

*** Applied Machine Learning Fundamentals ***

Regression

Daniel Wehner, M.Sc.

SAP SE / DHBW Mannheim

Winter term 2023/2024



Find all slides on [GitHub](https://github.com/DaWe1992/Applied_ML_Fundamentals) (DaWe1992/Applied_ML_Fundamentals)

Lecture Overview

Unit I	Machine Learning Introduction
Unit II	Mathematical Foundations
Unit III	Bayesian Decision Theory
Unit IV	Regression
Unit V	Classification I
Unit VI	Evaluation
Unit VII	Classification II
Unit VIII	Clustering
Unit IX	Dimensionality Reduction

Agenda for this Unit

① Introduction

② Solutions to Regression

③ Basis Function Regression

④ Regularization Techniques

⑤ Wrap-Up

Section: Introduction

What is Regression?
Least Squares Error Function

Regression

Type of target variable

Continuous

Type of training information

Supervised

Example Availability

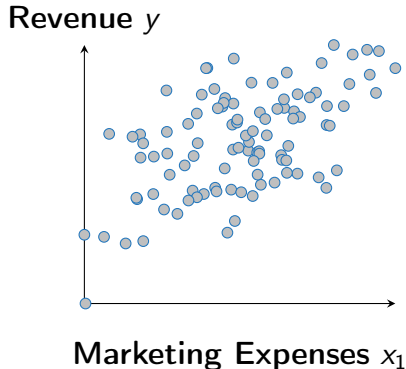
Batch learning

Algorithm sketch: Given the training data \mathcal{D} , the algorithm derives a function of the type

$$h_{\theta}(\mathbf{x}) = \theta_0 + \theta_1 x_1 + \cdots + \theta_m x_m \quad \mathbf{x} \in \mathbb{R}^m, \theta \in \mathbb{R}^{m+1} \quad (1)$$

from the data. θ is the parameter vector containing the coefficients to be estimated by the regression algorithm. Once θ is learned, it can be used for prediction.

Example Data Set: Revenues



- Find a linear function:

$$h_{\theta}(\mathbf{x}) = \theta_0 + \theta_1 x_1 + \cdots + \theta_m x_m$$

- Usually: $x_0 = 1$:

$$\hat{\mathbf{x}} \in \mathbb{R}^{m+1} = [1 \ \mathbf{x}]^T$$

$$h_{\theta}(\hat{\mathbf{x}}) = \sum_{j=0}^m \theta_j x_j = \boldsymbol{\theta}^T \hat{\mathbf{x}}$$

Error Function for Regression

- We need an error function $\mathcal{J}(\boldsymbol{\theta})$ in order to know how good the function fits:

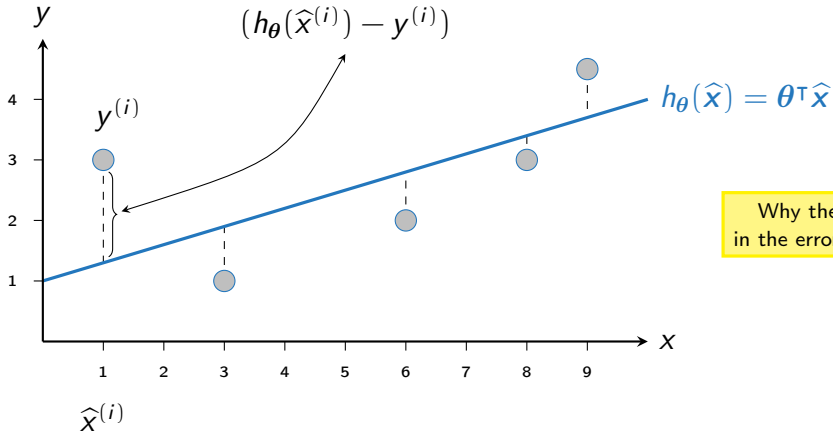
$$\mathcal{J}(\boldsymbol{\theta}) = \frac{1}{2n} \sum_{i=1}^n (h_{\boldsymbol{\theta}}(\hat{\mathbf{x}}^{(i)}) - y^{(i)})^2 \quad (2)$$

- We want to minimize $\mathcal{J}(\boldsymbol{\theta})$:

$$\min_{\boldsymbol{\theta}} \frac{1}{2n} \sum_{i=1}^n (h_{\boldsymbol{\theta}}(\hat{\mathbf{x}}^{(i)}) - y^{(i)})^2$$

- This is **ordinary least squares (OLS)**

Error Function Intuition



Why the **square**
in the error function?

