Martin Mundt, Dr. Iuliia Pliushch, Prof. Dr. Visvanathan Ramesh

Pattern Analysis & Machine Intelligence Praktikum: MLPR WS-19/20

Week 2: Regression and Gradient Descent

Lecture requirements

- 4 hours per week, 8 credit points -> attend the class!
- Final class group projects. Will start at the end of the lecture period with presentations at the end of the semester (we will choose a date together later).

- Bring a laptop with WiFi access to the university network (Flughafen, eduroam etc.).
- Each session is going to start with a roughly 30-60 minute lecture.
- Online notebooks will be used for the 3 hour long practical sessions.
- We will use Google's Colaboratory for cloud computation. If you have a Gmail, android etc. account, then it is the same account you already have.
- If you don't want to use Colab (because you don't want a google account), you can execute the notebooks locally, we advise a Linux or Mac OS (and will provide no Windows support). Please be reminded that some of the later deep learning content will be difficult to setup/run on your local laptops (unless they have a GPU) as we will use Google Colab's free GPU instances.

- **All lecture materials will be shared on GitHub: https://github.com/ccc-frankfurt/Practical_ML_WS19**

Schedule

Introduction:

Week 1: 14.10 – Introduction, python tools review, software management (version-control & documentation)

Week 2: 21.10 – Ideas behind ML, gradient descent on functions, logistic regression -> Kaggle Titanic dataset

Block 1 - Supervised Learning:

Week 3: 28.10 – Random forests from scratch -> Revisiting Titanic

Week 4: 04.11 – Random forests application, intro to sklearn -> San Francisco crime challenge

Week 5: 11.11 – Basic neural networks from scratch -> Multi-layer perceptron for classification of fashion images

Week 6: 18.11 – Introduction to PyTorch, deep learning, convolutional neural networks -> Reading traditional Japanese characters (Kuzushiji)

Week 7: 25.11 – Neural sequence models, recurrent neural networks -> Shakespeare poetry text generation

Schedule

Block 2 - Unsupervised Learning:

Week 08: 02.12 – Unsupervised learning: k-means clustering and principal component analysis -> self-generated known data distributions

Week 09: 09.12 – Unsupervised neural networks, autoencoders (representation learning/unsupervised pre-training) -> Revisiting fashion and Kuzushiji images

Week 10: 16.12 – Generative models: variational autoencoders -> handwriting generation

Week 11: 13.01 – Generative models: generative adversarial networks -> Face generation

