* * * **Artificial Intelligence and Machine Learning** * * *

Logistic Regression and Softmax Regression

Daniel Wehner, M.Sc.

SAP SE / DHBW Mannheim

Summer term 2025

Find all slides on GitHub (DaWe1992/Applied_ML_Fundamentals)

Lecture Overview

I Machine Learning Introduction
II Optimization Techniques
III Bayesian Decision Theory
IV Non-parametric Density Estimation
V Probabilistic Graphical Models
VI Linear Regression
• VII Logistic Regression
VIII Deep Learning
IX Evaluation
X Decision Trees
XI Support Vector Machines
XII Clustering
XIII Principal Component Analysis
XIV Reinforcement Learning
XV Advanced Regression

Agenda for this Unit

1. Introduction
2. Model Architecture and Training
3. Non-linear Data
4. Multi-Class Classification
5. Wrap-Up

Section:

Introduction

What is logistic Regression?

Why you should not use linear Regression

What is logistic Regression?

- Logistic regression is a learning algorithm for **classification (!!!)**
- In its standard form it is applicable to **binary classification problems only**
- **Class labels:**
  - The 'positive class' $\oplus$ is encoded as **1**
  - The 'negative class' $\ominus$ as **0**
- **Probabilistic interpretation:** The raw output of the algorithm is between 0 and 1 and can be interpreted as *the probability of the instance belonging to the positive class*

Important

Why you should not use linear Regression...

- Linear regression:

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{\theta}^\top \boldsymbol{x}$$

- We can turn linear regression into a classifier by putting a **threshold** at $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = 0.5$ *(or any other value between 0 and 1)*:
  - If $h_{\boldsymbol{\theta}}(\boldsymbol{x}) \geqslant 0.5$, predict $\oplus$
  - If $h_{\boldsymbol{\theta}}(\boldsymbol{x}) < 0.5$, predict $\ominus$

Using linear regression in classification tasks has several downsides.

Can you imagine which ones?

Important

Why you should not use linear Regression... (Ctd.)

Problem 1: Outliers heavily affect the decision boundary *(see example below)*

Problem 2: Furthermore, we want the output of the model to be in the range $[0, 1]$ *(to allow for a probabilistic interpretation)*, i. e. $0 \leqslant h_{\theta}(\boldsymbol{x}) \leqslant 1$. However, **linear regression can output any value**, specifically

$$h_{\theta}(\boldsymbol{x}) \ll 0 \qquad \text{or} \qquad h_{\theta}(\boldsymbol{x}) \gg 1$$

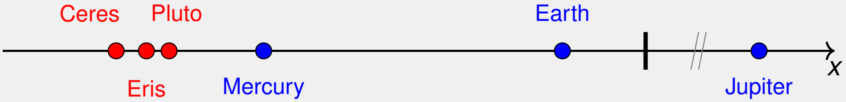
Why you should not use linear Regression... (Ctd.)

Consider the following dataset:

Row	**Object**	**Radius** ($\times 10^6$ m)	**Label**	**Label encoded**
1	Ceres	1.0	dwarf planet	0
2	Eris	2.3	dwarf planet	0
3	Pluto	2.4	dwarf planet	0
4	Mercury	4.9	planet	1
5	Earth	12.8	planet	1
6	Jupiter	143.0	planet	1

Why you should not use linear Regression... (Ctd.)

- Let us train a linear regression model for classification:



- Both, Mercury and Earth, are **classified as dwarf planets** due to Jupiter's massive radius!

Linear regression is sensitive to outliers! We need a better cost function!

Why you should not use linear Regression... (Ctd.)

- Logistic regression to the rescue:

- Logistic regression is less sensitive to outliers

(It is a valuable exercise to reproduce this result. See the exercise sheet!)

Section:

Model Architecture and Training

- Sigmoid / Logistic Function
- Probabilistic Interpretation of the Output
- Model Training
- Logistic Regression Decision Boundary

Logistic Regression Model

- Remember that we want: $0 \leqslant h_{\boldsymbol{\theta}}(\boldsymbol{x}) \leqslant 1$
- **Solution: Logistic function / Sigmoid function:**

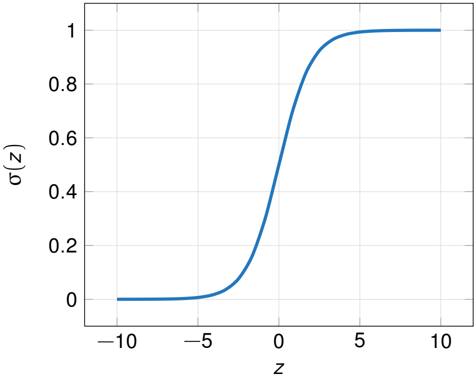
$$\sigma(z) := \frac{1}{1 + e^{-z}} \tag{1}$$

- We plug $\boldsymbol{\theta}^\top \boldsymbol{x}$ into the sigmoid function to obtain our new model function:

Logistic regression model function:

