Entwicklung & Deployment von ML-Modellen mit python und mlflow

Entwicklung & Deployment von ML-Modellen mit python und mlflow

Über mich

- Florian Schmoll
- Mathematik in Kassel studiert
- Seit 2017 bei eoda als Data Scientist
- Arbeit in Data-Science-Projekten mit Kunden aus verschiedenen Branchen (Industrie, Handel, etc.)
- Durchführung von Trainings in R und python (Einführung in DS & ML, Zeitreihen, …)
- Fokus auf ML-Lifecycle-Management und ML im Allgemeinen
- R-Muttersprachler; Python-Fremdsprachler

eoda FACTS

2010

Gegründet
10 Jahre Erfahrung

>50

Mitarbeiter
Interdisziplinäres Team

>200

Kunden
in unseren Projekten

14

DAX-Konzerne
vertrauen uns

>1.700

Teilnehmer
in unseren Schulungen

eoda
daten ▸ wissen ▸ nutzen

EINE AUSWAHL UNSERER REFERENZEN

UNSER PORTFOLIO

Beratung & Projekte

Strategieberatung

Use-Case-Konzeption

Prototyping

Produktivsetzung

Wissenstransfer

Data-Science-Training

Coaching & Support

Impulsvorträge

aicon | Assessment

aicon | Implementierung

aicon | Betrieb & Support

Technologie & Infrastruktur

Software

Ziele und Inhalte des Workshops

- Ziel: Grundsätzliches Verständnis vom „ML-Workflow“ und „ML-Lifecycle“
  - Konzepte des ML: training/test-Split, Experimente, Grid Search, Cross Validation, …
  - mlflow-Komponenten: Tracking, Projects, Models, Registry

- Nutzung der python-Bibliotheken pandas und scikit-learn zur Entwicklung von ML-Modellen
- Nutzung von mlflow zum Abbilden des „ML-Lifecycles“

- Hands-On: Angebot zum „mitcoden“ und Wechsel zwischen „Live-Demo“ und „Praxis-Blöcken“

Agenda

- 9.00 – 10.30 – Block I
  - Vorstellung der Inhalte und Agenda
  - technisches Setup
  - Einführung in die grundlegenden Begriffe des ML
  - Entwicklung von ML-Modellen mit scikit-learn
- 10.45 – 12.15 – Block II
  - Einführung in das Konzept des ML-Lifecycle und die Komponenten von mlflow
  - mlflow Tracking
- 13.15 – 14.45 – Block III
  - mlflow projects & models
  - Einfache neuronale Netze mit tensorflow/keras
- 15.00 – 16.00 – Block IV
  - mlflow registry
- 16.00 – 16.30 – Ende
  - Offene Fragen & Diskussion

Techniches Setup

- python: https://www.python.org/downloads/
- mlflow: https://mlflow.org/docs/latest/quickstart.html
- evtl. sqlite: https://www.sqlite.org/download.html
- evtl. git: https://git-scm.com/downloads
- Linux oder MacOS, Windows funktioniert nicht zwangsläufig
- Skripte, Notebooks und mehr Infos im git repo: https://github.com/SchmoFl/mlflow_enterpy

- mlflow server kann auf dem lokalen System oder auf einem externen Server laufen (z.B. in der Cloud, docker, Raspberry Pi)
- Mein Setup für diesen Workshop: Jupyter Notebook, python (3.7), mlflow (1.17) und sqlite auf einem Raspberry Pi 4 (Linux)

Cross Industry Standard Porcess for Data Mining

- **Business Understanding**

  Verständnis für die zugrundeliegenden Herausforderungen entwickeln
- **Data Understanding**

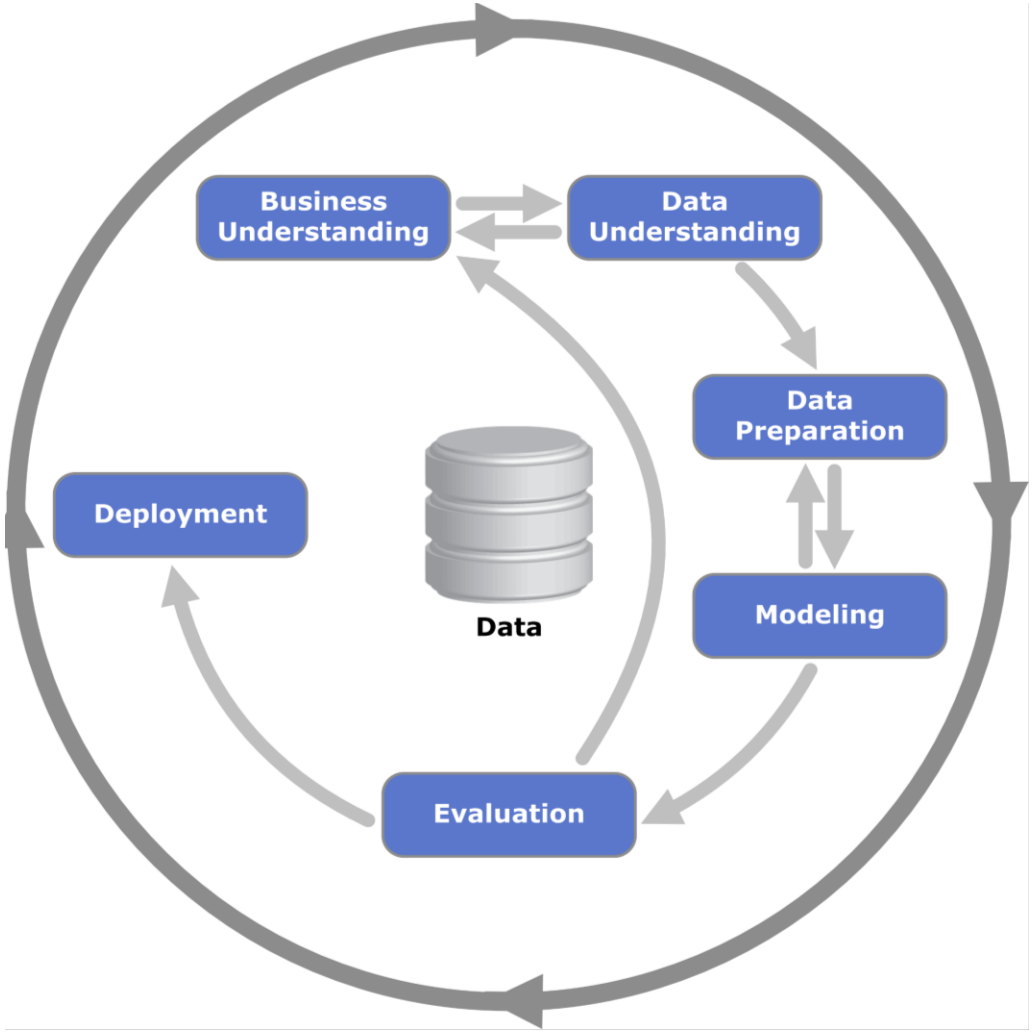
  Explorative Datenanalyse
- **Data Preparation**