Block 3 – Reinforcement Learning

Week 12: 20.01 – Classic Q-learning -> Q-learning

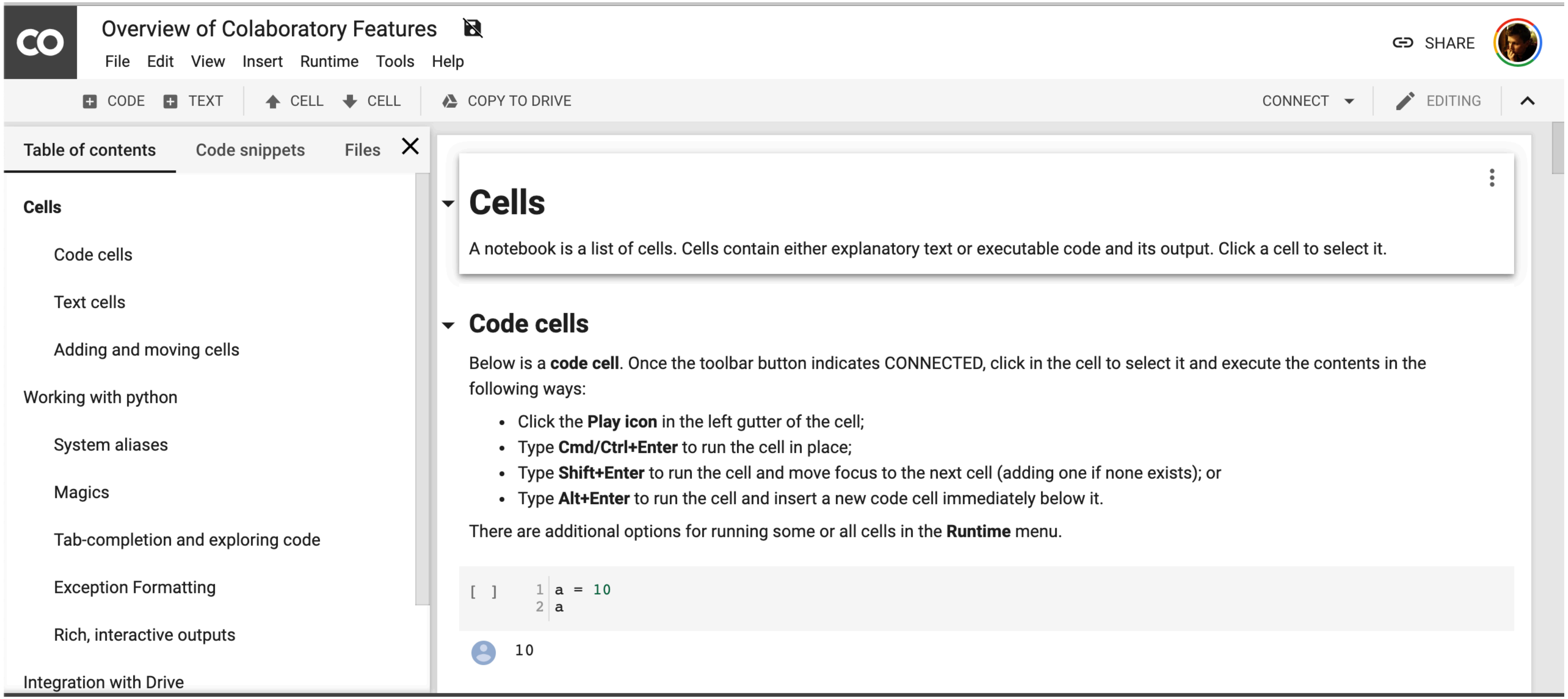
Week 13: 27.01 – Deep reinforcement learning, QNN -> Taxi driver

Week 14: 03.02 – Reinforce algorithm -> Robotic application (walking/grasping)

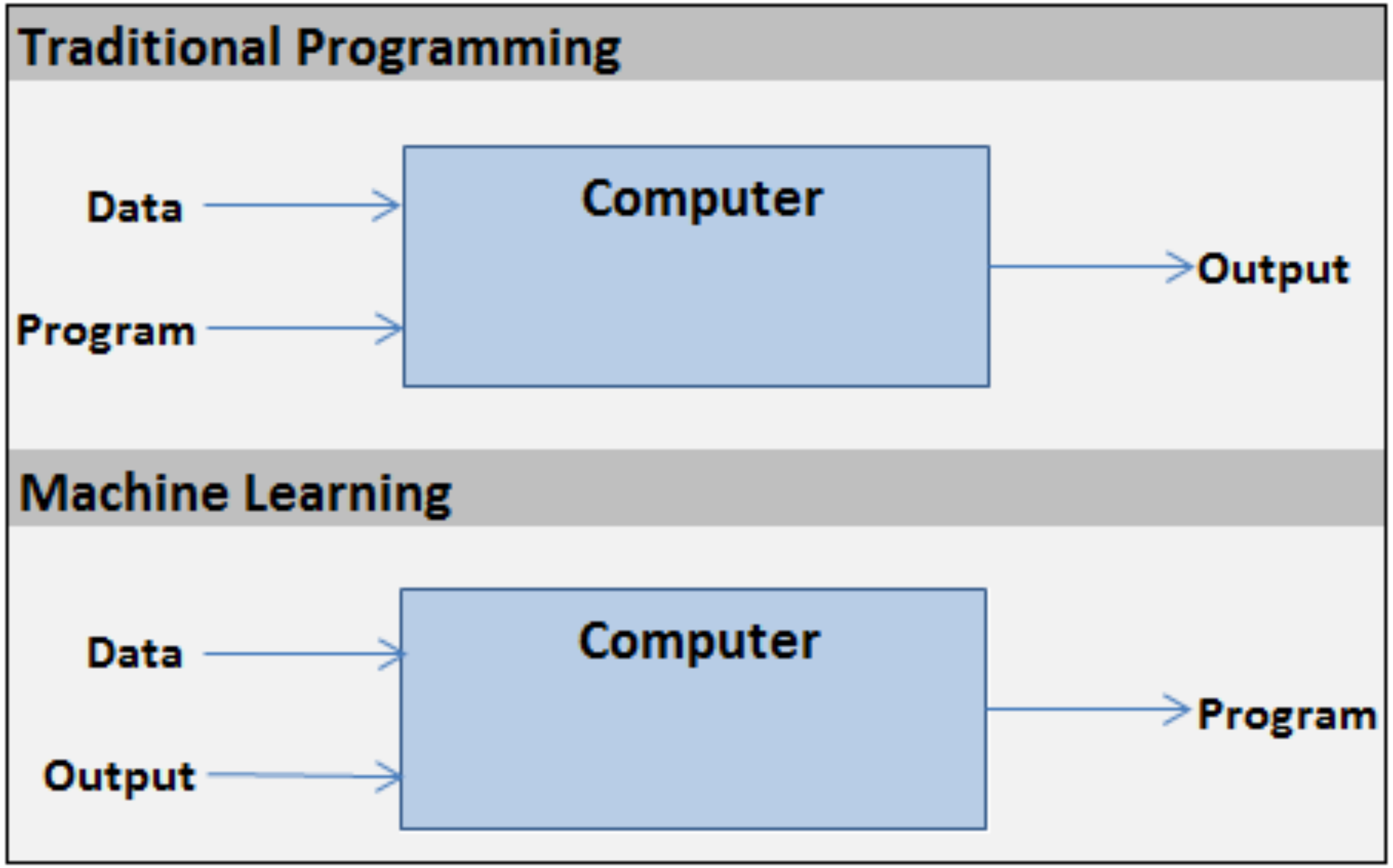
Project planning:

Week 15: 10.02 – State-of-the-art, open questions and existing issues -> Project pitches (3 slides per person)

Reminder: Google Colab: https://colab.research.google.com/



What is Machine Learning?



- Train parameters by optimizing a cost function
- Generalize

Machine Learning

Supervised: Target outputs (labelled) are known and can be used for training

* Regression, Classification

Unsupervised: Target outputs not defined or unavailable

* Density estimation, clustering

Reinforcement: Agent learns by being rewarded for certain actions

* Robotics, playing games

Linear Regression

- Expresses a linear relationship between several independent **x** and a dependent variable **y** (labeled target)

$$t = w_0 + \sum_{i=1}^{D} w_i x_i + \varepsilon$$

- Regression weights **w** are parameters of the model linking inputs **x** with targets **t**

- In machine learning terminology linear regression is a supervised learning algorithm with continuous targets, i.e. it assumes a labeled dataset **D** with N training pairs of inputs and targets.

Linear Regression: how do we find the model weights?

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

- Need a way to measure the performance of the linear model with different weights

$$MSE_D(w) = \frac{1}{2}\sum_{n=1}^{N}(w_0 + \sum_{i} w_i x_{ni} - t_n)^2$$

- In ML this is referred to as a cost function and it measures the error of the model with respect to the weights **w.** (In this case the mean-squared difference between prediction and target)

- The optimal regression weights can be found by minimizing this error

$$w_{LR} = argmin_w MSE_D(w) = argmin_w \frac{1}{2}\sum_{n=1}^{N}(w_0 + \sum_{i} w_i x_{ni} - t_n)^2$$

Solving linear regression

- We can express linear regression in matrix notation if we extend $x_0 \equiv 1$ and denote the transposed weight vector as $\boldsymbol{w}^T$

$$t = \boldsymbol{w}^T \boldsymbol{x} + \varepsilon$$

- The minimization problem can then be written as

$$w_{LR} = argmin_w \frac{1}{2} \sum_{n=1}^{N} (t_n - \boldsymbol{w}^T \boldsymbol{x}_n)^2$$

Linear regression in practice

- Let’s take a look at practical linear regression in

  https://github.com/PyMLVizard/PyMLViz

- PyMLViz is a project together with Prof. Dr. Bertschinger that features interactive widgets to explore different basic machine learning algorithms inside the browser.

- We can extend linear regression to more complicated basis functions, e.g. polynomial or radial basis functions. This is further explained in the notebook we will go through.

- If you haven‘t been able to attend the practice please take a look at the linear regression notebook at above URL, including polynomial and radial basis functions.

