From Machine Learning to Deep Learning: A concise introduction
March 26-28, 2024, HLRS

Day 1:
Pre-processing, Feature Engineering and Machine Learning (Lorenzo Zanon)
Day 2:
Focus on supervised Deep Learning to classify images of waste in the wild
(Khatuna Kakhiani)
… continues…

From Machine Learning to Deep Learning: A concise introduction
March 26-28, 2024, HLRS

Day 3:

- Guest Lecture: Towards Data-Driven Computational Fluid Dynamics (A. Beck, A. Schwarz IAG)
- Generalization and the problem of leakage (Nico Formanek)
- Data Compression of numerical data sets with the BigWhoop library (Patrick Vogler)

Learning outcomes

- Pre-processing:
  - Read-in, transform and merge datasets,
  - Derive new features,
  - Correlation analysis,
  - *Implicit* data processing:
    Vectorisation, Normalisation, Assembling.

Part I
Part II

Learning outcomes

- Supervised learning algorithms:
  - Linear regression,
  - Random forest (classification).
- Steps of the Machine Learning pipeline:
  - **Training**, regularisation and cross validation,
  - Prediction/Inference on a **test** dataset.
- Working on a cluster.
- Parallel Spark.

Work in progress

Part II
Part III

Learning outcomes

+

Machine Learning concepts requires in Deep Learning and applications (e.g., CFD in Day 3).

–

Delay prediction of Stuttgart metropolitan trains:
We focus on the general concepts.
We will **not** outperform existing tools provided by the Deutsche Bahn or the public transport in Stuttgart (VVS).

Focus on Pre-processing, Feature Engineering and Machine Learning

- Part I: Introduction
  - Pre-processing
- Part II: Example on the Jupyter Notebooks
  - Pre-processing
  - Supervised learning techniques in a Machine Learning pipeline
- Part III: HLRS Systems ~~and Example as a Python script~~
  - Work on a cluster, parallel Spark

Work in progress

Tue, March 26, 2023 **09:00 – 17:xx**
Lunch break 13:00 – 14:15

Part I
Part II
Part III

Focus on Pre-processing, Feature Engineering and Machine Learning

- Part I: Introduction
  - Pre-processing
    - « General Introduction
    - « Source Data

More learning outcomes

Main index

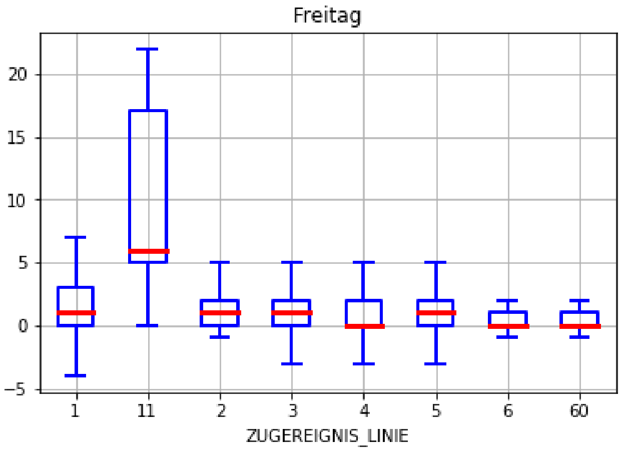
Part I
Part II
Part III

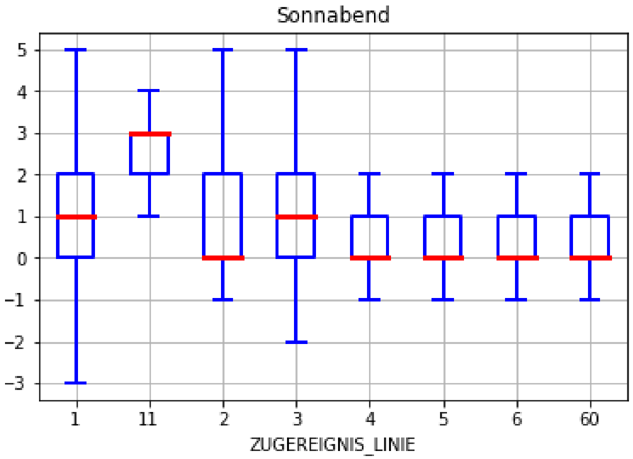
Introduction

Focus on Pre-processing, Feature Engineering and Machine Learning

Stuttgart S-Bahn Example

Lorenzo Zanon, Oleksandr Shcherbarkov, Dennis Hoppe (HLRS), Li Zhong (former HLRS)





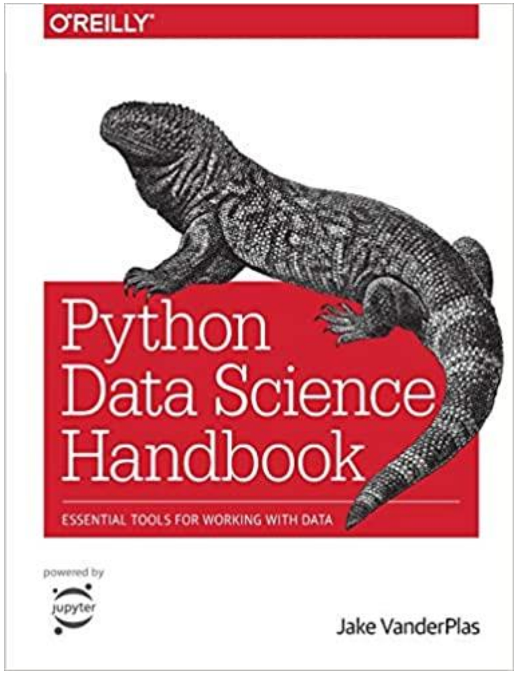
Further acknowledgements

- Origin of this example: Project “Simulated Worlds”, a cooperation between:
  - HLRS,
  - Steinbuch Centre for Computing (SCC),
  - Stuttgart Research Center for Interdisciplinary Risk and Innovation Studies (ZIRIUS).
- Ahmed Masood (HLRS working student 01/07/2020 – 30/06/2021)

https://simulierte-welten.de/

Main reference

- **Python Data Science Handbook** by Jake VanderPlas (O’Reilly). Copyright 2017 Jake VanderPlas, 978-1-491-91205-8.


O'REILLY
Python Data Science Handbook
ESSENTIAL TOOLS FOR WORKING WITH DATA
powered by jupyter
Jake VanderPlas

- Abbreviated as **[PHB, *page*]**
- Online (with code of the examples): `https://jakevdp.github.io/PythonDataScienceHandbook`

General objectives of the example

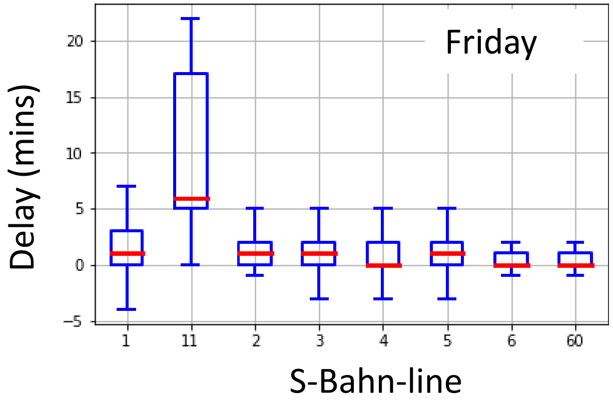
Can I improve my travel experience in the Stuttgart S-Bahn with the help of Machine Learning?

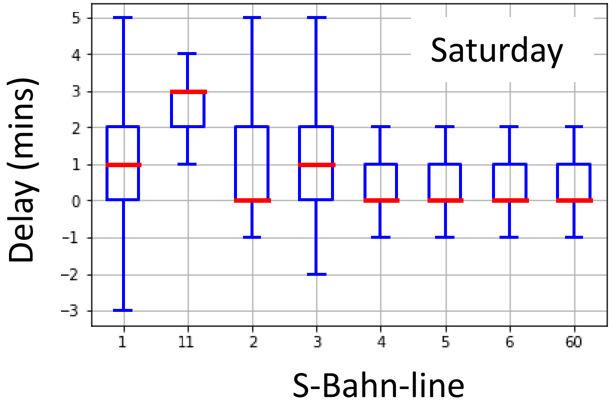
1. Predict the S-Bahn delays accurately to the **minute**: is that feasible?
2. Is my train going to be **late at all**?

- How to deal with a given set of data?
  → data preparation and **data manipulation**
- ML **pipeline**: Training and Validation, Test.
- ML quality optimisation.

Explorative analysis / Manipulation

- **Explorative / statistical** analysis:
  - make a first interpretation,
  - extract first simple statistical values for the **delay**.