  Data Cleaning, Feature Engineering, …
- **Modeling**

  Auswahl und Training von Modellen
- **Evaluation**

  Modell-Performance evaluieren
- **Deployment**

  Einbettung in Business Prozesse und bestehende Systeme



https://en.wikipedia.org/wiki/Cross_Industry_Standard_Process_for_Data_Mining#/media/File:CRISP-DM_Process_Diagram.png

Teilgebiete des Machine Learning

- Supervised Learning: Vorhersage einer Zielvariable
- Unsupervised Learning: Erkennen von Zusammenhängen ohne Zielvariable (z.B. Clustering oder Anomaly Detection)
- Reinforcement Learning, Active Learning, …

- Klassifizierung: Vorhersage einer diskreten (Klassen-)variable (z.B. Churn Prediction, Fraud Detection, Image Classification, …)
- Regression: Vorhersage einer metrischen Variable (z.B. Preis- oder Bedarfsprognosen)

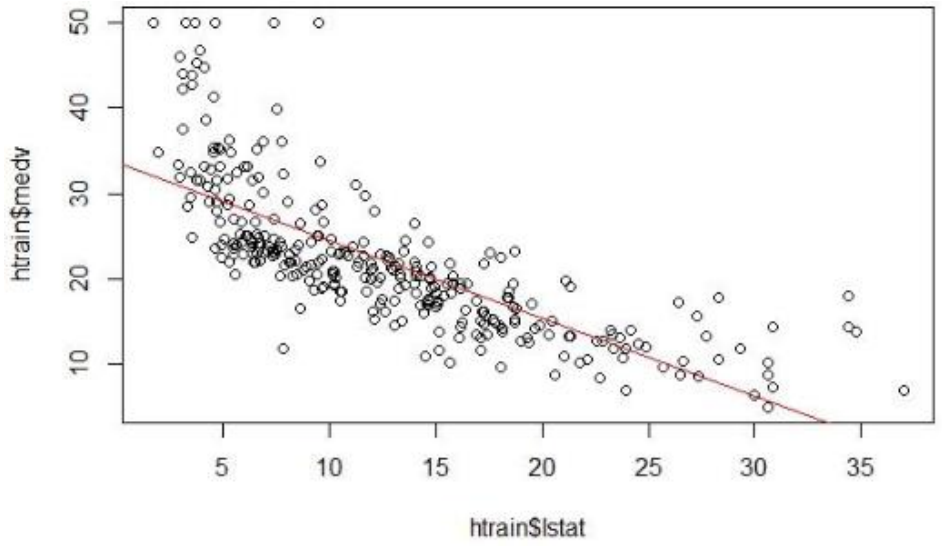
- Daten liegen z.B. in Form eines Datensatzes/DataFrames vor
  - Rechteckige/2-dim. Datenstruktur
  - Zeilen: Beobachtungen, Ausprägungen
  - Spalten: Merkmale, Variablen, Feature
- Außerdem: Zeitreihen, Bild- oder Audiodaten (Tensoren – höherdimensionale DataFrames)

Lineare Regression

Das einfachste Modell, um eine abhängige Variable (Zielvariable) durch eine oder mehrere unabhängige Variablen (vorhersagende Variablen) zu erklären.

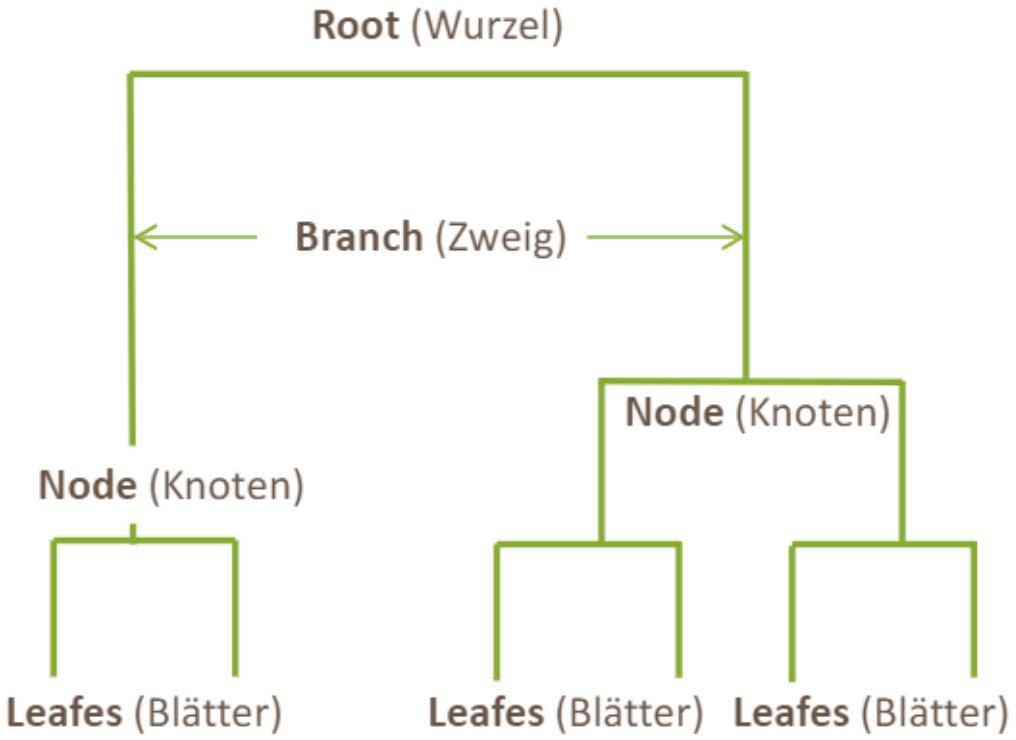
$$Hauspreis = \beta_0 + \beta_1 * Zimmeranzahl + \beta_2 * Lage + \varepsilon$$