Section: Solutions to Regression

Closed-Form Solutions and Normal Equation
Gradient Descent

Closed-Form Solutions

- Usual approach (for two unknowns): Calculate θ_0 and θ_1 according to

sample mean \bar{x}

$$\theta_0 = \bar{y} - \theta_1 \bar{x} \qquad \theta_1 = \frac{\sum_{i=1}^n (x^{(i)} - \bar{x}) \cdot (y^{(i)} - \bar{y})}{\sum_{i=1}^n (x^{(i)} - \bar{x})^2} \quad (3)$$

- 'Normal equation' (scales to arbitrary dimensions):

$$\theta = \underbrace{(\hat{X}^\top \hat{X})^{-1} \hat{X}^\top}_{\text{Moore-Penrose pseudo-inverse}} y \quad (4)$$

\hat{X} is called 'design matrix' or 'regressor matrix'

Design Matrix / Regressor Matrix

- The design matrix $\hat{\mathbf{X}} \in \mathbb{R}^{n \times (m+1)}$ looks as follows:

In the following

$$\hat{\mathbf{X}} \equiv \mathbf{X}$$

$$\hat{\mathbf{X}} = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_m^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \cdots & x_m^{(2)} \\ 1 & x_1^{(3)} & x_2^{(3)} & \cdots & x_m^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(n)} & x_2^{(n)} & \cdots & x_m^{(n)} \end{pmatrix} \quad (5)$$

- And the $n \times 1$ label vector:

$$\mathbf{y} = (y^{(1)}, y^{(2)}, y^{(3)}, \dots, y^{(n)})^\top$$



Derivation of the Normal Equation

- The derivation involves a bit of linear algebra
- Step ❶: Rewrite $\mathcal{J}(\boldsymbol{\theta})$ in matrix-vector notation:

$$\begin{aligned}\mathcal{J}(\boldsymbol{\theta}) &= 1/2(\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^\top(\mathbf{X}\boldsymbol{\theta} - \mathbf{y}) \\ &= 1/2((\mathbf{X}\boldsymbol{\theta})^\top - \mathbf{y}^\top)(\mathbf{X}\boldsymbol{\theta} - \mathbf{y}) \\ &= 1/2((\mathbf{X}\boldsymbol{\theta})^\top\mathbf{X}\boldsymbol{\theta} - (\mathbf{X}\boldsymbol{\theta})^\top\mathbf{y} - \mathbf{y}^\top(\mathbf{X}\boldsymbol{\theta}) + \mathbf{y}^\top\mathbf{y}) \\ &= 1/2(\boldsymbol{\theta}^\top\mathbf{X}^\top\mathbf{X}\boldsymbol{\theta} - 2(\mathbf{X}\boldsymbol{\theta})^\top\mathbf{y} + \mathbf{y}^\top\mathbf{y})\end{aligned}$$

- To be continued...



Derivation of the Normal Equation (Ctd.)

- Step ②: Calculate the derivative of $J(\theta)$ and set it to zero:

$$\begin{aligned}\nabla_{\theta} J(\theta) &= 1/2(2\mathbf{X}^T \mathbf{X} \theta - 2\mathbf{X}^T \mathbf{y}) \stackrel{!}{=} 0 \\ \Leftrightarrow \mathbf{X}^T \mathbf{X} \theta &= \mathbf{X}^T \mathbf{y}\end{aligned}$$

- If $\mathbf{X}^T \mathbf{X}$ is invertible, we can multiply both sides by $(\mathbf{X}^T \mathbf{X})^{-1}$:

Normal equation:

$$\theta = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

Problems with Matrix Inversion?