- Problem: The initial **set of features** is small, the data are not ready for use!

Explorative analysis / Manipulation

Naming convention

- Sample: Each observation as an entry in the dataset.
- **Features and feature vectors <u>quantitatively</u> describe a sample** and are used to obtain a prediction.

Weather	Day of Week	Line	Duration	Pollution	Delay
Rainy	Monday	S2	35	Low	Delayed
Storm	Tuesday	S3	40	High	On time
Sunny	Thursday	S60	30	Medium	On time

sample

feature vector

Explorative analysis / Manipulation

Naming convention (cont’d)

- … quantitatively: Numbers, but also **categorical data**, (texts), images! (→ part 2)
- DataFrames: Each **row** corresponds to a **feature vector** describing a sample.

Weather	Day of Week	Line	Duration	Pollution	Delay
Rainy	Monday	S2	35	Low	Delayed
Storm	Tuesday	S3	40	High	On time
Sunny	Thursday	S60	30	Medium	On time

sample →

feature vector →

Explorative analysis / Manipulation

Naming convention (cont'd)

- … **Label or target**: "Special" feature that must be predicted from the data (dependent variable).

label or target

Weather	Day of Week	Line	Duration	Pollution	Delay
Rainy	Monday	S2	35	Low	Delayed
Storm	Tuesday	S3	40	High	On time
Sunny	Thursday	S60	30	Medium	On time

https://webis.de/downloads/lecturenotes/machine-learning/unit-en-ml-introduction.pdf

[PHB from p. 375]

Pre-processing and Feature engineering

What do we do with the data:

Before building the ML model:

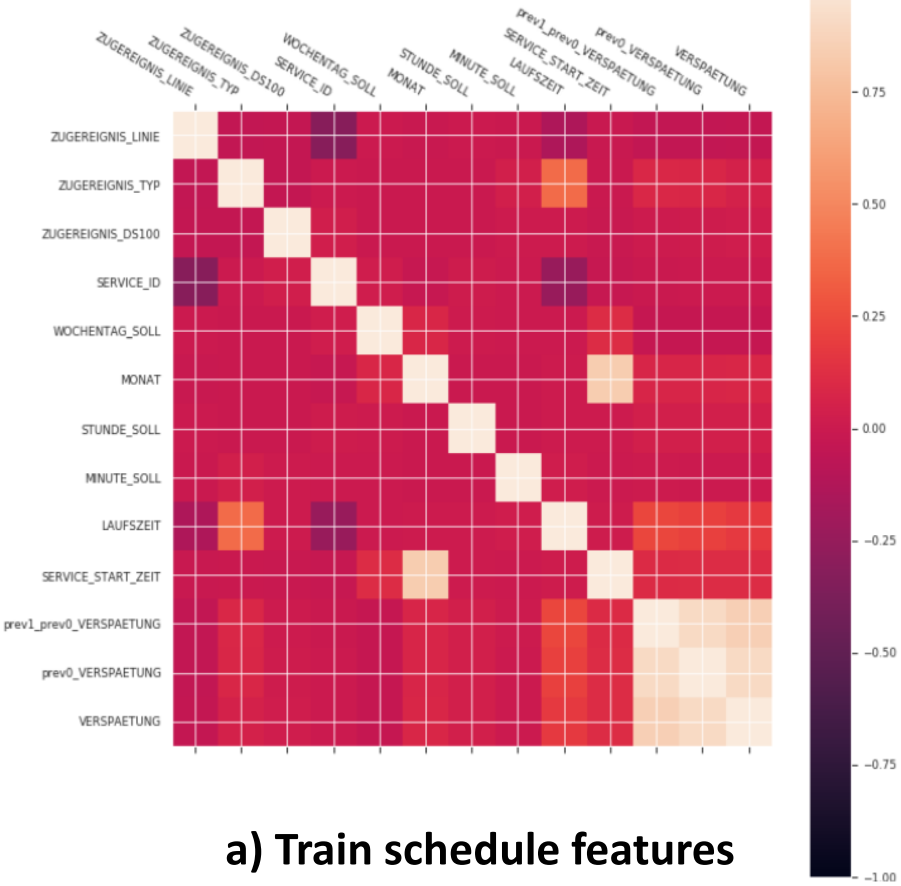
- Clear-up **noisy** data / outliers,
- Perform **data augmentation** / **fusion**,
- **Partition** the data between a training and a test set,
- Further explorative analysis:
  Find out linear relations with a **correlation map**.

Example: next slide / Details: later on.

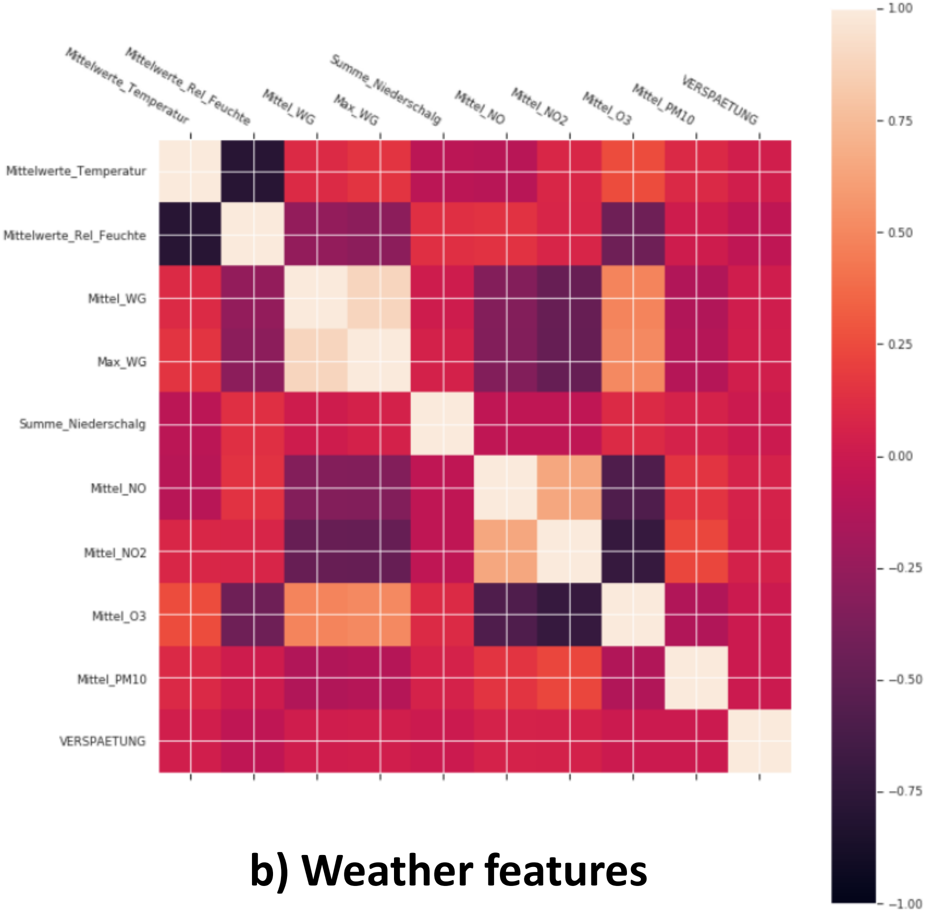
After having built the ML model:

- Evaluate which features deliver the best model quality.

Pre-processing and Feature engineering



a) Train schedule features



b) Weather features

Predicting Train Delays with Machine Learning

- **ML/DL Models...**
  will **learn** how to solve our problem on our data without need of explicit programming (Arthur Samuel, 1959).
- The **model** is defined by a (large in DL) set of parameters or **weights**.
- The weights are **learnt** and progressively improved through **training**.
- The **feedback** received from the data corresponds to different **learning** types.

[Vivienne Sze et al., Efficient Processing of Deep Neural Networks, 2017.]

Predicting Train Delays with Machine Learning

Details skipped

Active Learning:

- The training samples **can be** labelled by a "teacher".

Supervised Learning:

- All training samples **are** already labelled.
- **Discriminative** classifiers partition **test** data by predicting given labels.

→ **This example:** The *true* **delay** is known for all training samples.