Linearer Zusammenhang zwischen einer **abhängigen** und einer **unabhängigen** Variablen:



Entscheidungs- & Regressionsbäume

- Teilen Daten in hierarchisch homogene Klassen auf
- Bilden Entscheidungsregeln ab
- Klassifizierung → Entscheidungsbaum Regression → Regressionsbaum



Training/Test-Split

- Zur Evaluation der ML-Modelle werden die Daten in mind. einen Trainings- und Testdatensatz (auch: Evaluationsdatensatz) geteilt
- Die Modelle werden auf Grundlage der Trainingsdaten trainiert
- Die Modelle werden auf Grundlage der Testdaten evaluiert
- Ziel: Vermeidung von Overfitting, d.h. Modell kann die Trainingsdaten sehr gut abbilden, hat aber Probleme mit neuen Daten

Metriken zur Modellevaluation | Regression

Root Mean Squared Error (RMSE):

Wurzel der Mittelwerte der quadrierten Abweichung zwischen den wahren Werten und der Prognose

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum(\mathrm{Wahr.} - \mathrm{Vorh.})^2}$$

Mean Absolut Error (MAE):

Mittelwert der absoluten Abweichung zwischen den wahren Werten und der Prognose

$$\mathrm{MAE} = \frac{1}{n}\sum|\mathrm{Wahr.} - \mathrm{Vorh.}|$$

Maximaler Absolute Error (Max AE):

Die größte Abweichung zwischen den wahren Werten und der Prognose

$$\mathrm{MAX\ AE} = \max(\mathrm{Wahr.} - \mathrm{Vorh.})$$

Mean Error:

Mittelwert der Abweichung zwischen den wahren Werten und der Prognose

$$\mathrm{Bias} = \frac{1}{n}\sum(\mathrm{Wahr.} - \mathrm{Vorh.})$$

Metriken zur Modellevaluation | Klassifizierung

Präzision
Der Anteil der **echt Positiv**-Werte an den positiv **prognostizierten** Werten insgesamt: $\frac{A}{A+B}$

Sensitivität (Recall)
Der Anteil der **echt Positiv**-Werte an den **tatsächlich** positiven Werten: $\frac{A}{A+C}$

Spezifität
Der Anteil der **echt Negativ**-Werte an den **tatsächlich** negativen Werten: $\frac{D}{B+D}$

Genauigkeit
Der Anteil der **echt Positiven/Negativen**-Werte an den **gesamten** Werten: $\frac{A+D}{A+B+C+D}$

Confusion Matrix

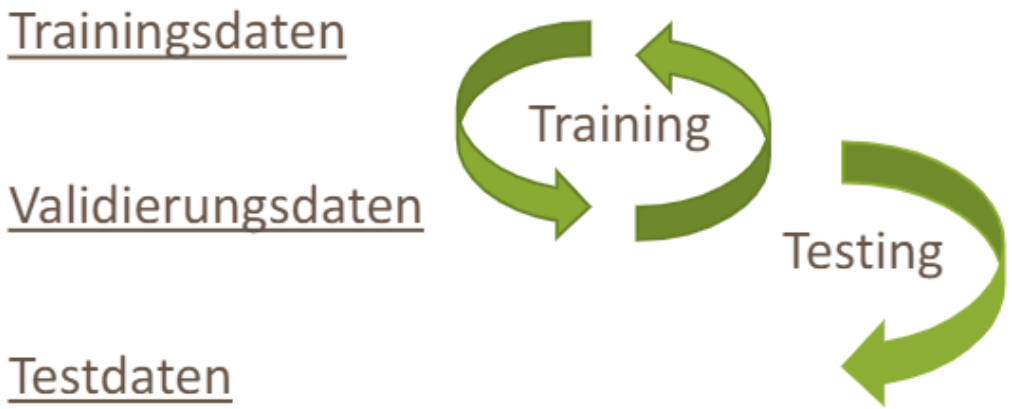
		Tatsächlich		
		Positiv	**Negativ**	**Summe**
Prognose	**Positiv**	Echt Positiv **A**	Falsch Positiv **B**	Positive Prognosen **A+B**
	Negativ	Falsch Negativ **C**	Echt Negativ **D**	Negative Prognosen **C+D**
	Summe	Positive tatsächlich **A+C**	Negative tatsächlich **B+D**	

Hyperparametertuning

ML-Algorithmen weisen in der Regel verschiedene Konfiguartionsmöglichkeiten (Hyperparameter) auf, die den Algorithmus beeinflussen