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) := \sigma(\boldsymbol{\theta}^\top \boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^\top \boldsymbol{x}}} \tag{2}$$

Logistic/Sigmoid Function



- $\sigma(z)$ is symmetric around $z = 0$
- $0 \leqslant \sigma(z) \leqslant 1$ holds true

Where does the Sigmoid come from?

$$p(\mathcal{C}_1|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\boldsymbol{x})} = \frac{p(\boldsymbol{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{\sum_{k=1}^{2} p(\boldsymbol{x}, \mathcal{C}_k)} = \frac{p(\boldsymbol{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{\sum_{k=1}^{2} p(\boldsymbol{x}|\mathcal{C}_k)p(\mathcal{C}_k)}$$

$$= \frac{p(\boldsymbol{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\boldsymbol{x}|\mathcal{C}_1)p(\mathcal{C}_1) + p(\boldsymbol{x}|\mathcal{C}_2)p(\mathcal{C}_2)}$$

$$= \frac{1}{1 + p(\boldsymbol{x}|\mathcal{C}_2)p(\mathcal{C}_2)/\big(p(\boldsymbol{x}|\mathcal{C}_1)p(\mathcal{C}_1)\big)}$$

$$= \frac{1}{1 + \exp\{-z\}} = \sigma(z) \qquad \longrightarrow \textbf{logistic sigmoid}$$

$$z := \log \frac{p(\boldsymbol{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\boldsymbol{x}|\mathcal{C}_2)p(\mathcal{C}_2)} \qquad \longrightarrow \textbf{log odds}$$

Interpretation of Hypothesis Output

- $h_{\boldsymbol{\theta}}(\boldsymbol{x})$ is interpreted as the probability of instance $\boldsymbol{x}$ belonging to class $\mathcal{C}_1$
- **Example:**

$$\boldsymbol{x} = \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = \begin{pmatrix} 1 \\ \texttt{tumorSize} \end{pmatrix} \tag{3}$$

- If $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = 0.7 = p(y = 1|\boldsymbol{x}; \boldsymbol{\theta})$, we have to tell the patient that there is a **70 % chance** of the tumor being malignant
- **Binary case:**

$$p(y = 0|\boldsymbol{x}; \boldsymbol{\theta}) = 1 - p(y = 1|\boldsymbol{x}; \boldsymbol{\theta}) = 0.3$$

Training Setup

- We have a labeled training set:

$$\mathcal{D} := \left\{\left(\boldsymbol{x}^1, y_1\right), \left(\boldsymbol{x}^2, y_2\right), \dots, \left(\boldsymbol{x}^N, y_N\right)\right\} = \left\{\left(\boldsymbol{x}^n, y_n\right)\right\}_{n=1}^{N} \tag{4}$$

- $y_n \in \{0, 1\}$ are the labels, and $\boldsymbol{x}^n$ are the feature vectors ($n = 1, 2, \dots, N$):

$$\boldsymbol{x}^n = \begin{pmatrix} x_0^{(n)} \\ x_1^{(n)} \\ \vdots \\ x_M^{(n)} \end{pmatrix} = \begin{pmatrix} 1 \\ x_1^{(n)} \\ \vdots \\ x_M^{(n)} \end{pmatrix} \in \mathbb{R}^{M+1} \tag{5}$$

Logistic Regression Cost Function

- We require a suitable cost function:

$$\mathfrak{J}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} \ell\left(h_{\boldsymbol{\theta}}(\boldsymbol{x}^n), y_n\right) \tag{6}$$

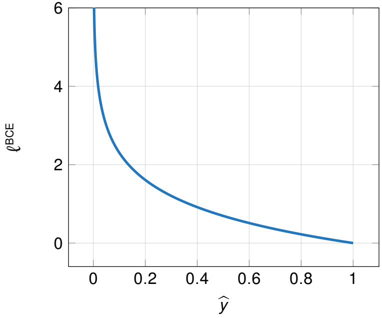
- For logistic regression, the cost function $\ell(\widehat{y}, y)$ is defined as follows:

  (square loss would be non-convex due to the sigmoid non-linearity...)

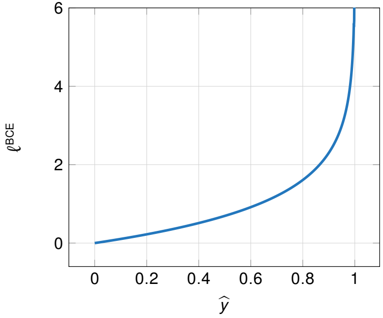
$$\ell^{\mathrm{BCE}}(\widehat{y}, y) := \begin{cases} -\log(\widehat{y}) & \text{if } y = 1 \\ -\log(1 - \widehat{y}) & \text{if } y = 0 \end{cases} \tag{7}$$

Logistic Regression Cost Function (Ctd.)

Case $y = 1$:



Case $y = 0$:



Logistic Regression Cost Function (Ctd.)