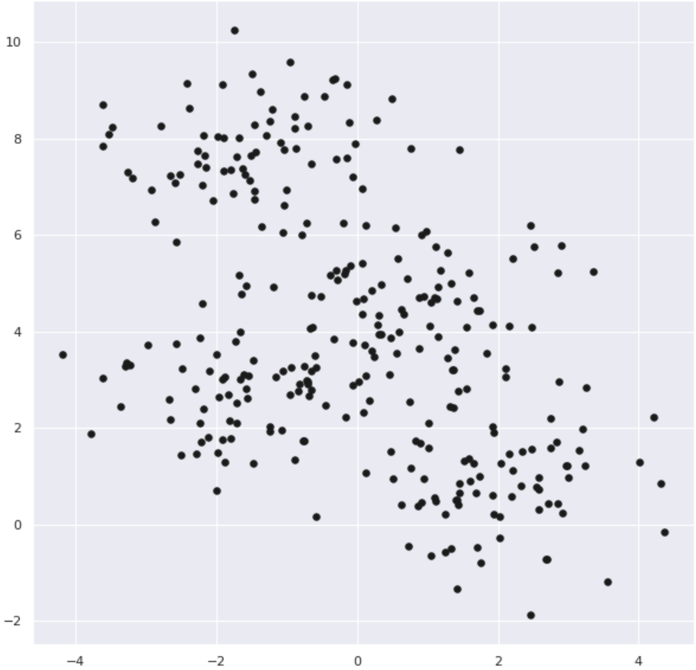
[Vivienne Sze et al., Efficient Processing of Deep Neural Networks, 2017.]

Predicting Train Delays with Machine Learning

ML/DL Models:

Unsupervised Learning:

- The training samples are **not labelled** (→ no feedback).
- The algorithm will **identify the label**
  - as a pattern or structure,
  - or a subdivision into clusters of the data



(**generative** classifiers and clustering).

- Model reduction or compression are further examples. → **Day 3**

[code adapted from: PHB Notebooks]

[Vivienne Sze et al., Efficient Processing of Deep Neural Networks, 2017.]

Details skipped

Predicting Train Delays with Machine Learning

ML/DL Models:

Semi-supervised Learning:

- Only a small subset of the training data is labelled.
- The (few) labelled data can **define** the cluster regions.
- Unlabelled data can be used to define the cluster **boundaries**.

Transfer Learning:

- Use a model trained on one task and re-train to use it on a different (or more specific) task.
- E.g. in computer vision, use a CNN to extract basic features and FCLs for the final goal (detection, segmentation etc.).

[Vivienne Sze et al., Efficient Processing of Deep Neural Networks, 2017.]

Details skipped

Predicting Train Delays with Machine Learning

ML/DL Models:

Reinforcement Learning:

- Rather than to predict an information, the ML/DL model predicts an **action**…
- …leading to a **reward** or to a **cost**.
- The weights of the model are trained to **maximise the payoff**.

[Vivienne Sze et al., Efficient Processing of Deep Neural Networks, 2017.]

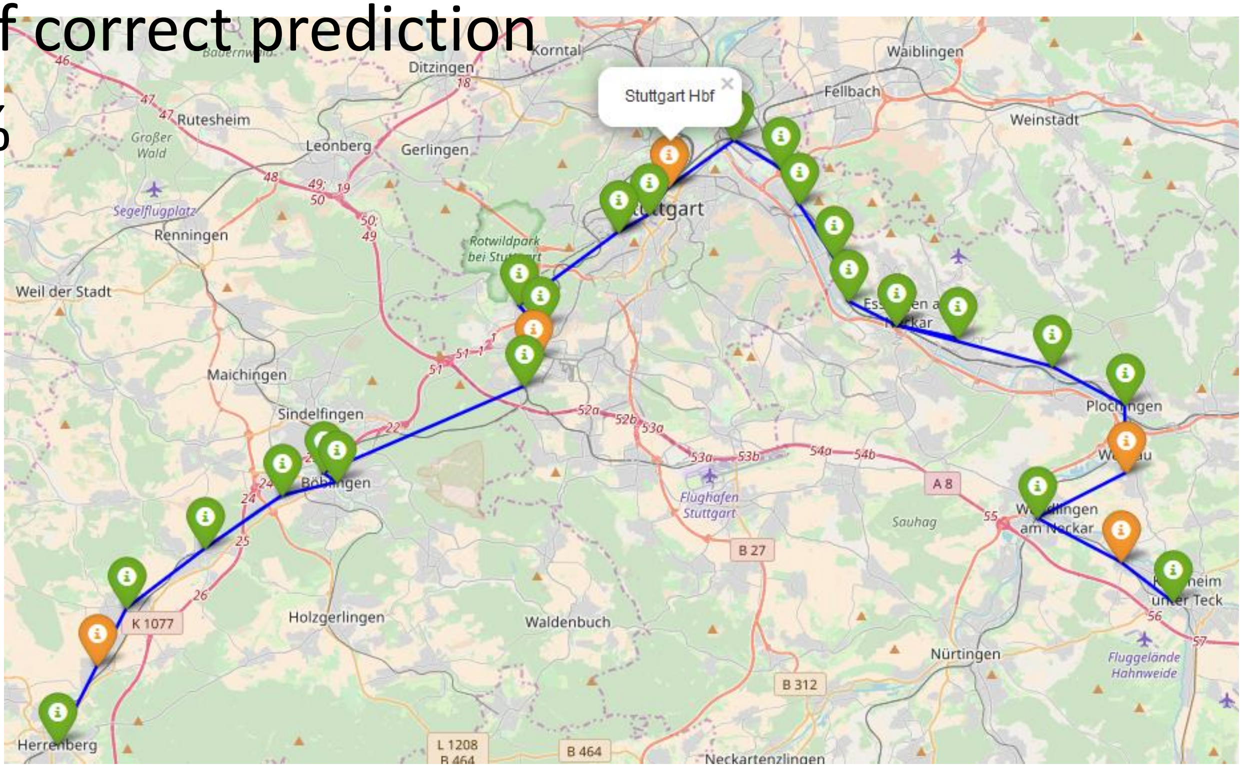
Predicting Train Delays with Machine Learning

In our **Supervised Learning** example:

- **Regression** to predict the continuous delay in minutes
  - Tools: (linear) regression models [PHB from p. 390].
  - Problems: Un-detected nonlinear relations, relevant features missing, …
  - Success guaranteed only at “short notice”.
- **Classification** of a binary discrete label (delay {yes, no}):
  - Tools: Random Forest as an ensemble of Decision Trees [PHB from p. 421].
  - … yields better results.

Example of classification result & visualisation

- **Accuracy** of classification of the line S1 at every station:
  - Green: > 80% of correct prediction
  - Orange: >= 50% (binary case = coin flip! ☹)
  - Black: < 50% ☹



Results obtained with Urika-GX, 90% training 10% test data(Q1/2020)

Focus on Pre-processing, Feature Engineering and Machine Learning

- Part I: Introduction
  - Pre-processing
    - « General Introduction
    - « Source Data

More learning outcomes

Main index

Part I
Part II
Part III

Source Data

In this section, we will cover:

- What are the source data for this example,
- What are the label and features for this example,
  - Small digression → ML/DL with engineering data.
- Which tools we use to run this example.

Source Data

Which **format** do the data have:

- *.csv* and *.xls* files are read-in from the S-Bahn code.
- You can find and download some of these files at https://fs.hlrs.de/projects/par/events/2021/DL3/S-Bahn-data/
- You can also have a look at them in your **workspace**:

```
> cd $MYSCR
> ll sbahn_data
```

Source Data

What do the **data represent**:

20170901-20171019_scheduled_S-Bahn_Stuttgart.csv

20170901-20171019_actual_S-Bahn_Stuttgart.csv

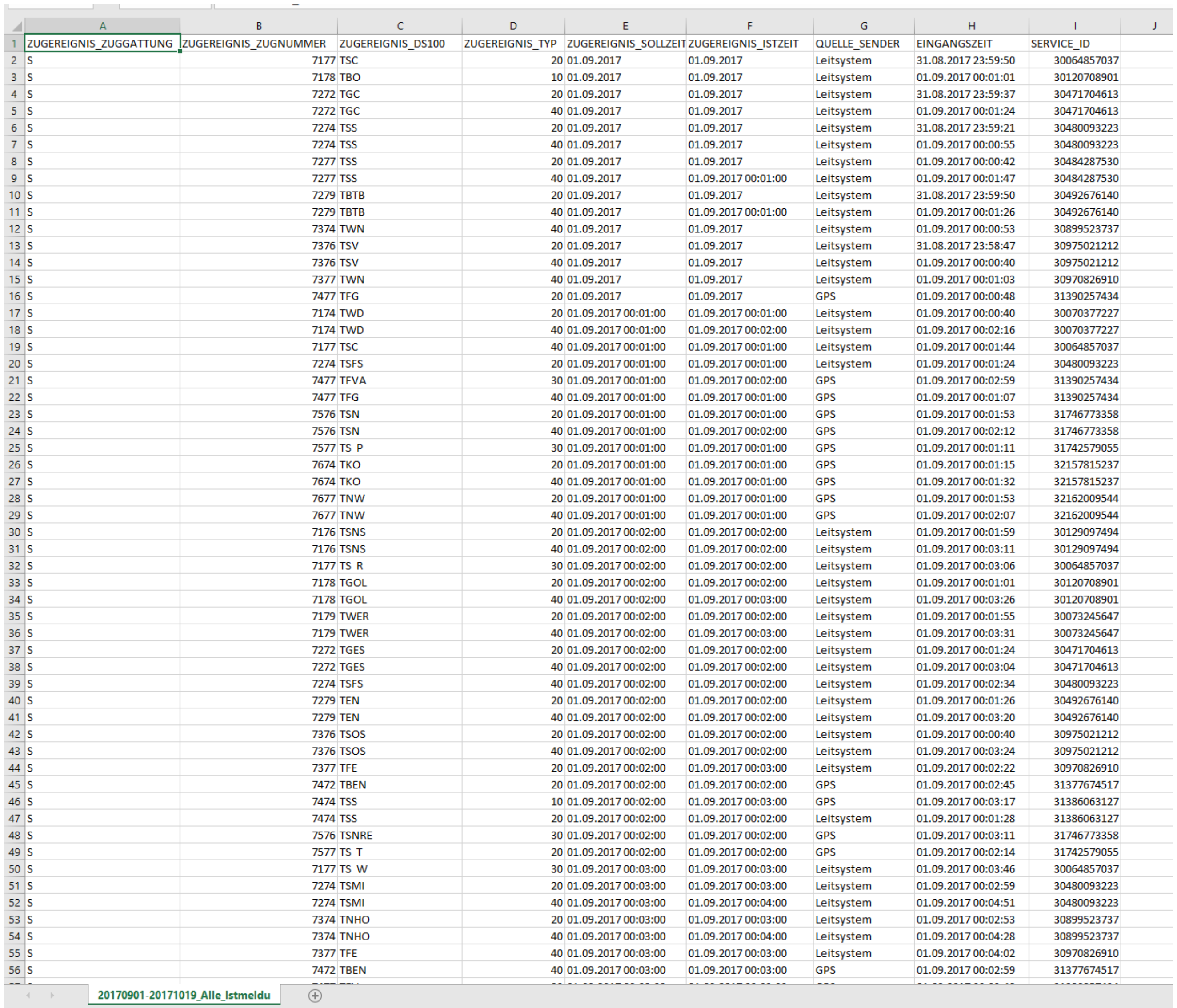
- All Stuttgart S-Bahn journeys over 50 days.

→ 1.639 journeys are analysed (complete dataset).

- S-Bahn events:
  - **scheduled timetable** (*Soll*-events),
  - **actual times** (*Ist*-events),

  each with information about the station, the line, the (planned) journey, …

https://data.deutschebahn.com/organization/s-bahn-stuttgart



	A	B	C	D	E	F	G	H	I	J
1	ZUGEREIGNIS_ZUGGATTUNG	ZUGEREIGNIS_ZUGNUMMER	ZUGEREIGNIS_DS100	ZUGEREIGNIS_TYP	ZUGEREIGNIS_SOLLZEIT	ZUGEREIGNIS_ISTZEIT	QUELLE_SENDER	EINGANGSZEIT	SERVICE_ID	
2	S	7177	TSC	20	01.09.2017	01.09.2017	Leitsystem	31.08.2017 23:59:50	30064857037	
3	S	7178	TBO	10	01.09.2017	01.09.2017	Leitsystem	01.09.2017 00:01:01	30120708901	
4	S	7272	TGC	20	01.09.2017	01.09.2017	Leitsystem	31.08.2017 23:59:37	30471704613	
5	S	7272	TGC	40	01.09.2017	01.09.2017	Leitsystem	01.09.2017 00:01:24	30471704613	
6	S	7274	TSS	20	01.09.2017	01.09.2017	Leitsystem	31.08.2017 23:59:21	30480093223	
7	S	7274	TSS	40	01.09.2017	01.09.2017	Leitsystem	01.09.2017 00:00:55	30480093223	
8	S	7277	TSS	20	01.09.2017	01.09.2017	Leitsystem	01.09.2017 00:00:42	30484287530	
9	S	7277	TSS	40	01.09.2017	01.09.2017 00:01:00	Leitsystem	01.09.2017 00:01:47	30484287530	
10	S	7279	TBTB	20	01.09.2017	01.09.2017	Leitsystem	31.08.2017 23:59:50	30492676140	
11	S	7279	TBTB	40	01.09.2017	01.09.2017 00:01:00	Leitsystem	01.09.2017 00:01:26	30492676140	
12	S	7374	TWN	40	01.09.2017	01.09.2017	Leitsystem	01.09.2017 00:00:53	30899523737	
13	S	7376	TSV	20	01.09.2017	01.09.2017	Leitsystem	31.08.2017 23:58:47	30975021212	
14	S	7376	TSV	40	01.09.2017	01.09.2017	Leitsystem	01.09.2017 00:00:40	30975021212	
15	S	7377	TWN	40	01.09.2017	01.09.2017	Leitsystem	01.09.2017 00:01:03	30970826910	
16	S	7477	TFG	20	01.09.2017	01.09.2017	GPS	01.09.2017 00:00:48	31390257434	
17	S	7174	TWD	20	01.09.2017 00:01:00	01.09.2017 00:01:00	Leitsystem	01.09.2017 00:00:40	30070377227	
18	S	7174	TWD	40	01.09.2017 00:01:00	01.09.2017 00:02:00	Leitsystem	01.09.2017 00:02:16	30070377227	
19	S	7177	TSC	40	01.09.2017 00:01:00	01.09.2017 00:01:00	Leitsystem	01.09.2017 00:01:44	30064857037	
20	S	7274	TSFS	20	01.09.2017 00:01:00	01.09.2017 00:01:00	Leitsystem	01.09.2017 00:01:24	30480093223	
21	S	7477	TFVA	30	01.09.2017 00:01:00	01.09.2017 00:02:00	GPS	01.09.2017 00:02:59	31390257434	
22	S	7477	TFG	40	01.09.2017 00:01:00	01.09.2017 00:01:00	GPS	01.09.2017 00:01:07	31390257434	
23	S	7576	TSN	20	01.09.2017 00:01:00	01.09.2017 00:01:00	GPS	01.09.2017 00:01:53	31746773358	
24	S	7576	TSN	40	01.09.2017 00:01:00	01.09.2017 00:02:00	GPS	01.09.2017 00:02:12	31746773358	
25	S	7577	TS P	30	01.09.2017 00:01:00	01.09.2017 00:01:00	GPS	01.09.2017 00:01:11	31742579055	
26	S	7674	TKO	20	01.09.2017 00:01:00	01.09.2017 00:01:00	GPS	01.09.2017 00:01:15	32157815237	
27	S	7674	TKO	40	01.09.2017 00:01:00	01.09.2017 00:01:00	GPS	01.09.2017 00:01:32	32157815237	
28	S	7677	TNW	20	01.09.2017 00:01:00	01.09.2017 00:01:00	GPS	01.09.2017 00:01:53	32162009544	
29	S	7677	TNW	40	01.09.2017 00:01:00	01.09.2017 00:01:00	GPS	01.09.2017 00:02:07	32162009544	
30	S	7176	TSNS	20	01.09.2017 00:02:00	01.09.2017 00:02:00	Leitsystem	01.09.2017 00:01:59	30129097494	
31	S	7176	TSNS	40	01.09.2017 00:02:00	01.09.2017 00:02:00	Leitsystem	01.09.2017 00:03:11	30129097494	
32	S	7177	TS R	30	01.09.2017 00:02:00	01.09.2017 00:02:00	Leitsystem	01.09.2017 00:03:06	30064857037	
33	S	7178	TGOL	20	01.09.2017 00:02:00	01.09.2017 00:02:00	Leitsystem	01.09.2017 00:01:01	30120708901	
34	S	7178	TGOL	40	01.09.2017 00:02:00	01.09.2017 00:03:00	Leitsystem	01.09.2017 00:03:26	30120708901	
35	S	7179	TWER	20	01.09.2017 00:02:00	01.09.2017 00:02:00	Leitsystem	01.09.2017 00:01:55	30073245647	
36	S	7179	TWER	40	01.09.2017 00:02:00	01.09.2017 00:03:00	Leitsystem	01.09.2017 00:03:31	30073245647	
37	S	7272	TGES	20	01.09.2017 00:02:00	01.09.2017 00:02:00	Leitsystem	01.09.2017 00:01:24	30471704613	
38	S	7272	TGES	40	01.09.2017 00:02:00	01.09.2017 00:02:00	Leitsystem	01.09.2017 00:03:04	30471704613	
39	S	7274	TSFS	40	01.09.2017 00:02:00	01.09.2017 00:02:00	Leitsystem	01.09.2017 00:02:34	30480093223	
40	S	7279	TEN	20	01.09.2017 00:02:00	01.09.2017 00:02:00	Leitsystem	01.09.2017 00:01:26	30492676140	
41	S	7279	TEN	40	01.09.2017 00:02:00	01.09.2017 00:02:00	Leitsystem	01.09.2017 00:03:20	30492676140	
42	S	7376	TSOS	20	01.09.2017 00:02:00	01.09.2017 00:02:00	Leitsystem	01.09.2017 00:00:40	30975021212	
43	S	7376	TSOS	40	01.09.2017 00:02:00	01.09.2017 00:02:00	Leitsystem	01.09.2017 00:03:24	30975021212	
44	S	7377	TFE	20	01.09.2017 00:02:00	01.09.2017 00:03:00	Leitsystem	01.09.2017 00:02:22	30970826910	
45	S	7472	TBEN	20	01.09.2017 00:02:00	01.09.2017 00:02:00	GPS	01.09.2017 00:02:45	31377674517	
46	S	7474	TSS	10	01.09.2017 00:02:00	01.09.2017 00:03:00	GPS	01.09.2017 00:03:17	31386063127	
47	S	7474	TSS	20	01.09.2017 00:02:00	01.09.2017 00:02:00	Leitsystem	01.09.2017 00:01:28	31386063127	
48	S	7576	TSNRE	30	01.09.2017 00:02:00	01.09.2017 00:02:00	GPS	01.09.2017 00:03:11	31746773358	
49	S	7577	TS T	20	01.09.2017 00:02:00	01.09.2017 00:02:00	GPS	01.09.2017 00:02:14	31742579055	
50	S	7177	TS W	30	01.09.2017 00:03:00	01.09.2017 00:03:00	Leitsystem	01.09.2017 00:03:46	30064857037	
51	S	7274	TSMI	20	01.09.2017 00:03:00	01.09.2017 00:03:00	Leitsystem	01.09.2017 00:02:59	30480093223	
52	S	7274	TSMI	40	01.09.2017 00:03:00	01.09.2017 00:04:00	Leitsystem	01.09.2017 00:04:51	30480093223	
53	S	7374	TNHO	20	01.09.2017 00:03:00	01.09.2017 00:03:00	Leitsystem	01.09.2017 00:02:53	30899523737	
54	S	7374	TNHO	40	01.09.2017 00:03:00	01.09.2017 00:04:00	Leitsystem	01.09.2017 00:04:28	30899523737	
55	S	7377	TFE	40	01.09.2017 00:03:00	01.09.2017 00:03:00	Leitsystem	01.09.2017 00:04:02	30970826910	
56	S	7472	TBEN	40	01.09.2017 00:03:00	01.09.2017 00:03:00	GPS	01.09.2017 00:02:59	31377674517	