- What if $(\mathbf{X}^\top \mathbf{X})^{-1}$ does not exist?
- Problems and solutions:
 - ① Linearly dependent (redundant) features or design matrix does not have full rank? (E. g. size in m^2 and size in feet^2)
⇒ **Delete correlated features**
 - ② Too many features ($m > n$)?
⇒ **Delete features (e. g. using PCA) / add training examples**
 - ③ Other numerical instabilities?
⇒ **Add a regularization term** (later)
 - ④ Computationally too expensive?
⇒ **Use gradient descent**



Gradient Descent

- We want to minimize a smooth function $\mathcal{J} : \mathbb{R}^{m+1} \rightarrow \mathbb{R}$:

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^{m+1}} \mathcal{J}(\boldsymbol{\theta})$$

- Update the parameters iteratively:

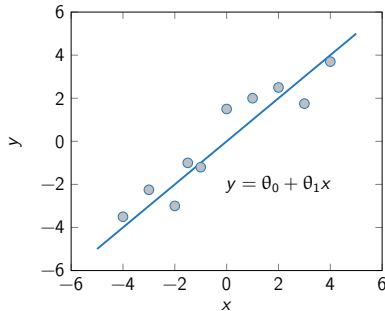
$$\boldsymbol{\theta}^{(t+1)} \longleftarrow \boldsymbol{\theta}^{(t)} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}^{(t)}) \quad (6)$$

- where $\alpha > 0$ (**learning rate**) and $\nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta})$ is the gradient of $\mathcal{J}(\boldsymbol{\theta})$ w. r. t. $\boldsymbol{\theta}$:

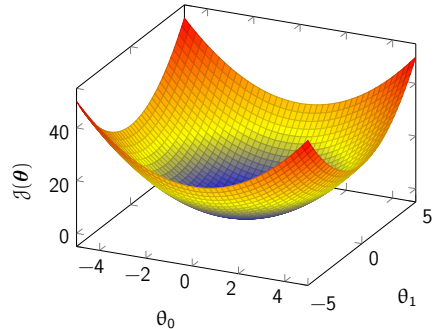
$$\nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}) = \left(\frac{\partial \mathcal{J}(\boldsymbol{\theta})}{\partial \theta_0}, \frac{\partial \mathcal{J}(\boldsymbol{\theta})}{\partial \theta_1}, \dots, \frac{\partial \mathcal{J}(\boldsymbol{\theta})}{\partial \theta_m} \right)^\top$$

Data Input Space vs. Hypothesis Space

Data input space



Hypothesis space \mathcal{H}



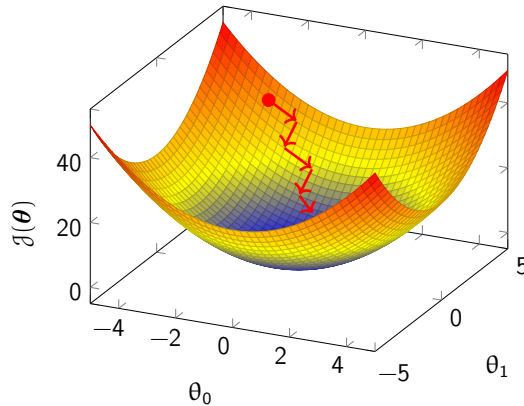
Data Input Space vs. Hypothesis Space (Ctd.)

- **Data input space**
 - Determined by the $m + 1$ **attributes** of the data set $x_0, x_1, x_2, \dots, x_m$
 - Often high-dimensional
- **Hypothesis space \mathcal{H}**
 - Determined by the **number of parameters** of the model
 - Each point in the hypothesis space corresponds to a **specific assignment of model parameters**
 - The error function gives information about how good this assignment is
 - **Gradient descent is applied in the hypothesis space \mathcal{H}**



Data Input Space vs. Hypothesis Space (Ctd.)

Visualization of Gradient Descent in 3 Dimensions



Versions of Gradient Descent

- Assume some training data \mathcal{D} : $\{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^n$
- Squared error for a **single** example: $\ell(y_{pred}, y_{true}) = (y_{pred} - y_{true})^2$
- Our objective is to minimize the **total** error:

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^{m+1}} \mathcal{J}(\boldsymbol{\theta}) = \min_{\boldsymbol{\theta} \in \mathbb{R}^{m+1}} \sum_{i=1}^n \ell(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}), y^{(i)})$$

- Three versions of gradient descent:
 - 1 Batch gradient descent
 - 2 Stochastic gradient descent
 - 3 Mini-batch gradient descent

Versions of Gradient Descent (Ctd.)