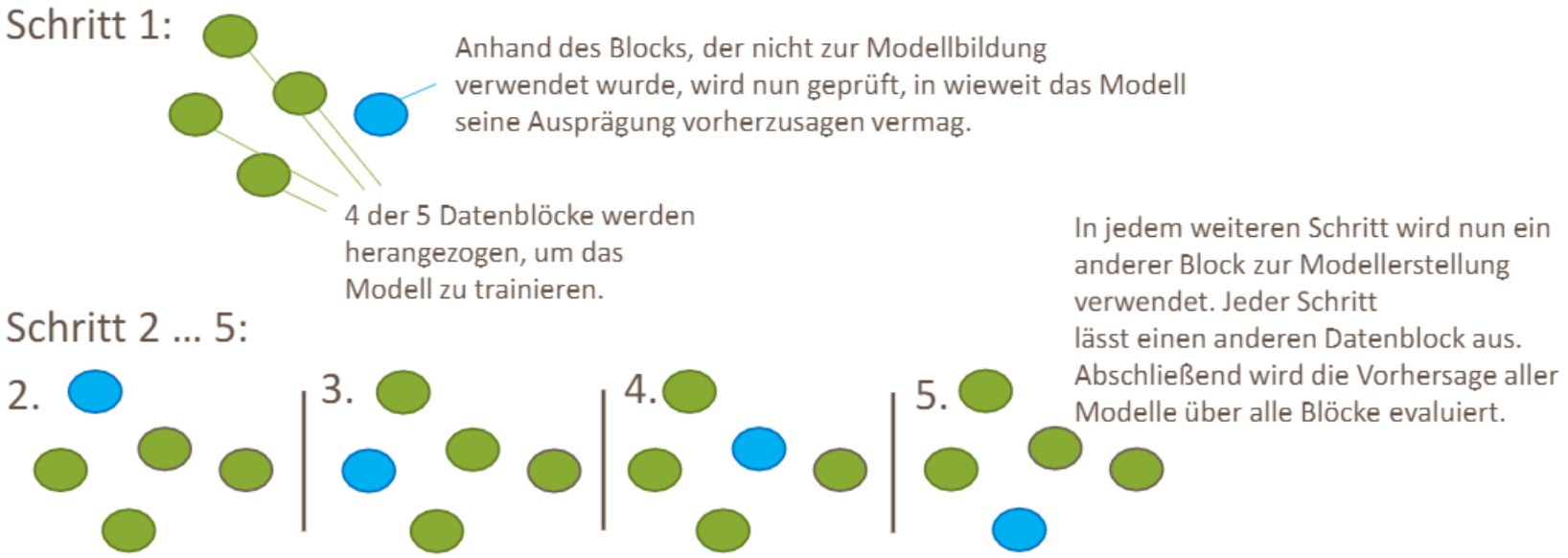
Die besten Hyperparameterkombinationen sind nicht bekannt und müssen anhand der spezifischen Daten abgeleitet werden.

Dafür werden anhand der Trainingsdaten mehrere kleinere Trainigs/Test-Splits erstellt, auf denen verschiedene Hyperparameterkombinationen evaluiert werden. Diese neuen (Test-)Daten bezeichnet man auch als Validierungsdaten.



k-Fold Cross Validation

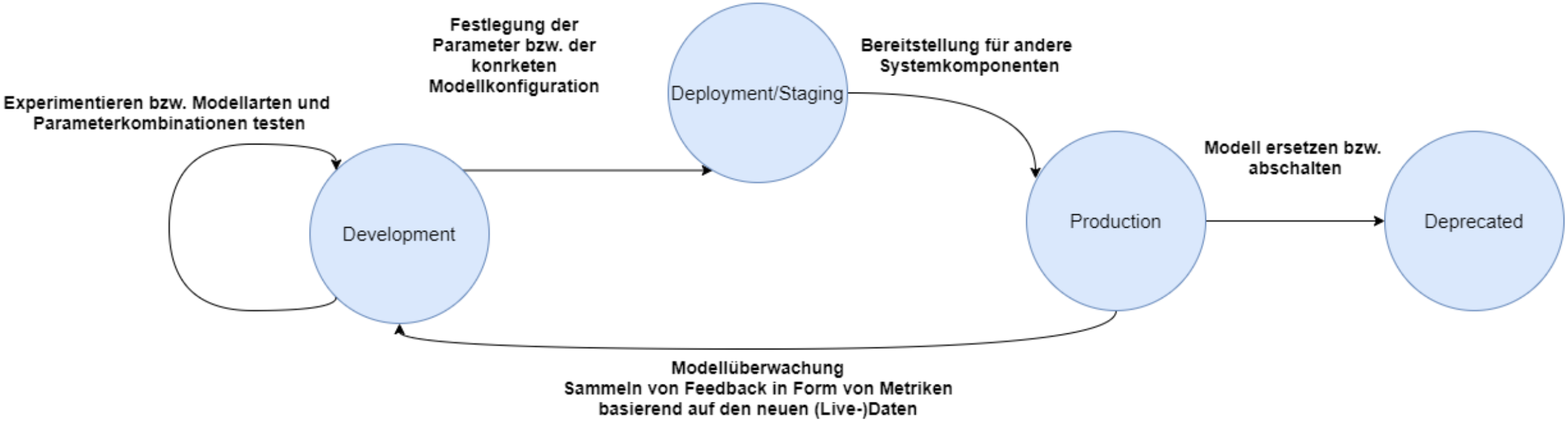
- Einteilung der (Trainings)Daten in k Blöcke
- In k Durchgängen (Folds) bilden jeweils k-1 Blöcke den jeweiligen Trainingsdatensatz und der letzte Block den Validierungsdatensatz
- Jede Hyperparameterkombination wird auf allen folds evaluiert. Die im Durchschnitt beste Hyperparameterkombination wird ausgewählt.
- Es ist möglich dieses Verfahren mehrfach hintereinander auszuführen. Ziel: Simulation der Vorhersage neuer Daten