20170901-20171019_Alle_Istmeldu

Source Data

StationData.csv

- All Stuttgart S-Bahn stations with their 3 attributes (numeric ID; short name as **DS100 code**; **full name**).

S-Bahn-coordinates.xlsx

- Geographical coordinates of the stations.

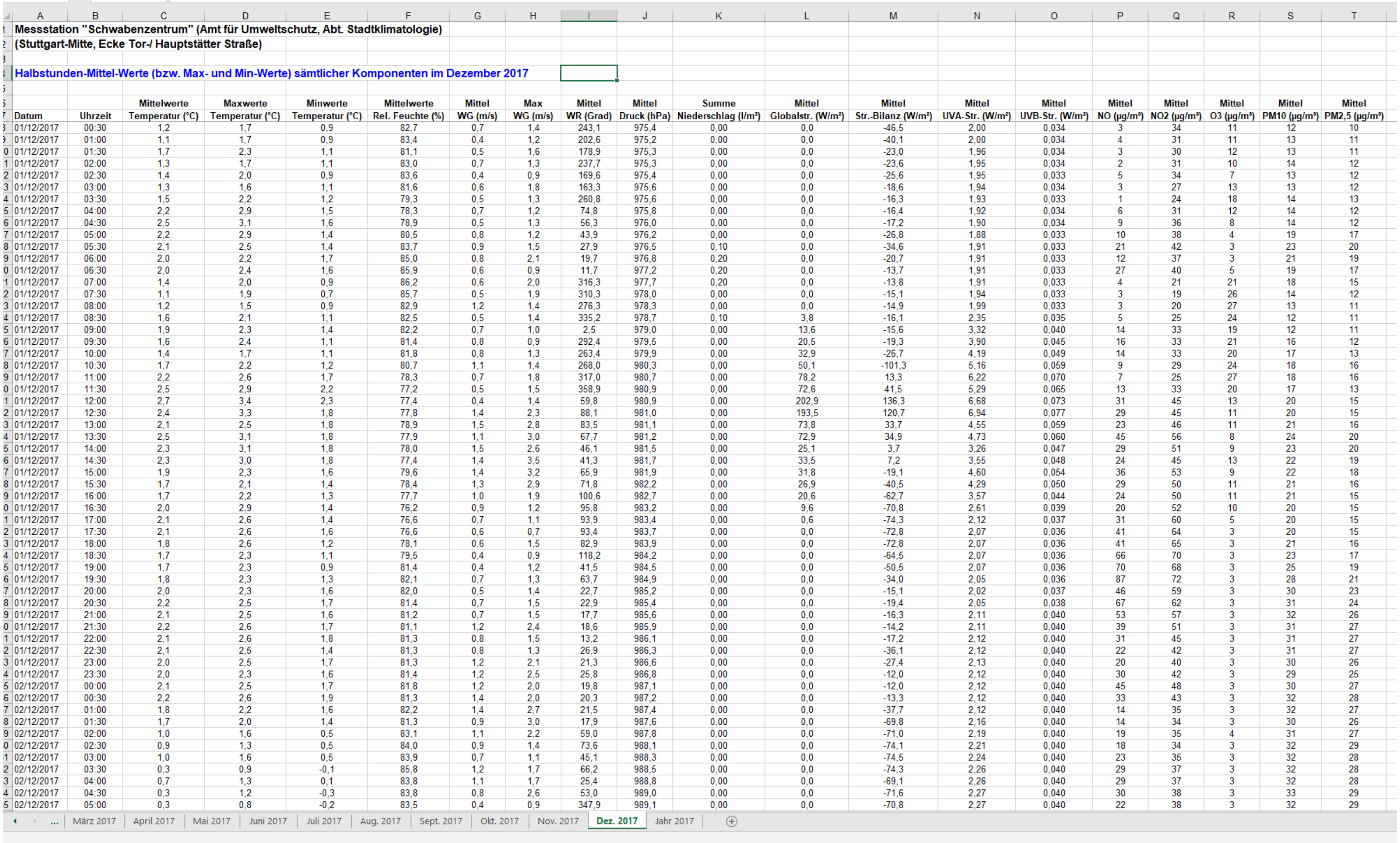
S-Mitte-SZ-30min-values_2017.xls

Picture next slide.

- **Weather** and **fine particles**, measured every 30 minutes.

Total size : O(1GB)

https://de.wikipedia.org/wiki/Betriebsstellenverzeichnis



Messstation "Schwabenzentrum" (Amt für Umweltschutz, Abt. Stadtklimatologie)
(Stuttgart-Mitte, Ecke Tor-/ Hauptstätter Straße)

Halbstunden-Mittel-Werte (bzw. Max- und Min-Werte) sämtlicher Komponenten im Dezember 2017