- ℓ^{BCE} can be written in a more compact form:

$$\ell^{\text{BCE}}(\widehat{y}, y) := -y \log(\widehat{y}) - (1 - y) \log(1 - \widehat{y}) \tag{8}$$

If $y = 1$, we get: $\qquad \ell^{\text{BCE}}(\widehat{y}, y) = -\log(\widehat{y})$

If $y = 0$, we get: $\qquad \ell^{\text{BCE}}(\widehat{y}, y) = -\log(1 - \widehat{y})$

- $y \in \{0, 1\}$ **acts as a switch** which activates the correct branch of the cost function

Logistic Regression Cost Function (Ctd.)

We use this result to define the

(Binary) cross entropy cost function:

$$\begin{aligned}\mathfrak{J}(\boldsymbol{\theta}) &\overset{(6)}{:=} \frac{1}{N}\sum_{n=1}^{N} \ell^{\mathrm{BCE}}\left(h_{\boldsymbol{\theta}}(\boldsymbol{x}^n), y_n\right) \\ &\overset{(8)}{=} \frac{1}{N}\sum_{n=1}^{N}\left[-y_n \log\left(h_{\boldsymbol{\theta}}(\boldsymbol{x}^n)\right) - (1-y_n)\log\left(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}^n)\right)\right] \qquad (9)\end{aligned}$$

Derivation of (binary) Cross Entropy using MLE

- The **likelihood function** for logistic regression can be written in the form:

$$p(\boldsymbol{y};\boldsymbol{\theta}) := \prod_{n=1}^{N} \left(h_{\boldsymbol{\theta}}(\boldsymbol{x}^n)\right)^{y_n} \cdot \left(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}^n)\right)^{1-y_n} \tag{10}$$

- The cost function is then given by the **negative log-likelihood**:

$$\mathfrak{J}(\boldsymbol{\theta}) = -\frac{1}{N}\log p(\boldsymbol{y};\boldsymbol{\theta}) = -\frac{1}{N}\mathcal{L}(\boldsymbol{\theta}) \tag{11}$$

Remark: We consider the **negative** log-likelihood because we prefer minimizing functions over maximizing them. Remember $\max\{f(x)\} = -\min\{-f(x)\}$.

Derivation of (binary) Cross Entropy using MLE (Ctd.)

$$
\begin{aligned}
\mathcal{L}(\boldsymbol{\theta}) &:= \log \left(\prod_{n=1}^{N} \left(h_{\boldsymbol{\theta}}(\boldsymbol{x}^n)\right)^{y_n} \cdot \left(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}^n)\right)^{1-y_n} \right) \\
&= \sum_{n=1}^{N} \log \left(\left(h_{\boldsymbol{\theta}}(\boldsymbol{x}^n)\right)^{y_n} \cdot \left(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}^n)\right)^{1-y_n} \right) \\
&= \sum_{n=1}^{N} \left[\log \left(\left(h_{\boldsymbol{\theta}}(\boldsymbol{x}^n)\right)^{y_n} \right) + \log \left(\left(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}^n)\right)^{1-y_n} \right) \right] \\
&= \sum_{n=1}^{N} \left[y_n \cdot \log\left(h_{\boldsymbol{\theta}}(\boldsymbol{x}^n)\right) + (1 - y_n) \cdot \log\left(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}^n)\right) \right]
\end{aligned}
$$

Remember the rules:

$$\log(ab) = \log a + \log b$$

$$\log(a^b) = b \log a$$

Can you see where we have used which rule?

Optimization of (binary) Cross Entropy

- Unfortunately, there is **no closed-form solution** to logistic regression
  (due to the sigmoid non-linearity in the model function)
- We have to resort to an iterative method like **gradient descent**

The partial derivative of $\ell^{\text{BCE}}\left(h_{\theta}(\boldsymbol{x}), y\right)$ (based on a single example) with respect to the m-th model parameter θ_m is given by:

$$\frac{\partial}{\partial \theta_m} \ell^{\text{BCE}}\left(h_{\theta}(\boldsymbol{x}), y\right) = \left(h_{\theta}(\boldsymbol{x}) - y\right) \cdot x_m \tag{12}$$

Important

Optimization of (binary) Cross Entropy (Ctd.)

The **stochastic gradient** of the binary cross entropy cost function is:

$$\nabla_{\boldsymbol{\theta}} \ell^{\text{BCE}}\left(h_{\boldsymbol{\theta}}(\boldsymbol{x}), y\right) = \left(\sigma\left(\boldsymbol{\theta}^{\top}\boldsymbol{x}\right) - y\right)\boldsymbol{x} \tag{13}$$

The **batch gradient** is given by the expression

$$\nabla_{\boldsymbol{\theta}} \mathfrak{J}(\boldsymbol{\theta}) = \frac{1}{N}\boldsymbol{X}^{\top}\left(\boldsymbol{\sigma}(\boldsymbol{X}\boldsymbol{\theta}) - \boldsymbol{y}\right), \tag{14}$$

where $\boldsymbol{\sigma}(\boldsymbol{z})$ applies the sigmoid function element-wise to the elements of $\boldsymbol{z}$

Derivation of the Gradient based on a single Example $(\boldsymbol{x}, y)$

- In the following we give a proof of equation (12)
- In the derivation we will need the derivative of the sigmoid function σ
- The derivative is given by:

$$\frac{\mathrm{d}}{\mathrm{d}z}\sigma(z) = \sigma(z) \cdot \big(1 - \sigma(z)\big) \tag{15}$$

  (You will be asked to prove this in the exercises!)