- **Batch gradient descent**: Compute gradient based on ALL data points

$$\boldsymbol{\theta}^{(t+1)} \longleftarrow \boldsymbol{\theta}^{(t)} - \alpha \sum_{i=1}^n \nabla \ell(h_{\boldsymbol{\theta}^{(t)}}(\mathbf{x}^{(i)}), y^{(i)}) \quad (7)$$

- **Stochastic gradient descent**: Compute gradient based on a SINGLE data point (**pick training example randomly and not sequentially!**)
- For $i \in \{1, \dots, n\}$ do:

$$\boldsymbol{\theta}^{(t+1)} \longleftarrow \boldsymbol{\theta}^{(t)} - \alpha \nabla \ell(h_{\boldsymbol{\theta}^{(t)}}(\mathbf{x}^{(i)}), y^{(i)}) \quad (8)$$

Solving linear Regression using Gradient Descent

- Randomly initialize θ
- To minimize the error, keep changing θ according to:

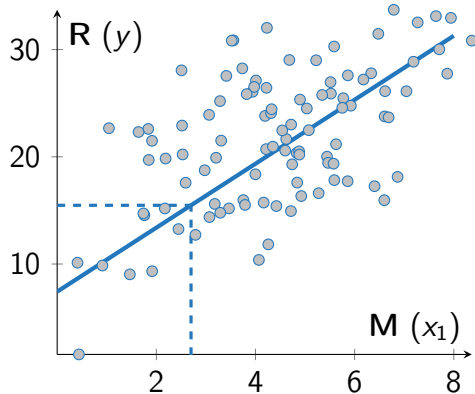
$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \alpha \nabla_{\theta} \mathcal{J}(\theta^{(t)}) \quad (9)$$

- We need to calculate $\nabla_{\theta_j} \mathcal{J}(\theta^{(t)})$: (based on a single example)

$$\frac{\partial}{\partial \theta_j} \mathcal{J}(\theta) = \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_{\theta}(\mathbf{x}) - y)^2 = 2 \cdot \frac{1}{2} (h_{\theta}(\mathbf{x}) - y) \cdot \frac{\partial}{\partial \theta_j} (h_{\theta}(\mathbf{x}) - y) \quad (10)$$

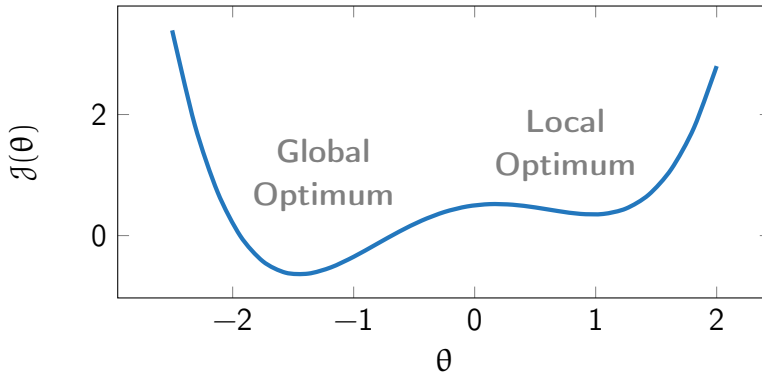
$$= (h_{\theta}(\mathbf{x}) - y) \cdot \frac{\partial}{\partial \theta_j} (\theta_0 x_0 + \dots + \theta_m x_m - y) = \boxed{(h_{\theta}(\mathbf{x}) - y) x_j} \quad (11)$$

Solving the introductory Example



- $\theta_0 \approx 7.4218$
- $\theta_1 \approx 2.9827$
- $J(\theta) \approx 446.9584$
- $h_{\theta}(\mathbf{x}) = 7.4218 + 2.9827 \cdot x_1$
- $R = h_{\theta}(2.7) = \underline{\underline{15.4750}}$