Logistic Regression

- Actually a technique for classification, i.e. $Y = \{0, 1\}$, with a probabilistic viewpoint.
- Goal is to correctly classify input x into one of two classes
- We could technically also do classification with the linear regression model we have just learned and treat the output as class 1 if the prediction surpasses a threshold and class 0 otherwise.
- In logistic regression we would like to interpret our model's decisions based on class probabilities $P(y = 1|x)$

Logistic Regression

- For binary classification we can write decisions based on the log odds.
  * If neither outcome is favored => log odds = 0
  * If one outcome is favored by log odds = x, the other is disfavored with –x

- We learn the log odds of P(y = 1|x) as a linear function:

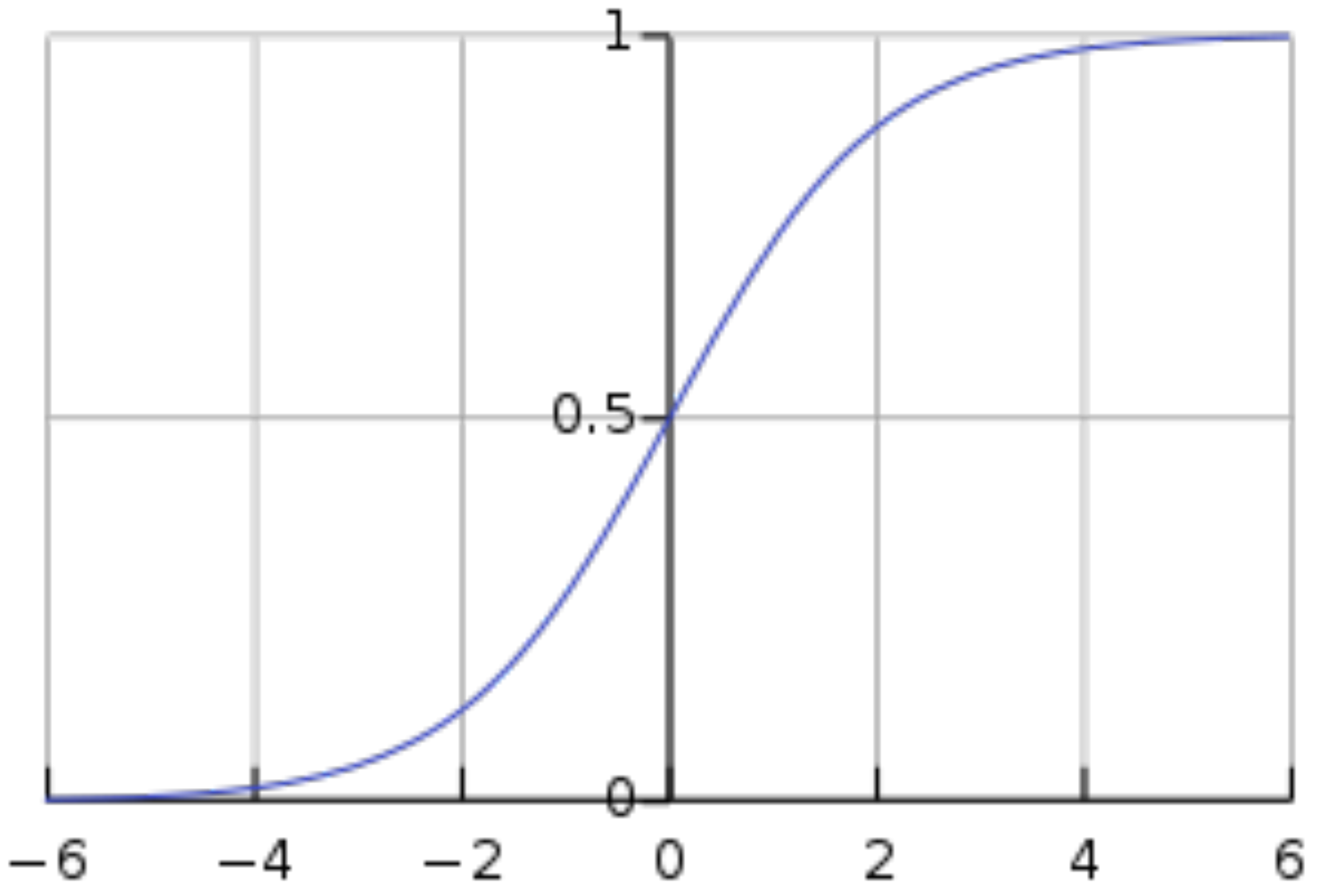
$$\log \frac{P(y=1|\boldsymbol{x})}{P(y=0|\boldsymbol{x})} = w_0 + \boldsymbol{x}^T \boldsymbol{w}_1$$