- Please find the full derivation of the partial derivative of ℓ^{BCE} $\Rightarrow$ here

Gradient Descent

- The goal is to minimize the cost function $\mathfrak{J}(\boldsymbol{\theta})$:

$$\boldsymbol{\theta}^\star = \arg\min_{\boldsymbol{\theta}} \mathfrak{J}(\boldsymbol{\theta})$$

- **Gradient descent:** Repeat until convergence:

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

- The gradient $\nabla_{\boldsymbol{\theta}} \mathfrak{J}(\boldsymbol{\theta})$ is given by equation (14)

The algorithm looks identical to linear regression, but the model function is different due to the sigmoid function!

Decision Boundary

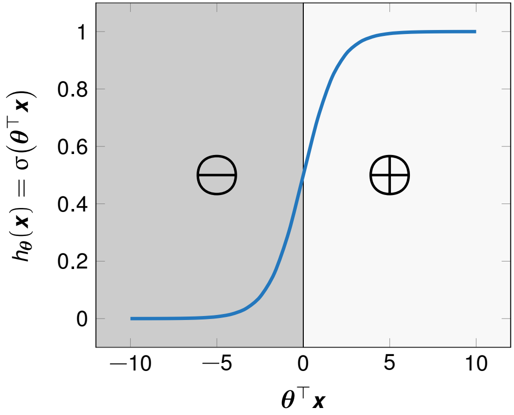
- Our trained model **outputs probabilities**
- To obtain a classifier, we have to **apply a threshold** ρ to the raw outputs
- Setting the threshold to $\rho := 0.5$ means:
  - Predict the positive class $\oplus$, if

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) \geqslant 0.5 \quad \Longleftrightarrow \quad \boldsymbol{\theta}^\top \boldsymbol{x} \geqslant 0$$

  - Predict the negative class $\ominus$, if

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) < 0.5 \quad \Longleftrightarrow \quad \boldsymbol{\theta}^\top \boldsymbol{x} < 0$$

Decision Boundary (Ctd.)



Example: Decision Boundary

- Let us consider a simple example
- Suppose our model function takes the form:

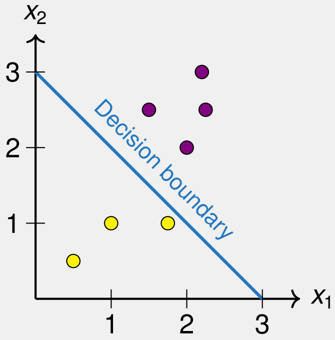
$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma\left(\boldsymbol{\theta}^\top \boldsymbol{x}\right) = \sigma(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

- Assume we obtain the following model parameters using gradient descent:

$$\theta_0 = -3, \qquad \theta_1 = 1, \qquad \theta_2 = 1$$

- Then predict $\oplus$, if $-3 + x_1 + x_2 \geqslant 0$

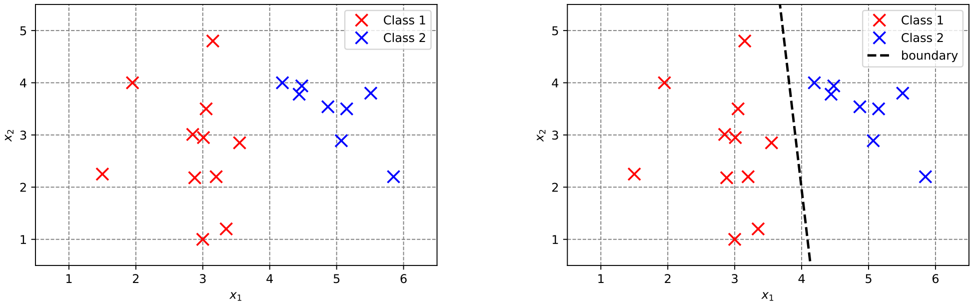
Example: Decision Boundary (Ctd.)