Disadvantage of Gradient Descent



Section: Basis Function Regression

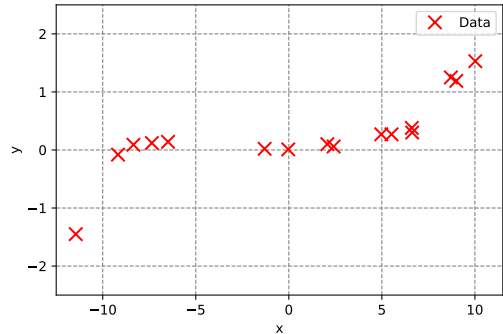
General Idea
Polynomial Basis Functions
Radial Basis Functions

What if the Data is non-linear?

- So far we have fitted straight lines
- **What if the data is not linear...?**

The best-fitting function is obviously **not a straight line**!

What would you do?



Basis Functions

- Remember: ‘When stuck switch to a different perspective’
- We can add **higher-order** features using **basis functions** φ :

We assume 1-D data

$$h_{\theta}(x) = \sum_{j=0}^p \theta_j \varphi_j(x) \quad (12)$$

- There exist several types of basis functions:
 - **linear**: $\varphi_0(x) = 1$ and $\varphi_1(x) = x$
 - **polynomial** \Rightarrow see below
 - **radial basis functions (RBFs)** \Rightarrow see below
 - **Fourier basis**

New Design Matrix

Applying the basis functions to \mathbf{X} we get the new design matrix Φ :

$$\Phi = \begin{pmatrix} \varphi_0(x^{(1)}) & \varphi_1(x^{(1)}) & \varphi_2(x^{(1)}) & \dots & \varphi_p(x^{(1)}) \\ \varphi_0(x^{(2)}) & \varphi_1(x^{(2)}) & \varphi_2(x^{(2)}) & \dots & \varphi_p(x^{(2)}) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \varphi_0(x^{(n)}) & \varphi_1(x^{(n)}) & \varphi_2(x^{(n)}) & \dots & \varphi_p(x^{(n)}) \end{pmatrix} \quad (13)$$

The model is still linear in the parameters, so we can still use the same algorithm as before. **This is still linear regression (!!!)**

Polynomial Basis Functions

- A quite frequently used basis function: The **polynomial basis**

$$\varphi_0(x) = 1$$

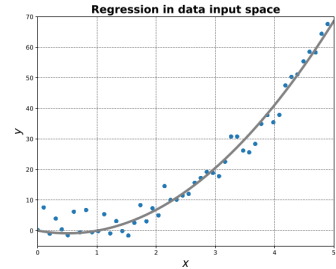
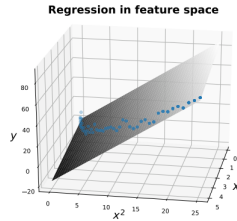
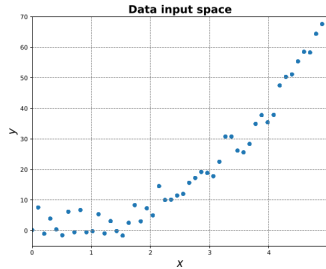
$$\varphi_j(x) = x^j$$

For N -D data we would also include cross-terms!

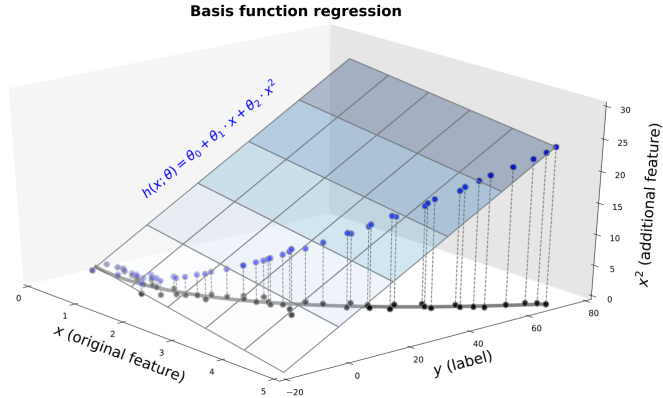
$$h_{\theta}(x) = \sum_{j=0}^p \theta_j \varphi_j(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \dots + \theta_p x^p$$

- Here, p is the degree of the polynomial
- Here: $\varphi(x) = [1, x, x^2, x^3, \dots, x^p]$

It is still linear!