Machine Learning Experiment, Run, Model

- Untersuchung, wie sich eine Änderung des Inputs auf den Output auswirkt
- Input = Parameter (z.B. Art des Modells, Hyperparameter, Größe des Datensatzes, o.ä.)
- Output = Metriken (RMSE, MAE, Accuracy, usw.)
- Experimente werden mehrmals in Runs durchgeführt, um die beste Parameterkombination zur Optimierung der Metrik(en) zu finden
- Model: Blackbox, die eine Prediction aufgrund von Eingangsdaten generiert

ML Model Lifecycle



mlflow - Konzepte

- It’s difficult to keep track of experiments → **mlflow tracking**
- It’s difficult to reproduce code → **mlflow projects**
- There’s no standard way to package and deploy models → **mlflow models**
- There’s no central store to manage models (their versions and stage transitions) → **mlflow registry**

mlflow tracking

- Organisieren von experiments und runs

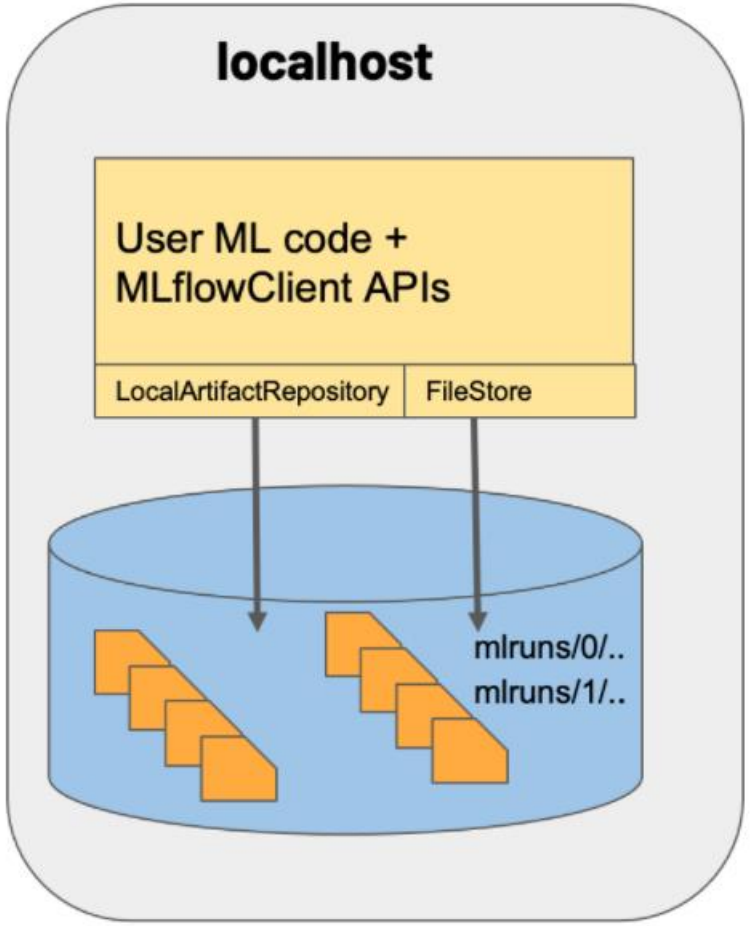
- Logging von
  - params
  - metrics
  - artifacts (Grafiken, Text-Files)
  - tags
  - models
  - start & end time

Wo werden die Informationen gespeichert?

- Tracking – experiments, runs, params, metrics, meta-info,…
  - Lokales Verzeichnis: Speichern in lokalen Directories/Files (*default*)
  - SQLAlchemy-kompatible DB (sqlite, postgresql, mysq): Speichern der Informationen in der DB
  - HTTP-Server (mlflow tracking server)
  - Databricks workspace

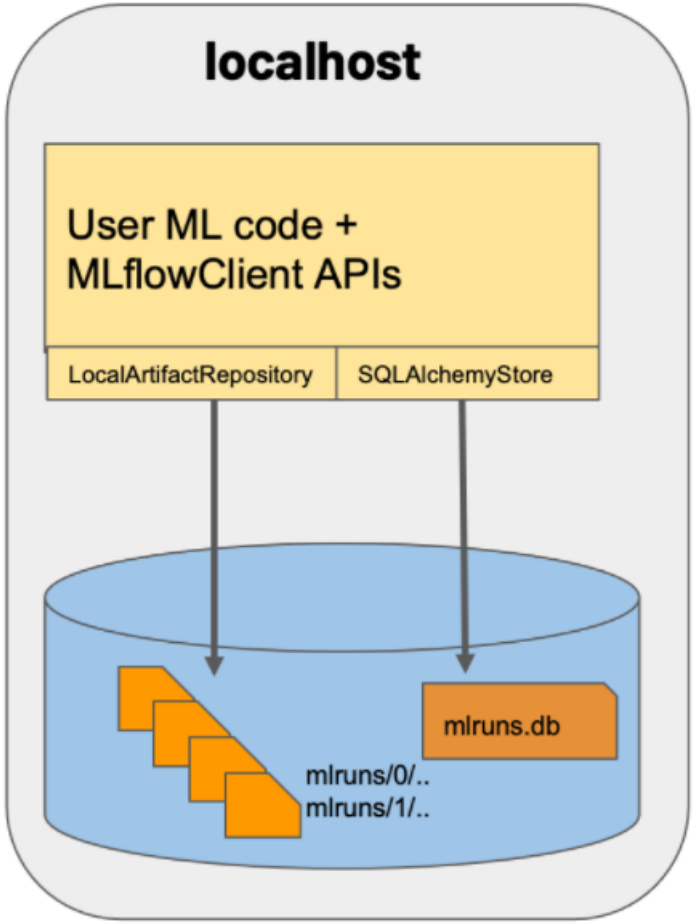
- Artifact Store
  - Lokales Verzeichnis
  - Amazon S3 Storage
  - Azure Blob Storage
  - Google Cloud Storage
  - FTP Server
  - SFTP Server
  - NFS
  - HDFS