- Probabilities <-> log odds

$$z = log \frac{p}{1-p} \qquad e^z = \frac{p}{1-p} \qquad p = \frac{1}{1+e^{-z}} \; (logistic\ function)$$

Logistic Regression

$$\Rightarrow P(y = 1|\boldsymbol{x}, \boldsymbol{w}) = \frac{1}{1 + e^{-(w_0 + \boldsymbol{x}^T \boldsymbol{w}_1)}}$$



https://en.wikipedia.org/wiki/Sigmoid_function

- Again, we can find the maximum log likelihood by setting the derivatives of the log-likelihood with respect to the weights (parameters) to zero

$$\frac{\partial}{\partial w_i} \frac{1}{N} \sum_{n=1}^{N} \left[y_n \log\big(P(y_n = 1|x_n, w) + (1 - y_n) \log\big(1 - P(y_n = 1|x_n, w)\big)\right] = 0$$

- We need to derive above expression (apply the chain rule multiple times). Because above expression isn't closed form we have to solve it numerically by maximizing the likelihood or using e.g. gradient descent to minimize the negative thereof. This cost function is typically referred to as the Cross Entropy loss.

Logistic Regression cost function derivation

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{\partial}{\partial \theta_j} \frac{-1}{m} \sum_{i=1}^m \left[y^{(i)} \log\left(h_\theta\left(x^{(i)}\right)\right) + (1-y^{(i)}) \log\left(1-h_\theta\left(x^{(i)}\right)\right)\right]$$

$$\underset{\text{linearity}}{=} \frac{-1}{m} \sum_{i=1}^m \left[y^{(i)} \frac{\partial}{\partial \theta_j} \log\left(h_\theta\left(x^{(i)}\right)\right) + (1-y^{(i)}) \frac{\partial}{\partial \theta_j} \log\left(1-h_\theta\left(x^{(i)}\right)\right)\right]$$

$$\underset{\text{chain rule}}{=} \frac{-1}{m} \sum_{i=1}^m \left[y^{(i)} \frac{\frac{\partial}{\partial \theta_j} h_\theta\left(x^{(i)}\right)}{h_\theta\left(x^{(i)}\right)} + (1-y^{(i)}) \frac{\frac{\partial}{\partial \theta_j} \left(1-h_\theta\left(x^{(i)}\right)\right)}{1-h_\theta\left(x^{(i)}\right)}\right]$$

$$\underset{h_\theta(x)=\sigma(\theta^\top x)}{=} \frac{-1}{m} \sum_{i=1}^m \left[y^{(i)} \frac{\frac{\partial}{\partial \theta_j} \sigma\left(\theta^\top x^{(i)}\right)}{h_\theta\left(x^{(i)}\right)} + (1-y^{(i)}) \frac{\frac{\partial}{\partial \theta_j} \left(1-\sigma\left(\theta^\top x^{(i)}\right)\right)}{1-h_\theta\left(x^{(i)}\right)}\right]$$

$$\underset{\sigma'}{=} \frac{-1}{m} \sum_{i=1}^m \left[y^{(i)} \frac{\sigma\left(\theta^\top x^{(i)}\right)\left(1-\sigma\left(\theta^\top x^{(i)}\right)\right) \frac{\partial}{\partial \theta_j}\left(\theta^\top x^{(i)}\right)}{h_\theta\left(x^{(i)}\right)} - (1-y^{(i)}) \frac{\sigma\left(\theta^\top x^{(i)}\right)\left(1-\sigma\left(\theta^\top x^{(i)}\right)\right) \frac{\partial}{\partial \theta_j}\left(\theta^\top x^{(i)}\right)}{1-h_\theta\left(x^{(i)}\right)}\right]$$

$$\underset{\sigma(\theta^\top x)=h_\theta(x)}{=} \frac{-1}{m} \sum_{i=1}^m \left[y^{(i)} \frac{h_\theta\left(x^{(i)}\right)\left(1-h_\theta\left(x^{(i)}\right)\right) \frac{\partial}{\partial \theta_j}\left(\theta^\top x^{(i)}\right)}{h_\theta\left(x^{(i)}\right)} - (1-y^{(i)}) \frac{h_\theta\left(x^{(i)}\right)\left(1-h_\theta\left(x^{(i)}\right)\right) \frac{\partial}{\partial \theta_j}\left(\theta^\top x^{(i)}\right)}{1-h_\theta\left(x^{(i)}\right)}\right]$$