It is still linear! (Ctd.)



Basis Functions: Radial Basis Functions

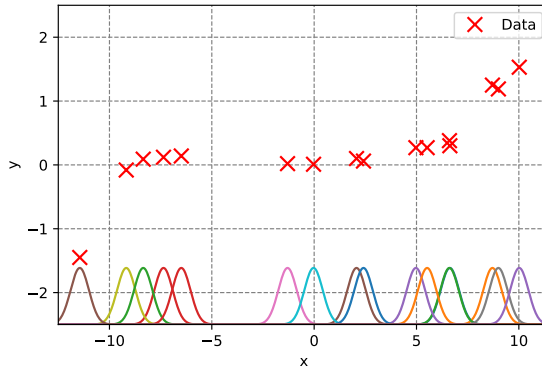
- Yet another possible choice of basis function: **Radial basis functions**

$$\varphi_0(x) = 1 \tag{14}$$

$$\varphi_j(x) = \exp \left\{ -1/2 \|x - z_j\|^2 / 2\sigma^2 \right\} \tag{15}$$

- $\{z_j\}$ are the centers of the radial basis functions
- p denotes the number of centers / number of radial basis functions
- Often we take each data point as a center, so $p = n$

Radial Basis Functions (Ctd.)



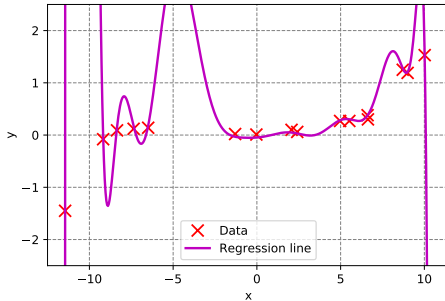
Section: Regularization Techniques

Underfitting and Overfitting
L1 and L2 Regularization

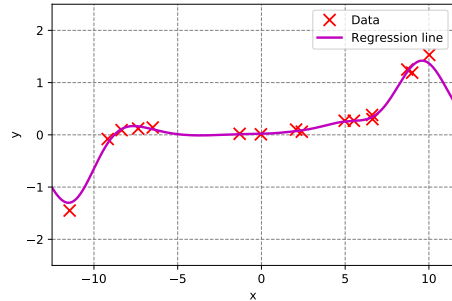


The Danger of too expressive Models...

Polynomial of degree $p = 16$
(☠️ **severe overfitting** ☠️)



RBF with $\sigma = 1.00$, $p = n$
(About right)



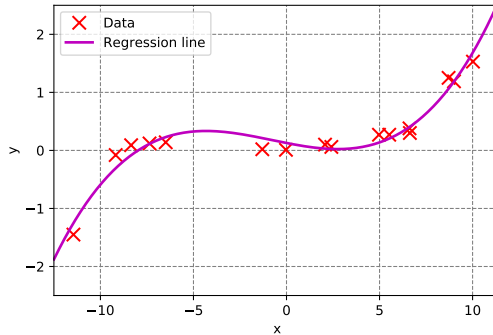
Overfitting vs. Underfitting

- **Underfitting**
 - The model is not complex enough to fit the data well \Rightarrow **High bias**
 - Make the model more complex; adding new examples **does not help**
- **Overfitting**
 - The model predicts the training data perfectly
 - But it **fails to generalize** to unseen instances \Rightarrow **High variance**
 - Decrease the degree of freedom or add more training examples
 - Also: Try **regularization**
- **Bias-Variance trade-off**

First Solution: Smaller Degree

One solution: Use a **smaller degree** (here: $p = 3$)

Much better :)



Second Solution: Regularization

- Enrich $\mathcal{J}(\boldsymbol{\theta})$ with a **regularization term**
- This can **prevent overfitting** and results in a smoother function (large values for θ_j are prevented)
- Two forms of regularization, **L1** and **L2**:

$$\min_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}) + \lambda |\boldsymbol{\theta}| \rightarrow (\text{L1})$$

$$\min_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|^2 \rightarrow (\text{L2})$$