Architektur



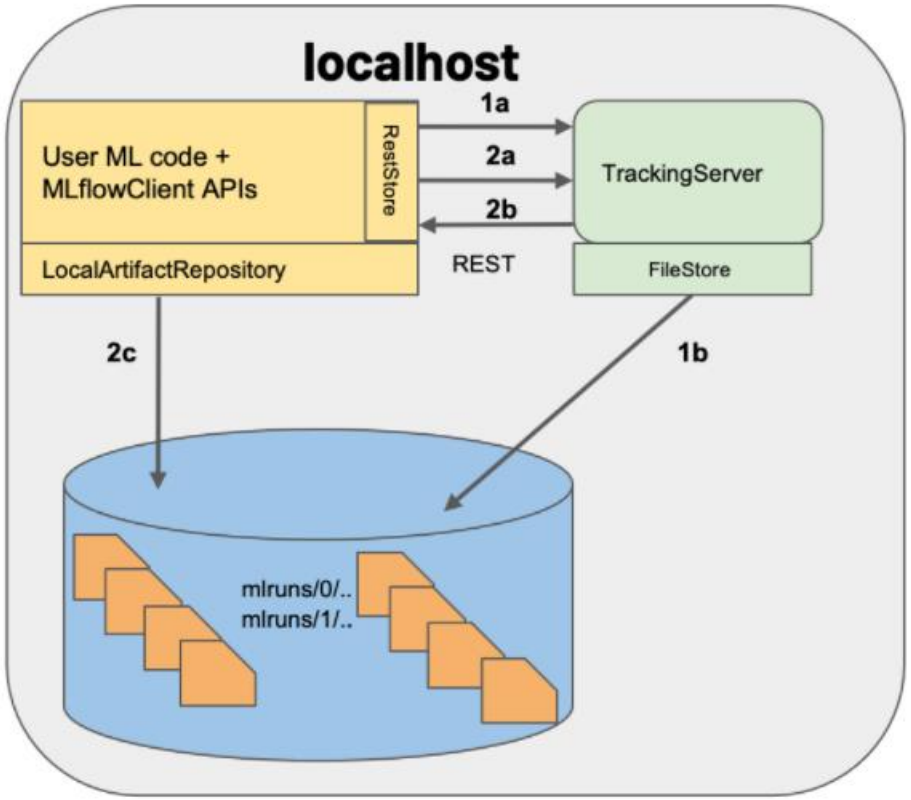
Scenario 1: MLflow on the localhost

Architektur



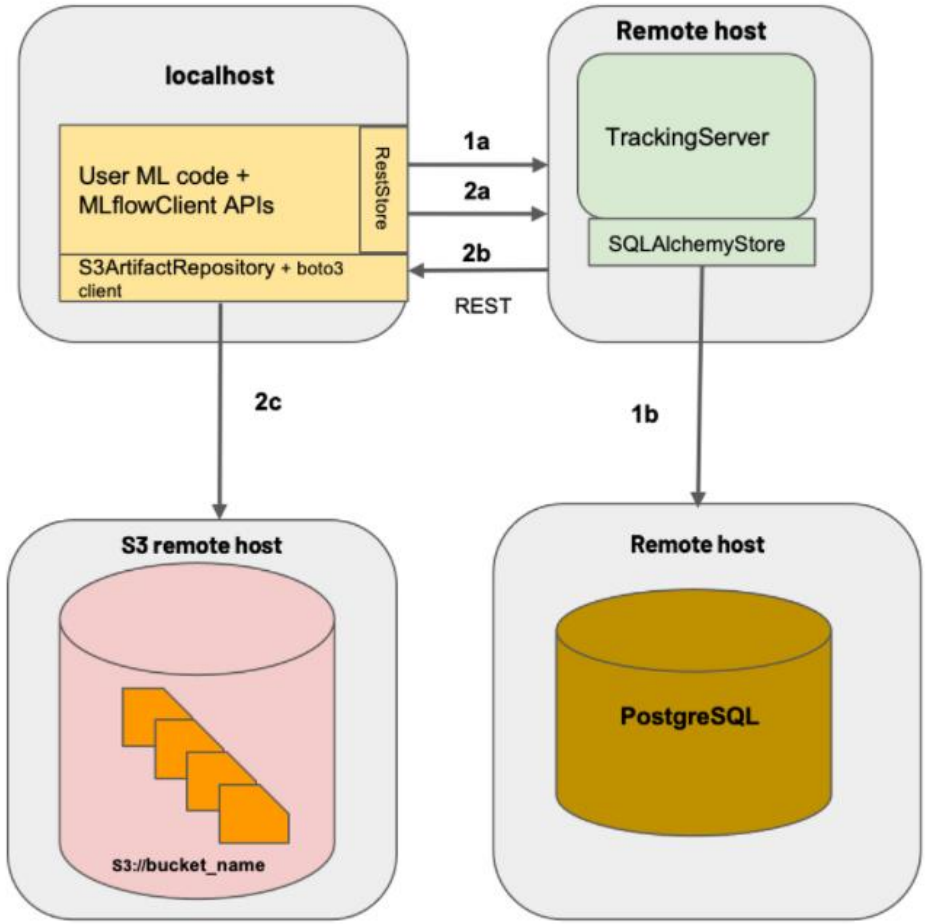
Scenario 2: MLflow on the localhost with backend store as an SQLAlchemy compatible database type: **SQLite**

Architektur



Scenario 3: Tracking server launched at localhost: `mlflow server --backend-store-uri file://path/mlruns`

Architektur



Scenario 4: `mlflow server --backend-store-uri postgresql://URI --default-artifact-root S3:/bucket_name --host remote_host`

mlflow models

- Standardformat für ML-Modelle, die u.a. als REST-API oder für batch inference (Spark) benutzt werden können

- Modelle können in versch. *flavors* gespeichert werden
  - *python function, R function*
  - *Keras, Pytorch, Tensorflow, MXNet Gluon, ONNX*
  - *Spark MLlib, MLeap*
  - *Scikit-learn, Spacy, Statsmodels*
  - *XGBoost, LightGBM, CatBoost*

- Modelle können gespeichert und geladen werden. Außerdem können sie während eines runs gelogged werden.

