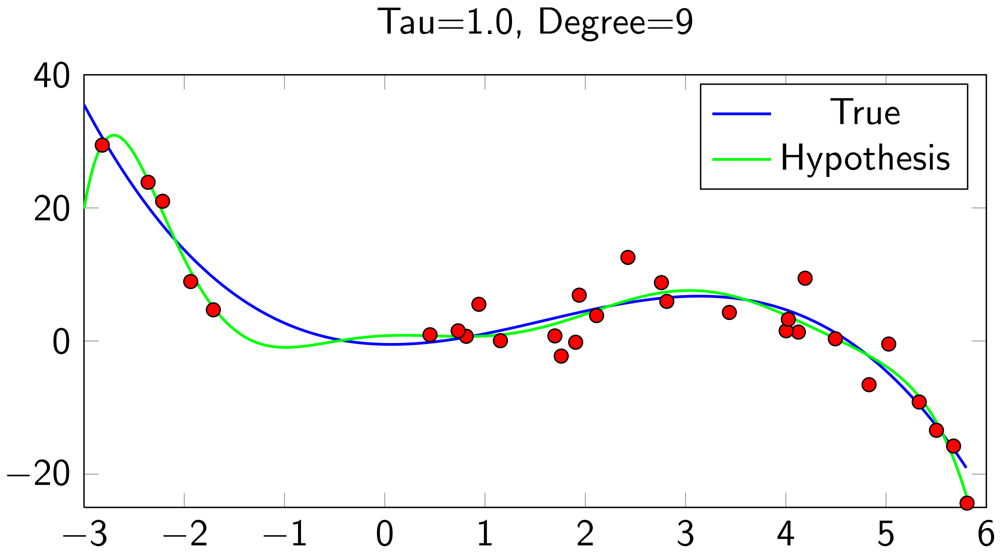
Ridge regression example



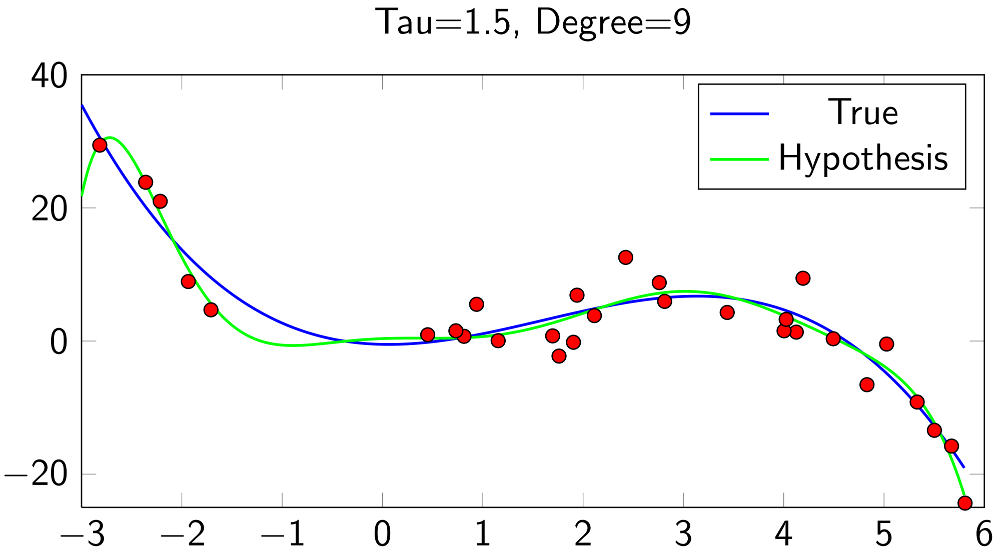
Ridge regression example



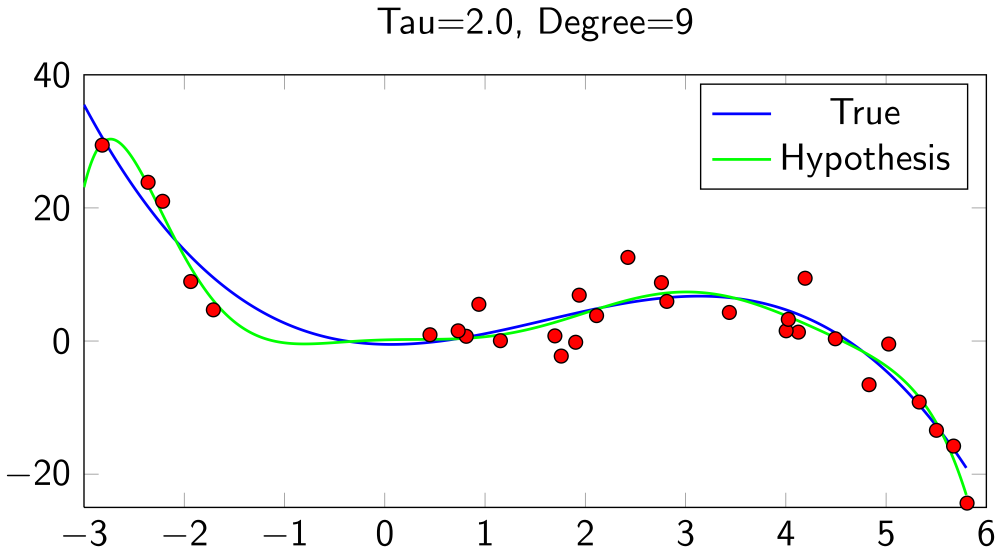
Ridge regression example



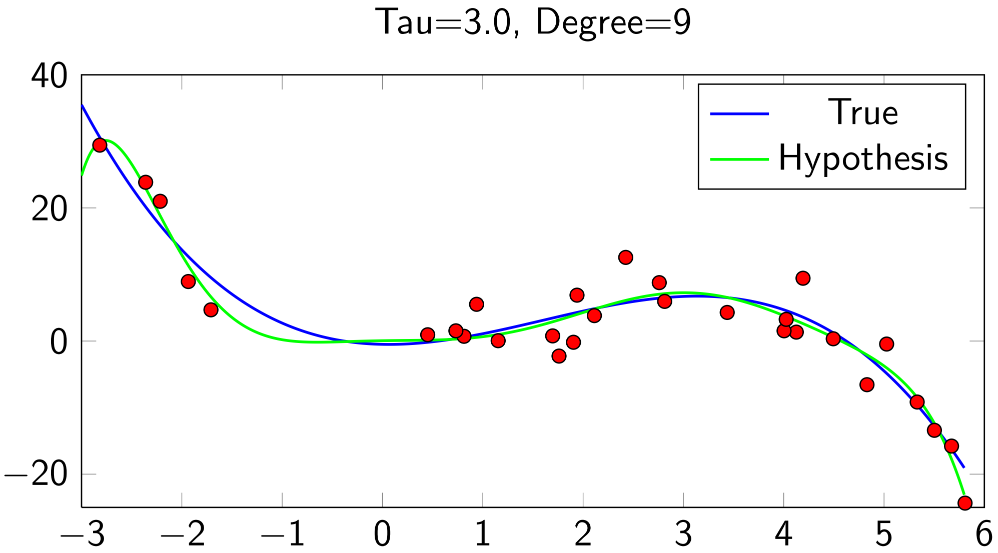
Ridge regression example



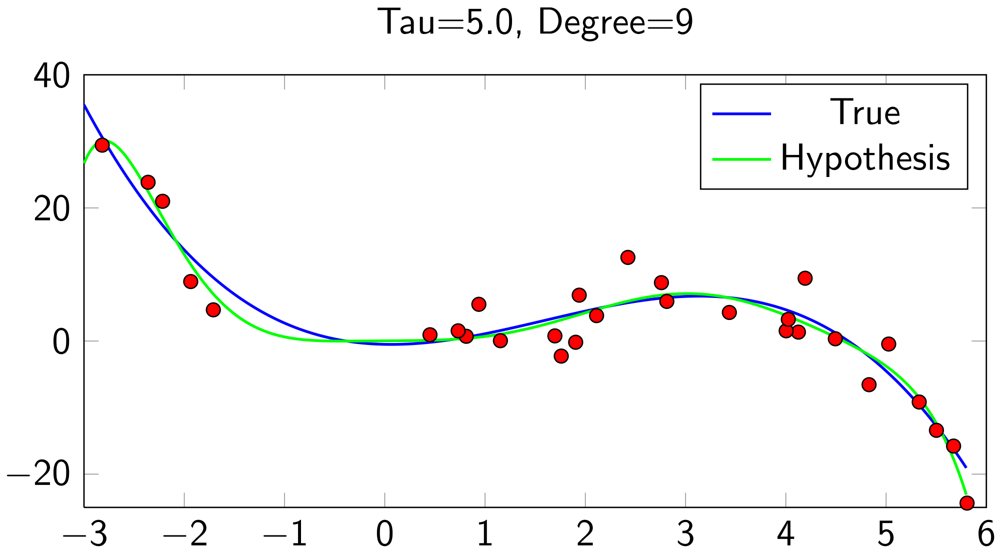