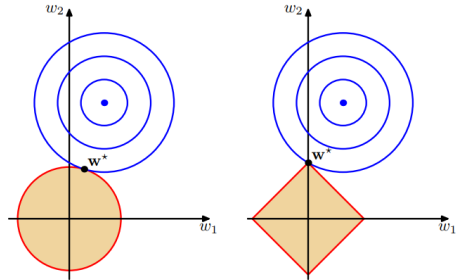
$$|\boldsymbol{\theta}| = \sum_{j=1}^m |\theta_j|$$

$$\|\boldsymbol{\theta}\|^2 = \sum_{j=1}^m \theta_j^2$$

- $\lambda \geq 0$ controls the **degree of regularization**

Regularization visualized

- Here: $\mathbf{w} \equiv \boldsymbol{\theta}$
- L1-Regularization
⇒ **Lasso regression**
(least abs. shrinkage and select. operator)
- L2-Regularization
⇒ **Ridge regression**
(Tikhonov regularization)
- The combination of both is called **elastic net**



cf. [?], p. 146; left: L2, right: L1

Incorporating Regularization

- Normal equation with regularization:
(ridge regression)

Regularization also
helps to overcome numerical issues!

$$\boldsymbol{\theta} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y} \quad (16)$$

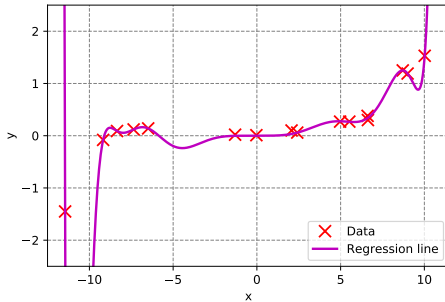
- Regularized gradient descent update rule:

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}^{(t)})$$

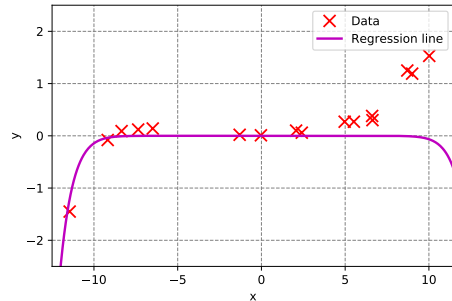
$$\frac{\partial}{\partial \theta_j} \mathcal{J}(\boldsymbol{\theta}) = (h_{\boldsymbol{\theta}}(\mathbf{x}) - y) x_j + \lambda \theta_j$$

Polynomial Regression with Regularization

At least better



Way too much regularization



Section: Wrap-Up

Summary
Self-Test Questions
Lecture Outlook

Summary

- Regression predicts **continuous target variables**
- The algorithm minimizes the **(mean) squared error**
- Minimizing the squared error gives the **maximum likelihood solution**
- **Two approaches:**
 - ① Normal equation
 - ② (Batch / stochastic / mini-batch) gradient descent
- Probabilistic regression allows to **quantify the uncertainty of the model**
- Use **basis functions** to fit non-linear regression lines
- **Regularization** is important



Self-Test Questions

- 1 What is the goal of regression?
- 2 What can you do if matrix inversion fails for the normal equation?
- 3 What is a suitable cost function for regression? Where does it come from?
- 4 Does gradient descent give the exact solution?
- 5 Which versions of gradient descent do you know?
- 6 What are basis functions? Why use them? State some examples.
- 7 What is overfitting and underfitting?
- 8 What is regularization? Why should you apply it?

What's next...?

Unit I	Machine Learning Introduction
Unit II	Mathematical Foundations
Unit III	Bayesian Decision Theory
Unit IV	Regression
Unit V	Classification I
Unit VI	Evaluation
Unit VII	Classification II
Unit VIII	Clustering
Unit IX	Dimensionality Reduction

Thank you very much for the attention!

Topic: *** Applied Machine Learning Fundamentals *** Regression

Term: Winter term 2023/2024

Contact:

Daniel Wehner, M.Sc.

SAP SE / DHBW Mannheim

daniel.wehner@sap.com

Do you have any questions?