		Mittelwerte	Maxwerte	Minwerte	Mittelwerte	Mittel	Max	Mittel	Mittel	Summe	Mittel	Mittel	Mittel	Mittel	Mittel	Mittel	Mittel	Mittel	Mittel
Datum	Uhrzeit	Temperatur (°C)	Temperatur (°C)	Temperatur (°C)	Rel. Feuchte (%)	WG (m/s)	WG (m/s)	WR (Grad)	Druck (hPa)	Niederschlag (l/m²)	Globalstr. (W/m²)	Str.-Bilanz (W/m²)	UVA-Str. (W/m²)	UVB-Str. (W/m²)	NO (µg/m³)	NO2 (µg/m³)	O3 (µg/m³)	PM10 (µg/m³)	PM2,5 (µg/m³)
01/12/2017	00:30	1,2	1,7	0,9	82,7	0,7	1,4	243,1	975,4	0,00	0,0	-46,5	2,00	0,034	3	34	11	12	10
01/12/2017	01:00	1,1	1,7	0,9	83,4	0,4	1,2	202,6	975,2	0,00	0,0	-40,1	2,00	0,034	4	31	11	13	11
01/12/2017	01:30	1,7	2,3	1,1	81,1	0,5	1,6	178,9	975,3	0,00	0,0	-23,0	1,96	0,034	3	30	12	13	11
01/12/2017	02:00	1,3	1,7	1,1	83,0	0,7	1,3	237,7	975,3	0,00	0,0	-23,6	1,95	0,034	2	31	10	14	12
01/12/2017	02:30	1,4	2,0	0,9	83,6	0,4	0,9	169,6	975,4	0,00	0,0	-25,6	1,95	0,033	5	34	7	13	12
01/12/2017	03:00	1,3	1,6	1,1	81,6	0,6	1,8	163,3	975,6	0,00	0,0	-18,6	1,94	0,034	3	27	13	13	12
01/12/2017	03:30	1,5	2,2	1,2	79,3	0,5	1,3	260,8	975,6	0,00	0,0	-16,3	1,93	0,033	1	24	18	14	13
01/12/2017	04:00	2,2	2,9	1,5	78,3	0,7	1,2	74,8	975,8	0,00	0,0	-16,4	1,92	0,034	6	31	12	14	12
01/12/2017	04:30	2,5	3,1	1,6	78,9	0,5	1,3	56,3	976,0	0,00	0,0	-17,2	1,90	0,034	9	36	8	14	12
01/12/2017	05:00	2,2	2,9	1,4	80,5	0,8	1,2	43,9	976,2	0,00	0,0	-26,8	1,88	0,033	10	38	4	19	17
01/12/2017	05:30	2,1	2,5	1,4	83,7	0,9	1,5	27,9	976,5	0,10	0,0	-34,6	1,91	0,033	21	42	3	23	20
01/12/2017	06:00	2,0	2,2	1,7	85,0	0,8	2,1	19,7	976,8	0,20	0,0	-20,7	1,91	0,033	12	37	3	21	19
01/12/2017	06:30	2,0	2,4	1,6	85,9	0,6	0,9	11,7	977,2	0,20	0,0	-13,7	1,91	0,033	27	40	5	19	17
01/12/2017	07:00	1,4	2,0	0,9	86,2	0,6	2,0	316,3	977,7	0,20	0,0	-13,8	1,91	0,033	4	21	21	18	15
01/12/2017	07:30	1,1	1,9	0,7	85,7	0,5	1,9	310,3	978,0	0,00	0,0	-15,1	1,94	0,033	3	19	26	14	12
01/12/2017	08:00	1,2	1,5	0,9	82,9	1,2	1,4	276,3	978,3	0,00	0,0	-14,9	1,99	0,033	3	20	27	13	11
01/12/2017	08:30	1,6	2,1	1,1	82,5	0,5	1,4	335,2	978,7	0,10	3,8	-16,1	2,35	0,035	5	25	24	12	11
01/12/2017	09:00	1,9	2,3	1,4	82,2	0,7	1,0	2,5	979,0	0,00	13,6	-15,6	3,32	0,040	14	33	19	12	11
01/12/2017	09:30	1,6	2,4	1,1	81,4	0,8	0,9	292,4	979,5	0,00	20,5	-19,3	3,90	0,045	16	33	21	16	12
01/12/2017	10:00	1,4	1,7	1,1	81,8	0,8	1,3	263,4	979,9	0,00	32,9	-26,7	4,19	0,049	14	33	20	17	13
01/12/2017	10:30	1,7	2,2	1,2	80,7	1,1	1,4	268,0	980,3	0,00	50,1	-101,3	5,16	0,059	9	29	24	18	16
01/12/2017	11:00	2,2	2,6	1,7	78,3	0,7	1,8	317,0	980,7	0,00	78,2	13,3	6,22	0,070	7	25	27	18	16
01/12/2017	11:30	2,5	2,9	2,2	77,2	0,5	1,5	358,9	980,9	0,00	72,6	41,5	5,29	0,065	13	33	20	17	13
01/12/2017	12:00	2,7	3,4	2,3	77,4	0,4	1,4	59,8	980,9	0,00	202,9	136,3	6,68	0,073	31	45	13	20	15
01/12/2017	12:30	2,4	3,3	1,8	77,8	1,4	2,3	88,1	981,0	0,00	193,5	120,7	6,94	0,077	29	45	11	20	15
01/12/2017	13:00	2,1	2,5	1,8	78,9	1,5	2,8	83,5	981,1	0,00	73,8	33,7	4,55	0,059	23	46	11	21	16
01/12/2017	13:30	2,5	3,1	1,8	77,9	1,1	3,0	67,7	981,2	0,00	72,9	34,9	4,73	0,060	45	56	8	24	20
01/12/2017	14:00	2,3	3,1	1,8	78,0	1,5	2,6	46,1	981,5	0,00	25,1	3,7	3,26	0,047	29	51	9	23	20
01/12/2017	14:30	2,3	3,0	1,8	77,4	1,4	3,5	41,3	981,7	0,00	33,5	7,2	3,55	0,048	24	45	13	22	19
01/12/2017	15:00	1,9	2,3	1,6	79,6	1,4	3,2	65,9	981,9	0,00	31,8	-19,1	4,60	0,054	36	53	9	22	18
01/12/2017	15:30	1,7	2,1	1,4	78,4	1,3	2,9	71,8	982,2	0,00	26,9	-40,5	4,29	0,050	29	50	11	21	16
01/12/2017	16:00	1,7	2,2	1,3	77,7	1,0	1,9	100,6	982,7	0,00	20,6	-62,7	3,57	0,044	24	50	11	21	15
01/12/2017	16:30	2,0	2,9	1,4	76,2	0,9	1,2	95,8	983,2	0,00	9,6	-70,8	2,61	0,039	20	52	10	20	15
01/12/2017	17:00	2,1	2,6	1,4	76,6	0,7	1,1	93,9	983,4	0,00	0,6	-74,3	2,12	0,037	31	60	5	20	15
01/12/2017	17:30	2,1	2,6	1,6	76,6	0,6	0,7	93,4	983,7	0,00	0,0	-72,8	2,07	0,036	41	64	3	20	15
01/12/2017	18:00	1,8	2,6	1,2	78,1	0,6	1,5	82,9	983,9	0,00	0,0	-72,8	2,07	0,036	41	65	3	21	16
01/12/2017	18:30	1,7	2,3	1,1	79,5	0,4	0,9	118,2	984,2	0,00	0,0	-64,5	2,07	0,036	66	70	3	23	17
01/12/2017	19:00	1,7	2,3	0,9	81,4	0,4	1,2	41,5	984,5	0,00	0,0	-50,5	2,07	0,036	70	68	3	25	19
01/12/2017	19:30	1,8	2,3	1,3	82,1	0,7	1,3	63,7	984,9	0,00	0,0	-34,0	2,05	0,036	87	72	3	28	21
01/12/2017	20:00	2,0	2,3	1,6	82,0	0,5	1,4	22,7	985,2	0,00	0,0	-15,1	2,02	0,037	46	59	3	30	23
01/12/2017	20:30	2,2	2,5	1,7	81,4	0,7	1,5	22,9	985,4	0,00	0,0	-19,4	2,05	0,038	67	62	3	31	24
01/12/2017	21:00	2,1	2,5	1,6	81,2	0,7	1,5	17,7	985,6	0,00	0,0	-16,3	2,11	0,040	53	57	3	32	26
01/12/2017	21:30	2,2	2,6	1,7	81,1	1,2	2,4	18,6	985,9	0,00	0,0	-14,2	2,11	0,040	39	51	3	31	27
01/12/2017	22:00	2,1	2,6	1,8	81,3	0,8	1,5	13,2	986,1	0,00	0,0	-17,2	2,12	0,040	31	45	3	31	27
01/12/2017	22:30	2,1	2,5	1,4	81,3	0,8	1,3	26,9	986,3	0,00	0,0	-36,1	2,12	0,040	22	42	3	31	27
01/12/2017	23:00	2,0	2,5	1,7	81,3	1,2	2,1	21,3	986,6	0,00	0,0	-27,4	2,13	0,040	20	40	3	30	26
01/12/2017	23:30	2,0	2,3	1,6	81,4	1,2	2,5	25,8	986,8	0,00	0,0	-12,0	2,12	0,040	30	42	3	29	25
02/12/2017	00:00	2,1	2,5	1,7	81,8	1,2	2,0	19,8	987,1	0,00	0,0	-12,0	2,12	0,040	45	48	3	30	27
02/12/2017	00:30	2,2	2,6	1,9	81,3	1,4	2,0	20,3	987,2	0,00	0,0	-13,3	2,12	0,040	33	43	3	32	28
02/12/2017	01:00	1,8	2,2	1,6	82,2	1,4	2,7	21,5	987,4	0,00	0,0	-37,7	2,12	0,040	14	35	3	32	27
02/12/2017	01:30	1,7	2,0	1,4	81,3	0,9	3,0	17,9	987,6	0,00	0,0	-69,8	2,16	0,040	14	34	3	30	26
02/12/2017	02:00	1,0	1,6	0,5	83,1	1,1	2,2	59,0	987,8	0,00	0,0	-71,0	2,19	0,040	19	35	4	31	27
02/12/2017	02:30	0,9	1,3	0,5	84,0	0,9	1,4	73,6	988,1	0,00	0,0	-74,1	2,21	0,040	18	34	3	32	29
02/12/2017	03:00	1,0	1,6	0,5	83,9	0,7	1,1	45,1	988,3	0,00	0,0	-74,5	2,24	0,040	23	35	3	32	28
02/12/2017	03:30	0,3	0,9	-0,1	85,8	1,2	1,7	66,2	988,5	0,00	0,0	-74,3	2,26	0,040	29	37	3	32	28
02/12/2017	04:00	0,7	1,3	0,1	83,8	1,1	1,7	25,4	988,8	0,00	0,0	-69,1	2,26	0,040	29	37	3	32	28
02/12/2017	04:30	0,3	1,2	-0,3	83,8	0,8	2,6	53,0	989,0	0,00	0,0	-71,6	2,27	0,040	30	38	3	33	29
02/12/2017	05:00	0,3	0,8	-0,2	83,5	0,4	0,9	347,9	989,1	0,00	0,0	-70,8	2,27	0,040	22	38	3	32	29