- Predict $\oplus$, if $-3 + x_1 + x_2 \geqslant 0$
- The decision boundary satisfies $-3 + x_1 + x_2 = 0$
- If $x_2 = 0$, then $x_1 = 3$, and vice versa

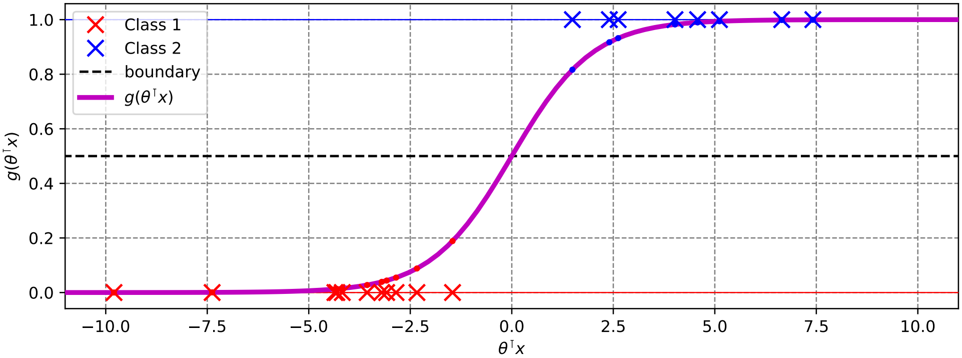
> Logistic regression is not a **maximum-margin classifier**, but the cost function can be adjusted to get that (see **Hinge loss**)

Another Example: Decision Boundary



Question: Where is the sigmoid function?

Another Example: Logistic Function



Comparison of logistic Regression and GDA

- Logistic regression:
  - **Makes weaker assumptions about the data**
  - **Works better than GDA, if the data is not GAUSSian**
  - **Requires a larger dataset**
  - **Choice of the learning rate is necessary**
- GAUSSian Discriminant Analysis (GDA):
  - **Requires less data**
  - **No hyperparameters**
  - **Works better if the data is GAUSSian**
  - **Stronger modeling assumptions: Assumes data to be GAUSSian**

Section:

Non-linear Data

Feature Mapping
Regularization

Non-Linear Decision Boundaries

Again, we can use **feature mapping** to obtain non-linear decision boundaries/surfaces

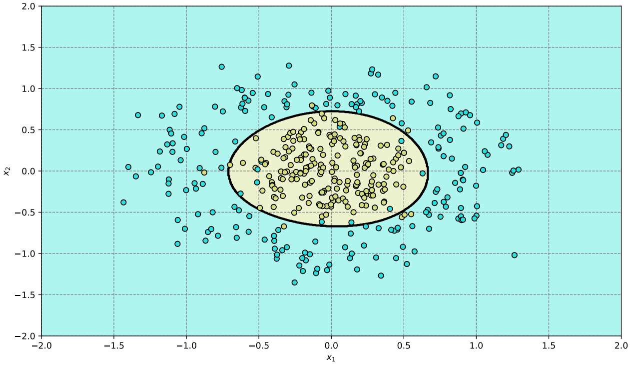
Example:

- Let a circular two-dimensional dataset be given (features x_1 and x_2)
- Assume we choose the following model function (we add the squares of x_1 and x_2):

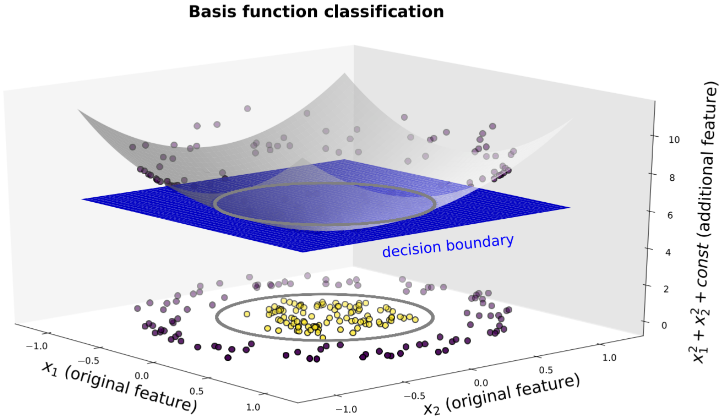
$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma\left(\boldsymbol{\theta}^\top \boldsymbol{x}\right) = \sigma\left(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2\right)$$

- The algorithm could choose the parameters $\boldsymbol{\theta}^\star := \left(-1, 0, 0, 1, 1\right)^\top$
- So we would get: $x_1^2 + x_2^2 = 1$ **(equation of a unit circle)**

Example: Non-Linear Decision Boundary



It is still linear!



Logistic Regression with Regularization

- Again, we should apply **regularization** when using the feature mapping approach to avoid running into ☠ **overfitting** ☠
- We add a **regularizer** to the cost function:

$$\widetilde{\mathfrak{J}}(\boldsymbol{\theta}) := \mathfrak{J}(\boldsymbol{\theta}) + \lambda ||\boldsymbol{\theta}||^2 \tag{16}$$

- The regularizer prevents the parameters θ_m from becoming too large
- $\lambda \geqslant 0$ controls the degree of regularization
- This leads to smoother decision boundaries

Section:

Multi-Class Classification

- Techniques Overview
- Multinomial Logistic Regression / Softmax Regression
- One-vs-Rest (OvR)
- One-vs-One (OvO)

Multi-Class Classification

In its basic form, logistic regression can handle two classes only!

Question: What if there are more than two classes?