Deployment eines mlflow Modells

- Speichern eines sklearn-Modells via **mlflow.sklearn.save_model(model, 'my_model')** erzeugt folgendes Verzeichnis:

```
my_model/
├── MLmodel
└── model.pkl
```

- Das **MLmodel** file (yaml) beschreibt zwei *flavor:*

```
time_created: 2018-05-25T17:28:53.35

flavors:
  sklearn:
    sklearn_version: 0.19.1
    pickled_model: model.pkl
  python_function:
    loader_module: mlflow.sklearn
```

- Deployment des Modells: **mlflow models serve –m my model**

mlflow projects

- Spezifierung eines Projekts in einem project-file
  - Name
  - Environment, in dem das Projekt ausgeführt werden soll
    - conda environment
    - docker container
  - Entry Points
    - Befehle, die innerhalb des Projektes ausgeführt werden
    - Parameter, die den Befehlen übergeben werden

```yaml
name: My Project

conda_env: my_env.yaml
# Can have a docker_env instead of a conda_env, e.g.
# docker_env:
#    image:  mlflow-docker-example

entry_points:
  main:
    parameters:
      data_file: path
      regularization: {type: float, default: 0.1}
    command: "python train.py -r {regularization} {data_file}"
  validate:
    parameters:
      data_file: path
    command: "python validate.py {data_file}"
```

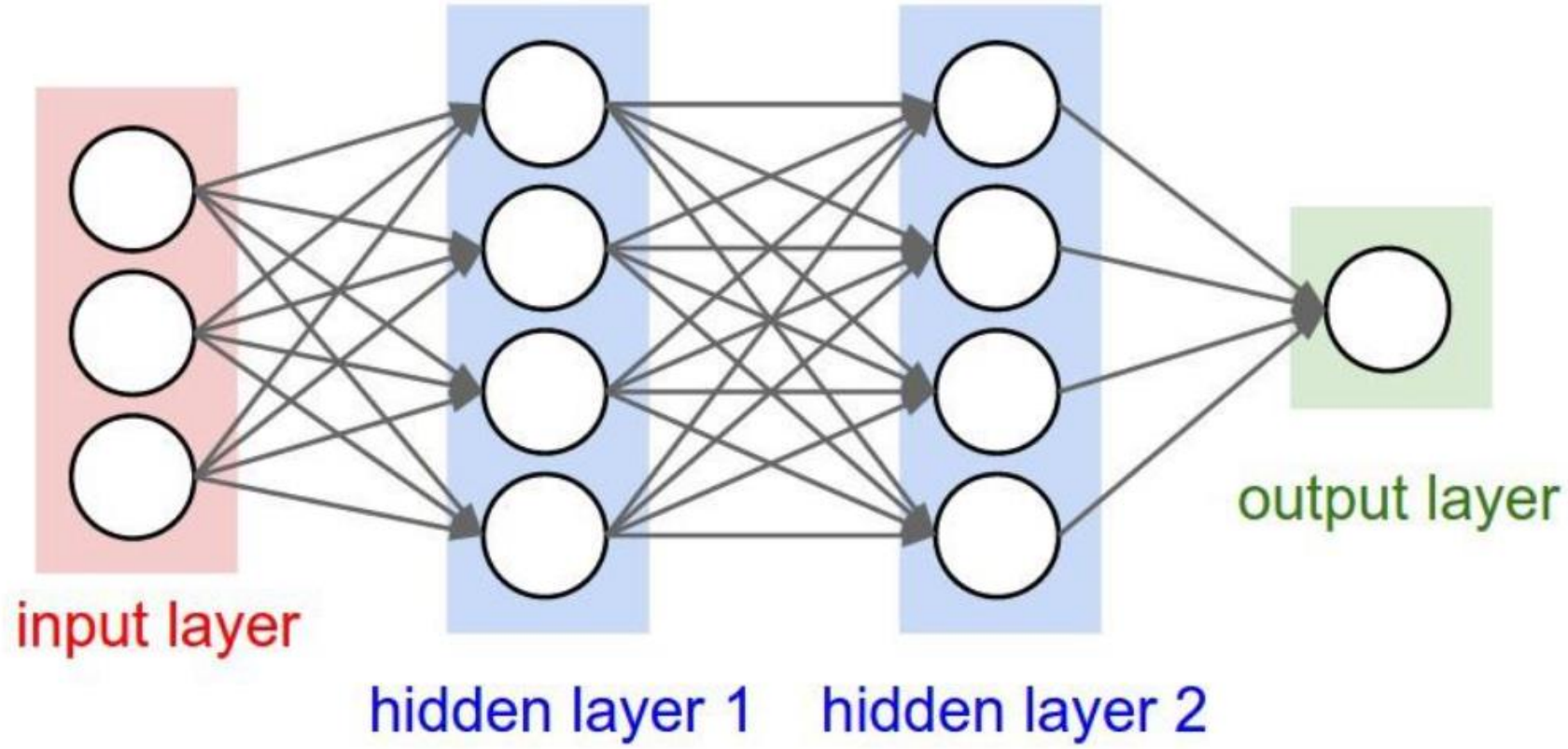
Your turn

- Schreibe ein kleines python Skript, dass einen Datensatz einliest und ein kleines ML-Modell trainiert
- Logge dabei mithilfe von mlflow die von dir bestimmten Hyperparameter und von dir ausgewählte Metriken (Trainings-/Test-Split)
- Parametrisiere dein Skript (z.B. Hyperparameter)
- Leg ein MLProject-File mit deinem Skript als entry_point an und führe das Skript mithilfe des mlflow CLI aus
- Bonus: Teile deinen Workflow in 2 Skripte auf (z.B. preprocessing, Modellierung, Evaluierung, etc.)