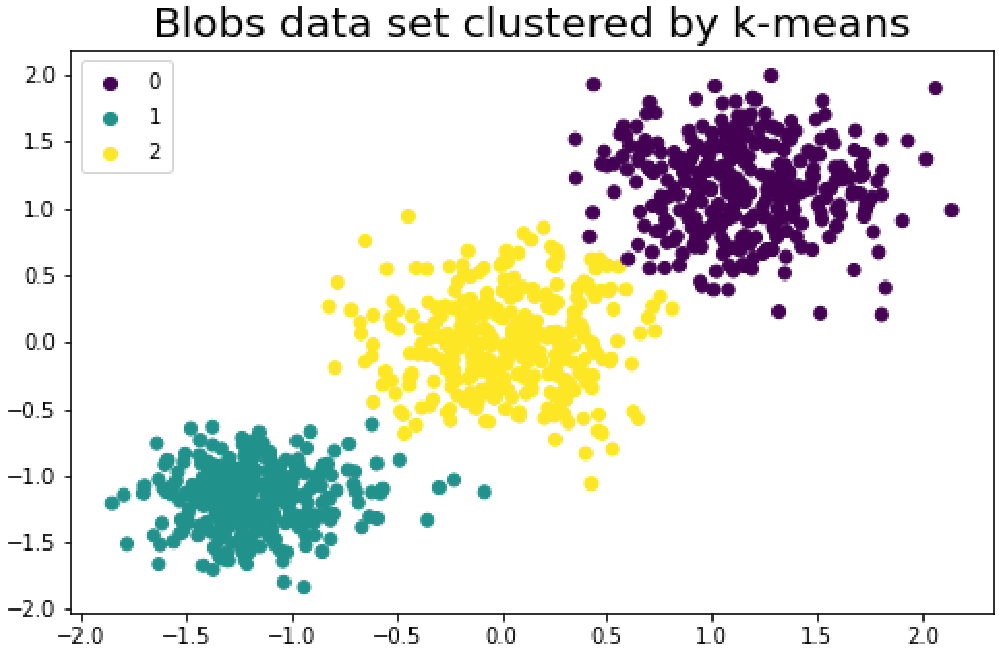
Source Data

- During the manipulation part, the data will be structured into **DataFrames (DF)** for the ML algorithms.
- Each row of the DF is a **sample** corresponding to an **event**:
  - A train's departure from a particular station:
    1.639 journeys x ca. 20 stations / line = ca. **33.000 events**,
  - each characterised by **one label** (= **delay** of the event),
  - … and **> 30 features** (about time, weather, station characteristics):
    33.000 events x 32 qualifiers = ca. **1M data** in different formats.

Source Data

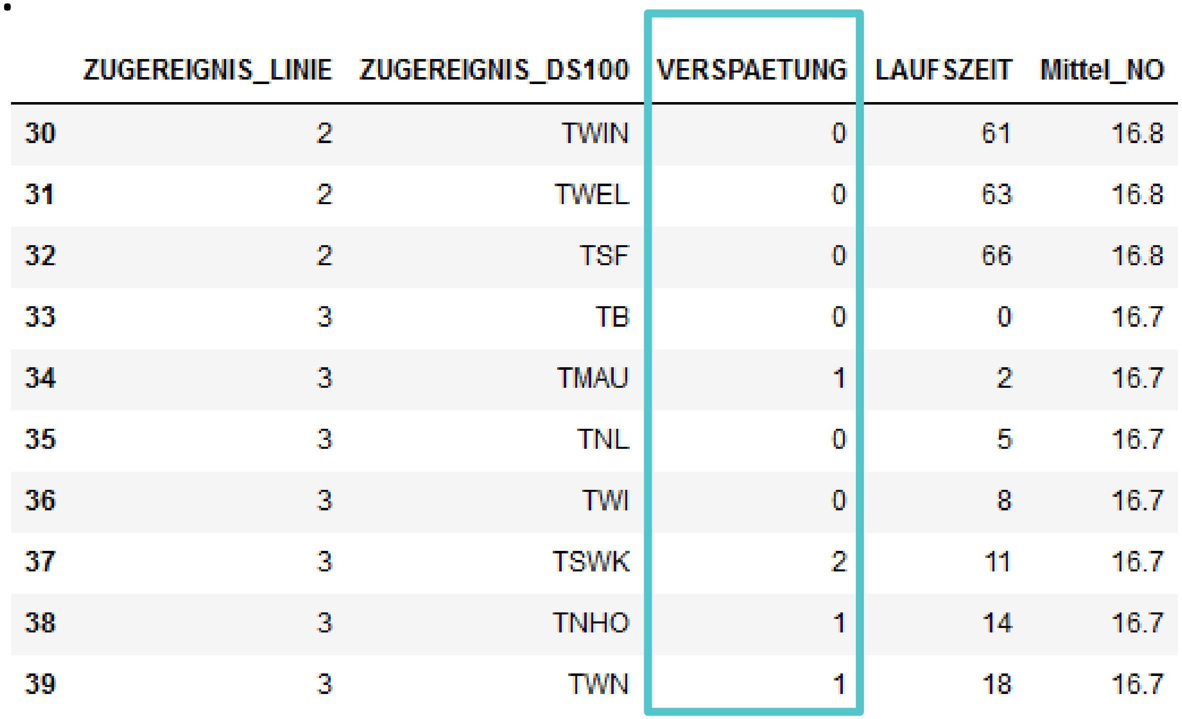
- From the source data, **31 features** and **1 label** are derived in the pre-processing phase:
  - **Features**: E.g. (x,y) positions of points on the plane.
  - **Label** or **target**: E.g. colour of these points.

  Image recognition (day 2)
  - **Features**: **clusters of pixels**,
  - **Label** or **target**: information from the picture.



Source Data

- The features are **columns** of the DataFrames used for the ML algorithms to predict the **delay** of the S-Bahn.
- See the next 3 slides for a **list of all features**, and the **label**, i.e. the delay.



	ZUGEREIGNIS_LINIE	ZUGEREIGNIS_DS100	VERSPAETUNG	LAUFSZEIT	Mittel_NO
30	2	TWIN	0	61	16.8
31	2	TWEL	0	63	16.8
32	2	TSF	0	66	16.8
33	3	TB	0	0	16.7
34	3	TMAU	1	2	16.7
35	3	TNL	0	5	16.7
36	3	TWI	0	8	16.7
37	3	TSWK	2	11	16.7
38	3	TNHO	1	14	16.7
39	3	TWN	1	18	16.7