Two conceivable approaches:

1. Change the algorithm so that it can deal with the non-binary case **(Multinomial Logistic Regression / Softmax Regression)**
2. Transform the problem into several binary problems, e. g. by using the techniques
   - **One-vs-Rest (OvR)**
   - **One-vs-One (OvO)**

Softmax Regression

- Again we consider a labeled dataset:

$$\mathcal{D} := \left\{ \left(\boldsymbol{x}^n, y_n \right) \right\}_{n=1}^{N}$$

- However, the labels can take one of K values: $y_n \in \{1, 2, \ldots, K\}$
- We apply **one-hot encoding** to the labels y_n to obtain $\boldsymbol{y}^n \in \mathbb{R}^K$, $n = 1, 2, \ldots, N$

One-hot encoding: The label of a record labeled with class k $(1 \leqslant k \leqslant K)$ is represented by a K-dimensional one-hot vector, i. e. all vector components are set to zero, except for the k-th component which is set to one.

Example: One-hot Encoding

- A softmax classifier is trained to detect dogs, cats, and mice in images
- Each image in the training set is labeled accordingly, i. e.

$$y_n \in \left\{1 \ (dog), 2 \ (cat), 3 \ (mouse)\right\}$$

- The resulting one-hot vectors $\boldsymbol{y}^n$ are three-dimensional as we have three distinct classes:

$$\text{dog} := \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \qquad \text{cat} := \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \qquad \text{mouse} := \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Softmax Regression: Model Architecture

- Due to the one-hot encoding it is no longer sufficient to produce a single scalar model output
- Insted, the model has to produce a K dimensional output *(i. e. a vector)*
- The k-th component of the output vector should **reflect the probability** of the unknown instance belonging to the k-th class, i. e. the sum of the components should be equal to 1

Goal: We want to interpret the model output as a **probability distribution** over the possible class labels!

Softmax Regression: Model Architecture (Ctd.)

- In our new model we replace the sigmoid function σ with the **softmax function** $\boldsymbol{\zeta}$:

$$\boldsymbol{\zeta} : \mathbb{R}^K \to \mathbb{R}^K, \qquad \boldsymbol{z} \mapsto \boldsymbol{\zeta}(\boldsymbol{z}), \qquad \zeta_k(\boldsymbol{z}) := \frac{e^{z_k}}{\sum_{j=1}^K e^{z_j}} \tag{17}$$

- The softmax function $\boldsymbol{\zeta}$ is a **vector-valued** function *(bold face notation!)*
- $\zeta_k(\boldsymbol{z})$ denotes the k-th component of the output vector
- Per construction we have

$$\sum_{k=1}^K \zeta_k(\boldsymbol{z}) = 1 \tag{18}$$

Softmax Regression: Model Architecture (Ctd.)

- In softmax regression, the input vector $\boldsymbol{z}$ to the softmax function is given by the **vector of logits**:

$$\boldsymbol{z} := \begin{pmatrix} \boldsymbol{x}^\top \boldsymbol{\theta}^1 & \boldsymbol{x}^\top \boldsymbol{\theta}^2 & \dots & \boldsymbol{x}^\top \boldsymbol{\theta}^K \end{pmatrix}^\top \in \mathbb{R}^K \tag{19}$$

- For each of the K classes we maintain a dedicated set of adjustable parameters $\boldsymbol{\theta}^k \in \mathbb{R}^{M+1}$
- **Important:** All parameters are trained **jointly**, i. e.
  - when the optimization algorithm **increases the probability** for one class...
  - ...it simultaneously **decreases the probabilities** for all other classes

Softmax Regression: Model Architecture (Ctd.)

- Let $\boldsymbol{\Theta} := \left[\boldsymbol{\theta}^1, \boldsymbol{\theta}^2, \dots, \boldsymbol{\theta}^K\right] \in \mathbb{R}^{(M+1)\times K}$ be the matrix of all model parameters *(where the vectors $\boldsymbol{\theta}^k$ are stacked column-wise)*
- The final model function for softmax regression takes the form

Model function for softmax regression:

$$\boldsymbol{h}_{\boldsymbol{\Theta}}(\boldsymbol{x}) := \zeta(\boldsymbol{z}) = \frac{1}{\sum_{j=1}^{K} \exp\left(\boldsymbol{x}^\top \boldsymbol{\theta}^j\right)} \begin{pmatrix} \exp\left(\boldsymbol{x}^\top \boldsymbol{\theta}^1\right) \\ \vdots \\ \exp\left(\boldsymbol{x}^\top \boldsymbol{\theta}^K\right) \end{pmatrix} \tag{20}$$

Categorical Cross Entropy

Categorical cross entropy cost function: *(BCE can handle two classes only)*

$$\mathfrak{J}(\boldsymbol{\Theta}) := \frac{1}{N} \sum_{n=1}^{N} \ell^{\mathrm{CE}}\left(\boldsymbol{h}_{\boldsymbol{\Theta}}(\boldsymbol{x}^n), \boldsymbol{y}^n\right) \tag{21}$$

$$\ell^{\mathrm{CE}}\left(\widehat{\boldsymbol{y}}, \boldsymbol{y}\right) := -\sum_{k=1}^{K} y_k \log(\widehat{y}_k), \qquad \widehat{y}_k = \zeta_k(\boldsymbol{z}) \tag{22}$$