Ridge regression example



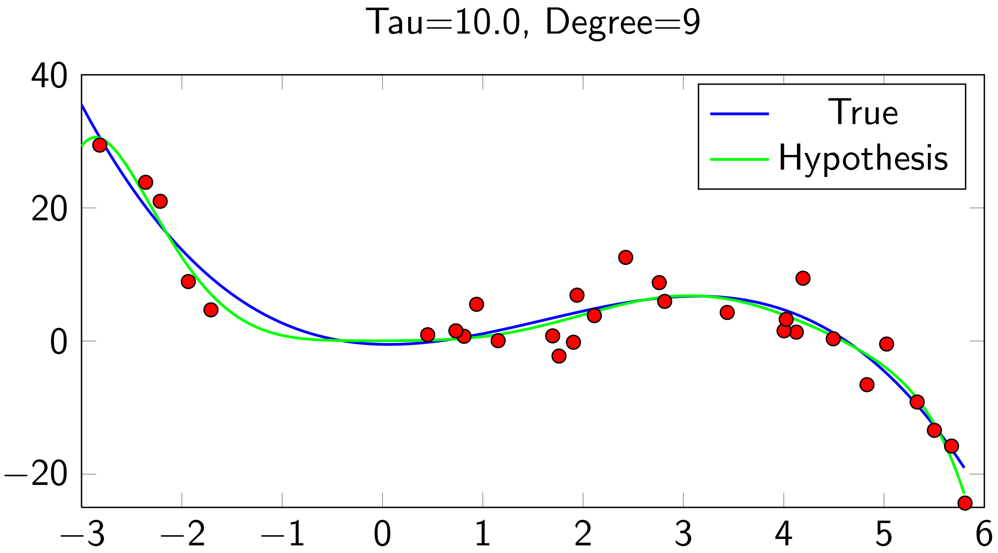
Ridge regression example



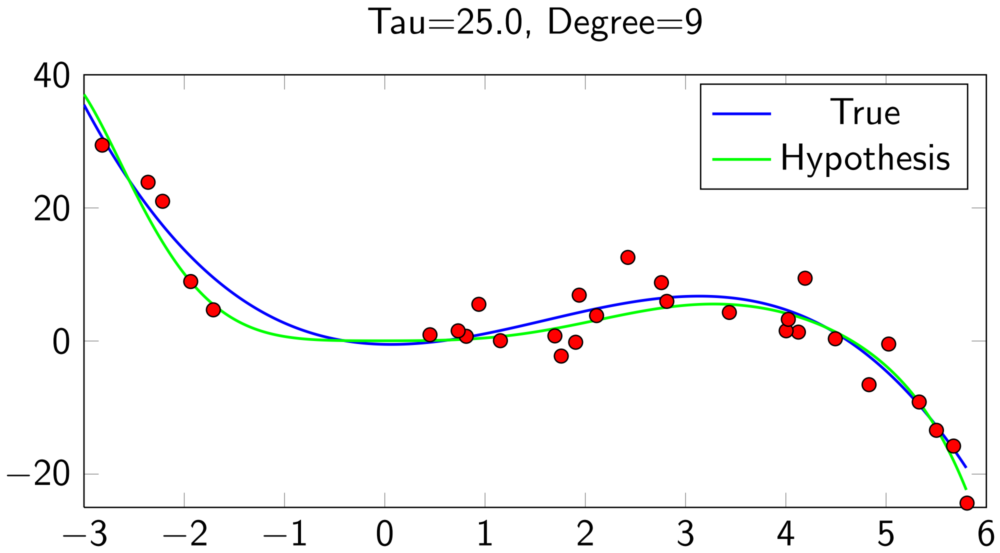
Ridge regression example



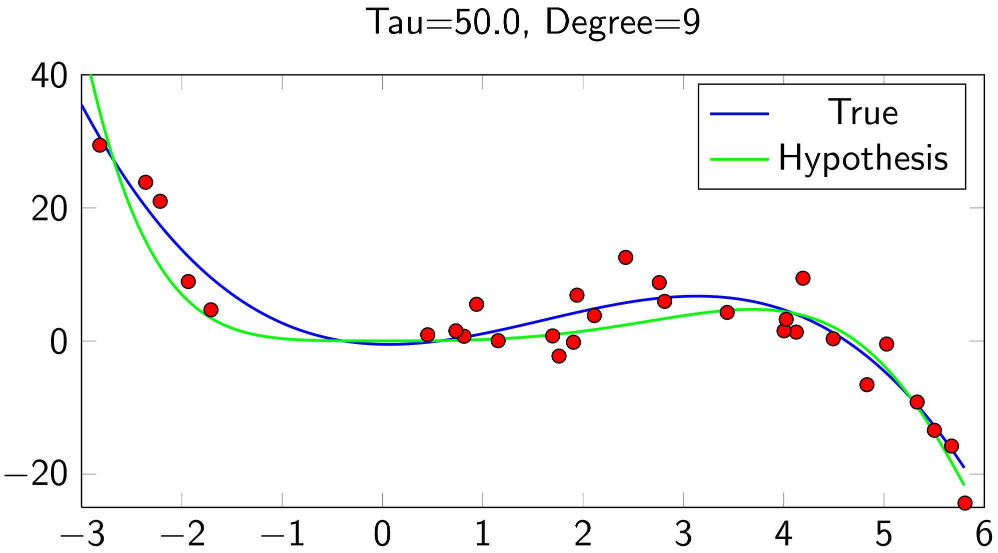
Ridge regression example



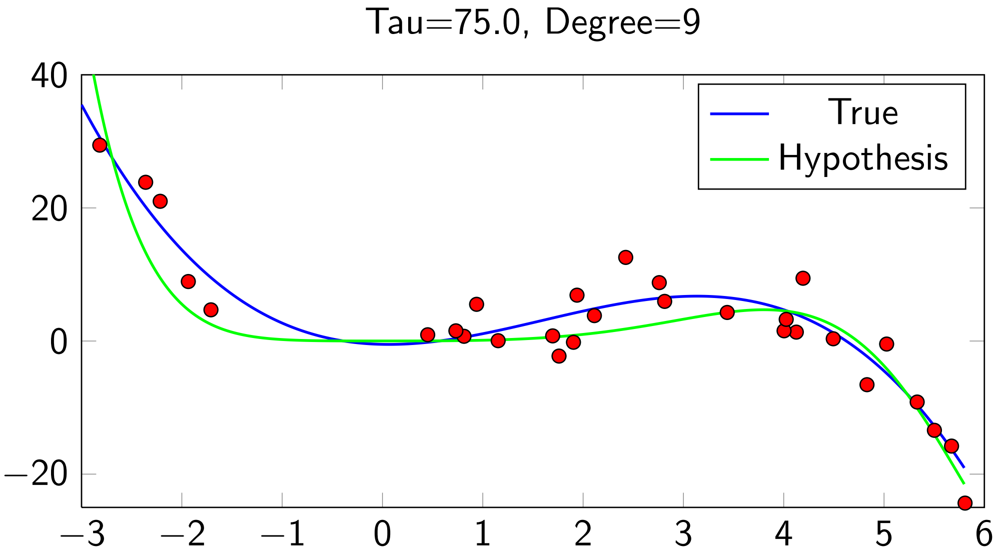
Ridge regression example