mlflow registry

- model registry benötigt ein DB backend
- Ein Modell kann auf zwei Weisen registriert werden:
  - UI
    - Auswählen des Modells innerhalb eines Runs
    - Registrierung durch Vergabe eines Namens
  - API
    - Mit der log_model-Methode (Logging + Registrieren)
    - Mit der register_model-Methode (Registrieren eines bereits geloggten Modells)
    - Mit den create_registered_model + create_model_version Methoden (Erstellen eines leeren registrierten Modells und einer neuen Version dieses Modells durch Angabe eines geloggten Modells)
- Registrierte Modelle können geladen und deployed werden

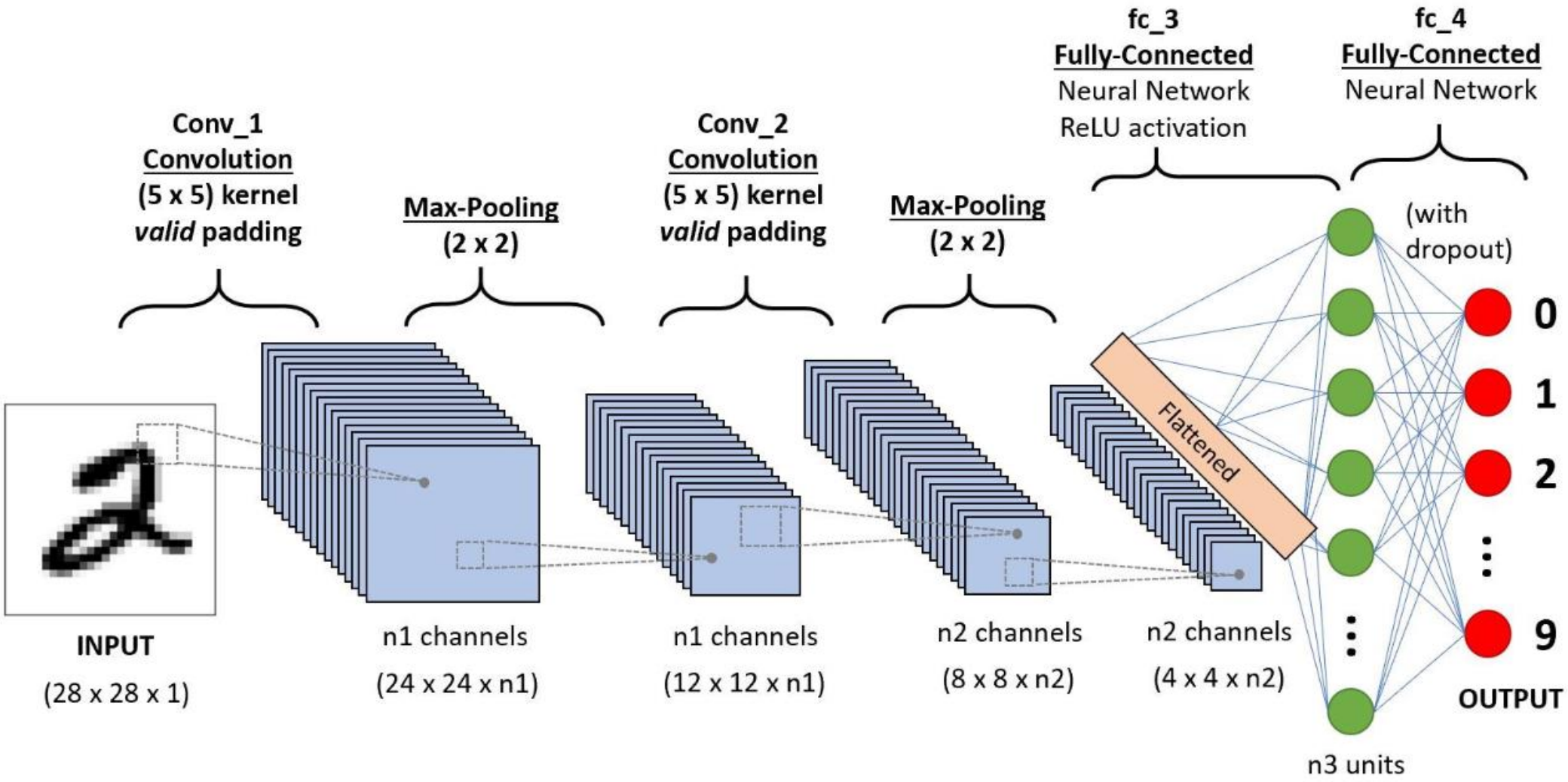
Neuronale Netze



Arten von neuronalen Netzen

- Feed-Forward Neural Networks
- Convolutional Neural Networks (z.B. Bilderkennung)
- Recurrent Neural Networks (z.B. Textdaten oder zeitabhängige Daten)
- ...

Convolutional Neural Network



Network Layer

- Linear Layers
- Convolutional Layers
- Pooling Layers
- Dropout Layers
- Normalization Layers
- Recurrent Layers

Die Data Science Spezialisten.

eoda GmbH
Universitätsplatz 12
34127 Kassel
www.eoda.de
info@eoda.de
+49 561 202724-40

@eodaGmbH

blog.eoda.de

@eodaGmbH

eodaGmbH