Remark: For $K = 2$ equation (22) is equivalent to equation (9)

Optimization of Categorical Cross Entropy

The partial derivative

$$\frac{\partial}{\partial \theta_{ij}} \ell^{\mathrm{CE}}\left(\boldsymbol{h}_{\boldsymbol{\Theta}}(\boldsymbol{x}), \boldsymbol{y}\right)$$

of the categorical cross entropy cost function (22) with respect to the i-th model parameter connected to the j-th class θ_{ij} is given by:

$$\frac{\partial}{\partial \theta_{ij}} \ell^{\mathrm{CE}}\left(\boldsymbol{h}_{\boldsymbol{\Theta}}(\boldsymbol{x}), \boldsymbol{y}\right) = \left(\zeta_j(\boldsymbol{z}) - y_j\right) \cdot x_i \tag{23}$$

Remark: Equation (23) resembles equation (12)

Optimization of Categorical Cross Entropy (Ctd.)

Batch gradient for softmax regression:

$$\nabla_{\boldsymbol{\Theta}} \mathfrak{J}(\boldsymbol{\Theta}) = \frac{1}{N} \boldsymbol{X}^\top \left(\zeta(\boldsymbol{X}\boldsymbol{\Theta}) - \boldsymbol{Y} \right) \tag{24}$$

Remarks:

- $\boldsymbol{Y} \in \mathbb{R}^{N \times K}$ is the one-hot label matrix
- $\zeta(\boldsymbol{X}\boldsymbol{\Theta}) \in \mathbb{R}^{N \times K}$ produces a matrix (raw predictions) of the same shape as $\boldsymbol{Y}$
- We have to predict the column index of the largest entry per row to retrieve the predicted class labels

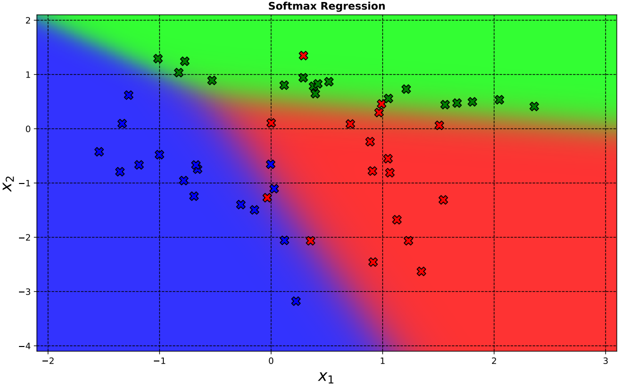
Derivation of the Gradient based on a single Example $(\boldsymbol{x}, \boldsymbol{y})$

- In the following we will prove equation (23)
- In the derivation we will need the derivative of the softmax function which is given by *(see exercise sheet)*:

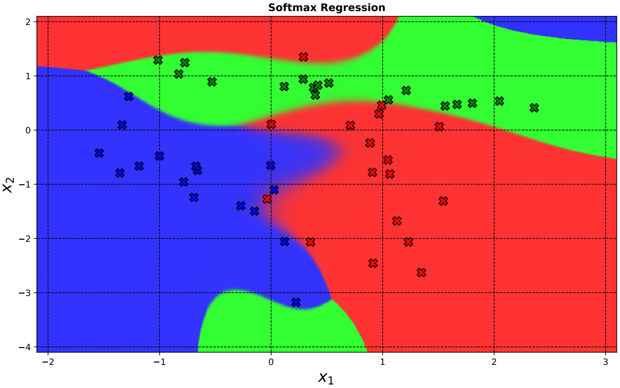
$$\frac{\partial}{\partial z_j}\zeta_k(\boldsymbol{z}) = \begin{cases} \zeta_j(\boldsymbol{z}) \cdot (1 - \zeta_j(\boldsymbol{z})) & \text{if } k = j \\ -\zeta_k(\boldsymbol{z}) \cdot \zeta_j(\boldsymbol{z}) & \text{if } k \neq j \end{cases} \tag{25}$$

- Please find the full derivation of equation (23) $\Rightarrow$ here

Example: Softmax Regression



Example: Softmax Regression with Basis Functions (Overfitting!)



Numerical Aspects of the Softmax Function

- Implementing the softmax function according to equation (17) might result in numerical overflows (NaN – Not a Number)
- We notice that for any constant $C \in \mathbb{R}$ we have:

$$\begin{aligned}\zeta_k(\boldsymbol{z}) := \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}} &= \frac{C}{C}\,\frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}} = \frac{Ce^{z_k}}{\sum_{j=1}^{K} Ce^{z_j}} \\ &= \frac{e^{z_k + \log C}}{\sum_{j=1}^{K} e^{z_j + \log C}}\end{aligned} \tag{26}$$

Numerical Aspects of the Softmax Function (Ctd.)