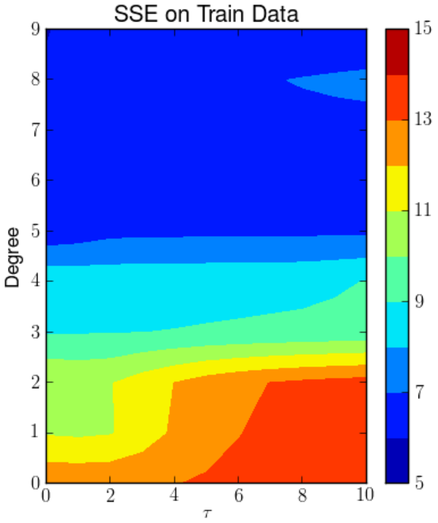
Ridge regression example

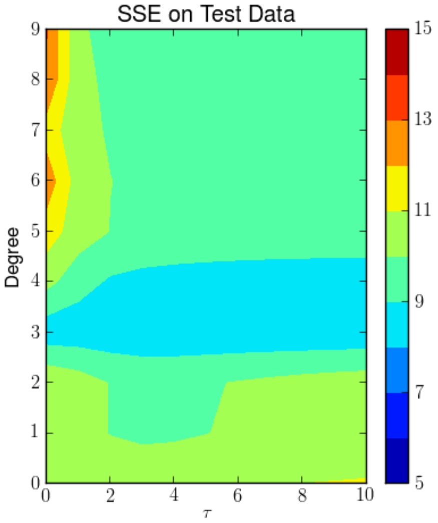


Ridge regression example

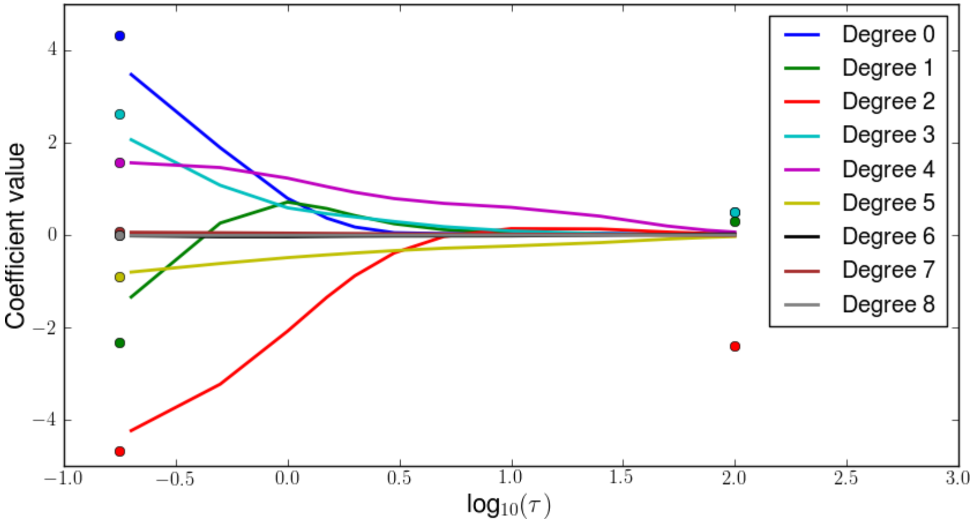


Example: SSE on Train and Test Data





Example: Coefficients (k=9)



Left: $\tau = 0$; right: true coefficients (blue==dark blue)

Remarks

- Linear regression combined with Tikhonov regularization is also called “Ridge Regression”
- Regularization cannot magically remove the risk of overfitting
- Instead of choosing the model complexity one has to choose the regularization parameter(s)
- But: Often the specific choice of the regularization parameter is less critical
- Furthermore: Remedies issues with numeric stability
- Even though Tikhonov regularization seems rather arbitrary, it can be motivated statistically
  (prior, German: A-priori-Wahrscheinlichkeit)

Thank You!

Please feel free to ask questions in the forums.