$$\underset{\frac{\partial}{\partial \theta_j}\left(\theta^\top x^{(i)}\right)=x_j^{(i)}}{=} \frac{-1}{m} \sum_{i=1}^m \left[y^{(i)} \left(1-h_\theta\left(x^{(i)}\right)\right) x_j^{(i)} - \left(1-y^{i}\right) h_\theta\left(x^{(i)}\right) x_j^{(i)}\right]$$

$$\underset{\text{distribute}}{=} \frac{-1}{m} \sum_{i=1}^m \left[y^{i} - y^{i} h_\theta\left(x^{(i)}\right) - h_\theta\left(x^{(i)}\right) + y^{(i)} h_\theta\left(x^{(i)}\right)\right] x_j^{(i)}$$

$$\underset{\text{cancel}}{=} \frac{-1}{m} \sum_{i=1}^m \left[y^{(i)} - h_\theta\left(x^{(i)}\right)\right] x_j^{(i)}$$

$$= \frac{1}{m} \sum_{i=1}^m \left[h_\theta\left(x^{(i)}\right) - y^{(i)}\right] x_j^{(i)}$$

$$\frac{d}{dx}\sigma(x) = \frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right)$$

$$= \frac{-(1+e^{-x})'}{(1+e^{-x})^2}$$

$$= \frac{e^{-x}}{(1+e^{-x})^2}$$

$$= \left(\frac{1}{1+e^{-x}}\right)\left(\frac{e^{-x}}{1+e^{-x}}\right)$$

$$= \left(\frac{1}{1+e^{-x}}\right)\left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right)$$

$$= \sigma(x)\left(\frac{1+e^{-x}}{1+e^{-x}} - \sigma(x)\right)$$

$$= \sigma(x)\left(1-\sigma(x)\right)$$

Taken from:

https://stats.stackexchange.com/questions/278771/how-is-the-cost-function-from-logistic-regression-derivated

Logistic Regression and Gradient Descent

- Let's **solve our logistic regression with gradient descent.** In fact this will be useful because what we have just seen and will learn about gradient descent, will be our first building block when we go on to e.g. neural networks later.

- What is gradient descent?
  * In its simplest form, it is a first order optimization algorithm designed to find the minimum of a differentiable function.
  * This is achieved by iteratively taking (small) steps towards the negative of the gradient. For a function f(**x**) this is the direction in which it decreases the fastest.

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \lambda \nabla f(\boldsymbol{x}_n)$$

- With a step size lambda (called learning rate in ML) we can find the minimum of the function such that

$$f(\boldsymbol{x}_0) \geq f(\boldsymbol{x}_1) \geq \ \ldots \ \geq f(\boldsymbol{x}_n)$$

Gradient Descent: the ML perspective

- The previous slide gave a definition of how to apply gradient descent to a function. How do we use gradient descent for ML?
  => We calculate the gradient based on the cost/loss function and our model's predictions and iteratively learn a good set of parameters θ (in ML we generally optimize θ as there can be many more parameters than just the weights)

- We have already seen two such loss functions $\mathcal{L}$ that we can numerically optimize: the MSE and Cross-Entropy loss functions.

- In the simplest form we can directly update our weights with the calculated gradients

$$\theta = \theta - \lambda \nabla \mathcal{L}(\theta) = \theta - \lambda \frac{1}{N} \sum_{n=1}^{N} \nabla \mathcal{L}_n(\theta)$$

Gradient Descent

- We will use gradient descent in this week's notebook to optimize a logistic regression on the Titanic passenger dataset.

- Before we proceed: gradient descent doesn't come without caveats and thus there is many variants of gradient descent that address individual aspects, including stochastic update rules or adaptive learning rates. We will again take a look at some interactive widgets in:

  https://github.com/PyMLVizard/PyMLViz

- If you haven't been able to attend the lectures you should read and explore the articles on introduction to gradient descent and the follow-up on its variants (SGD, momentum, nesterov, AdaGrad, Adam etc.)