- Equation (26) tells us that we can **offset the inputs by any constant** of our choice without changing the output of the softmax function ζ
- We choose $C := -\max_{k=1,\dots,K}(z_k)$ and obtain

$$\zeta_k(\boldsymbol{z}) = \frac{e^{z_k - \max_k(z_k)}}{\sum_{j=1}^{K} e^{z_j - \max_k(z_k)}} \tag{27}$$

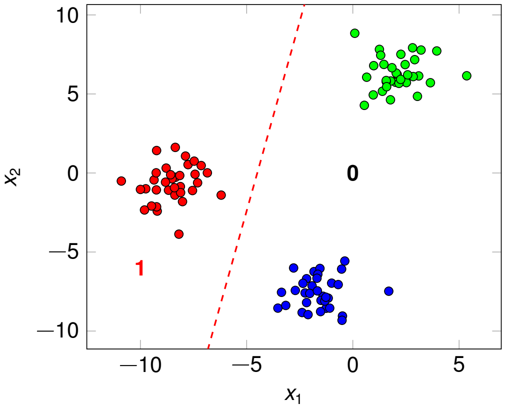
We get a **numerically stable** version of softmax! All exponentiated values will be between 0 and 1 **(no overflows!)**, and the denominator will be $\geqslant 1$, because at least one exponentiated value is equal to 1 **(no division by zero!)**.

Transforming the Problem into several binary Problems

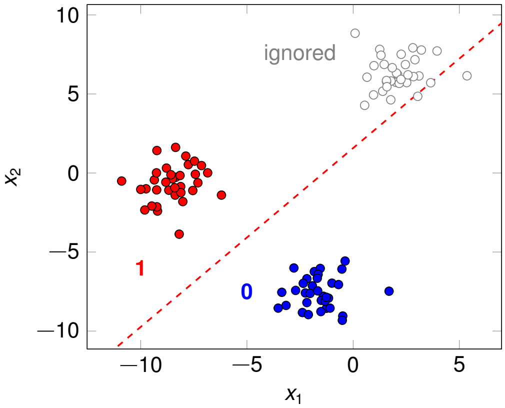
- Instead of adjusting the algorithm, we can also **transform the multi-class problem into several binary problems**
- Two common techniques are:
  - **One-vs-Rest (OvR)**
  - **One-vs-One (OvO)**
- **General idea:**
  - Several classifiers are trained individually
  - During prediction the classifiers **vote for the correct class**
- Such techniques can be used **for all binary classifiers**

Multi-Class Classification: One-vs-Rest (OvR)

- Also known as **One-against-All**
- **Train one classifier per class** (expert for that class)
- We get K classifiers
- The k-th classifier learns to distinguish the k-th class from all the others
- Set the labels of examples from class k to 1, all the others to 0



Multi-Class Classification: One-vs-One (OvO)



- **Train one classifier for each pair of classes** (pairwise classification)
- We get $\binom{K}{2}$ classifiers
- Ignore all other examples that do not belong to either of the two classes
- **Voting**: Count how often each class wins
- The class with the highest score is predicted

Section:

Wrap-Up

- Summary
- Recommended Literature
- Self-Test Questions
- Lecture Outlook

Summary

- **Logistic regression is used for classification (!!!)**
- It is used for **binary classification problems** (generalizations exist)
- **Output:** Probability of instance belonging to positive class
- Apply a **threshold** ρ to get the classification
- The algorithm minimizes the **cross entropy cost function**
- There is **no closed-form solution** (unlike for linear regression)
- **Basis functions** can be used for non-linear data
- **Multi-class classification:** Softmax regression, One-vs-Rest, One-vs-One

Recommended Literature

1. [BISHOP.2006], chapter 4
2. [MURPHY.2012], chapter 8

(For free PDF versions, see list in GitHub readme!)

Self-Test Questions

1. Why should you not use linear regression for classification?
2. How is the logistic function defined?
3. Why do we use cross entropy instead of the squared error?
4. Does logistic regression find the maximum margin hyperplane?
5. When should you use logistic regression, when GDA?
6. What techniques do you know for multi-class classification problems?
7. How does softmax regression differ from One-vs-Rest?
8. How can you avoid numerical issues when computing the softmax?

What's next...?

I Machine Learning Introduction
II Optimization Techniques
III Bayesian Decision Theory
IV Non-parametric Density Estimation
V Probabilistic Graphical Models
VI Linear Regression
VII Logistic Regression
• VIII Deep Learning
IX Evaluation
X Decision Trees
XI Support Vector Machines
XII Clustering
XIII Principal Component Analysis
XIV Reinforcement Learning
XV Advanced Regression

Thank you very much for the attention!

*** **Artificial Intelligence and Machine Learning** ***

Topic: Logistic Regression and Softmax Regression

Term: Summer term 2025

Contact:

Daniel Wehner, M.Sc.

SAP SE / DHBW Mannheim

daniel.wehner@sap.com

Do you have any questions?