Data preparation: feature columns (1)

|-- MONAT: **month**

German name in the DataFrame

|-- TAG: **day**

|-- STUNDE_SOLL: **sched. event hour**

English version

|-- MINUTE_RANGE: **0 or 30**

|-- ZUGEREIGNIS_LINIE: **S-Bahn line number**

|-- SERVICE_START_ZEIT: **unix time for scheduled service start**

★ |-- ZUGEREIGNIS_ISTZEIT: **unix time for actual event**

|-- ZUGEREIGNIS_TYP: **40=departure**

★ |-- ZUGEREIGNIS_SOLLZEIT: **unix time for scheduled event**

|-- SERVICE_ID: **service unique ID**

|-- ZUGEREIGNIS_ZUGNUMMER: **train number**

|-- ZUGEREIGNIS_DS100: **station code**

Some selected relevant features

Data preparation: feature columns (2)

|-- MINUTE_SOLL: **scheduled minute**

★|-- STUNDE_SER: **scheduled service start hour**

|-- MINUTE_SER: **scheduled service start minute**

|-- WOCHENTAG_SER: **scheduled service start day of week**

|-- WOCHENTAG_SOLL: **scheduled day of week**

|-- VERSPAETUNG: **delay (minutes)** ← Label or target

|-- LAUFSZEIT**: diff. sched. event - sched. service start (min.)**

|-- Datum: **date**

★|-- prev0_VERSPAETUNG: **delay (minutes) at the station -1**

|-- prev1_prev0_VERSPAETUNG: **delay (minutes) at the station -2**

Data preparation: feature columns (3)

|-- Mittelwerte_Temperatur: **temperature**

|-- Mittelwerte_Rel_Feuchte: **humidity**

|-- Mittel_WG: **avg. wind speed**

|-- Max_WG: **max wind speed**

★|-- Summe_Niederschlag: **total precipitation** (as L/m^2)

|-- Mittel_NO: **avg. pollution data…**

|-- Mittel_NO2: **…**

|-- Mittel_O3: **…**

|-- Mittel_PM10:

|-- TimeUnix: **unix time of the weather forecast**

Source Data: Engineering

Alternative data sources:

- They could be the result of a **numerical simulation**,
- or of several FEM simulations (**bundle**) with varying parameters.
- An example of **data augmentation** is combining
  - representative data such as measurements and experiments **+**
  - simulation and computation (e.g., when extreme scenarios must be included).
- For time-dependent problems:
  - **ML models**: Methods to extract input data,
  - In **DL**: Sequence models or recurrent NN.

Details skipped

Source Data: Engineering

→ **Unsupervised learning** methods for:

- compact representation / model surrogates
  → dimension reduction
- automatic categorisation of controller results → clustering

… for improved (**human**) interpretability of the results (e.g., detect anomalies).

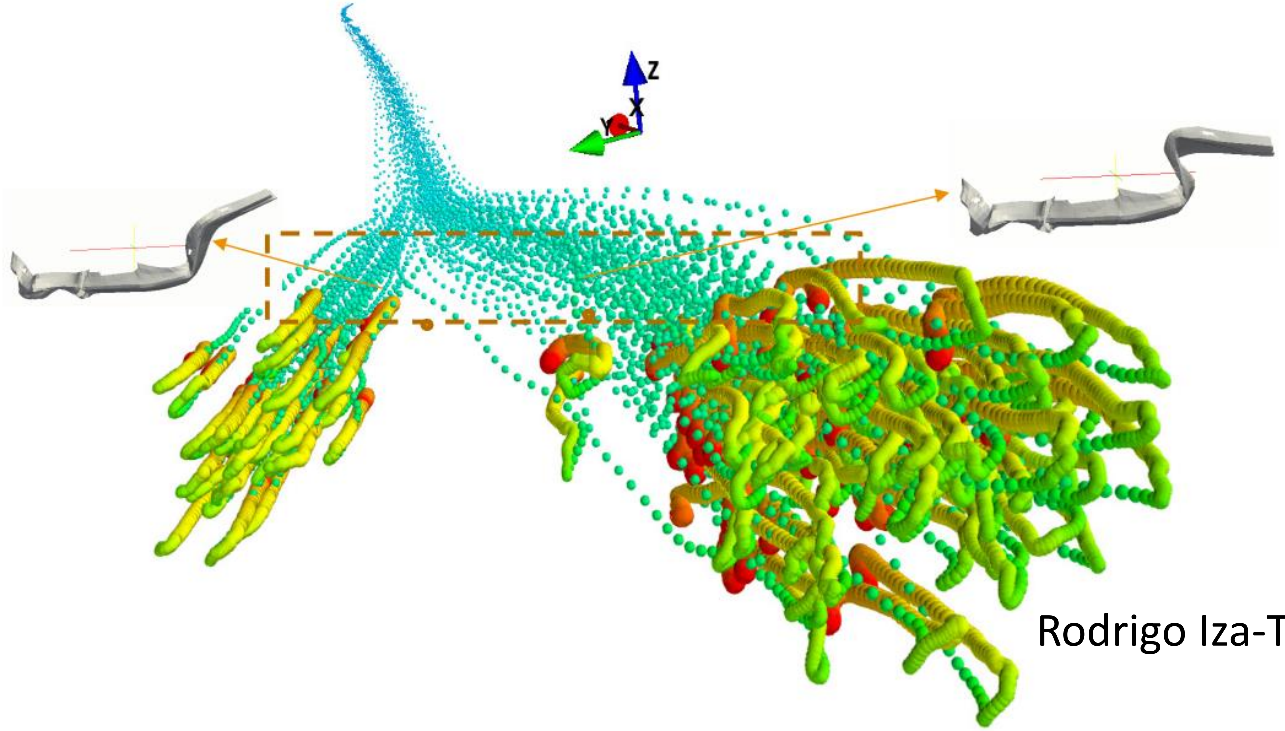
→ Also **supervised learning** methods play a role
E.g. day 3: shock detection in a turbulence model → binary classifier

Beck, Flad, Munz. "Deep neural networks for data-driven LES closure models." Journal of Computational Physics 398 (2019): 108910.
Kurz, Beck. "A machine learning framework for LES closure terms", arXiv, 2020.
Beck, Kurz. "A perspective on machine learning methods in turbulence modeling", GAMM Mitteilungen, 2021.

Source Data: Engineering

Example:

Dimension reduction of an automotive FEM simulation (3 modes), and **clustering** (= colour, quantity of interest).



https://www.hlrs.de/training/2024/ml4simx

EXCELLERAT P2

Rodrigo Iza-Teran / Fraunhofer SCAI

Source Data: Engineering

Some challenges:

- Coupling of engineering code (C, C++, Fortran) with ML/DL libraries and instructions (Python).
- Data storage.
- HPC: data- and model-parallelism on CPU and GPUs.
- …

Data preparation: Tools

Programming language:

- Python:
  https://docs.python.org/3/tutorial/

Main **tools** for data **manipulation** and **ML**:

- Numpy:
  Efficient interface to store and operate on **dense data buffers**:
  https://numpy.org/doc/stable/reference/index.html

Data preparation: Tools

- **Apache Spark**
  Analytics and ML framework released in 2014:
  - Originally from Berkeley AMPLab/BDAS stack, now Apache project.
  - Native APIs in Scala; Java, **Python**, and R APIs available as well.
    https://spark.apache.org

Data preparation: Tools

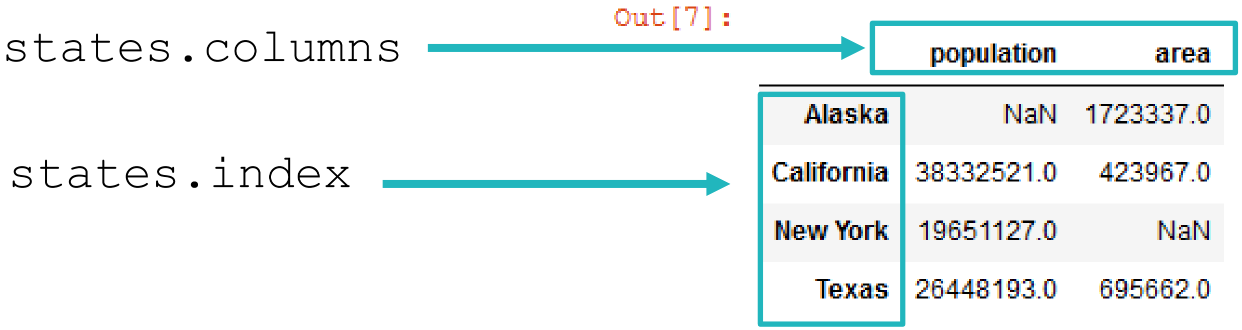
*“Spark is a **fast and general cluster computing system for Big Data**. It provides **high-level APIs** in Scala, Java, **Python**, and R, and an optimized engine that supports general computation graphs for data analysis. It also supports a rich set of higher-level tools including **Spark SQL for** SQL and **DataFrames, MLlib for machine learning**, GraphX for graph processing, and Spark Streaming for stream processing.“*

https://pypi.org/project/pyspark/

Compare to Scikit-learn! Similar tools with some differences: Article.

Data preparation: Tools

- **Pandas**
  https://pandas.pydata.org
  Data Manipulation with Pandas [**PHB pp. 97-215**]
  - Built on Numpy,
  - Provides useful structures for ML such as **Series** and **DataFrames**.
  - **DataFrames** (DFs):
    2D objects with flexible **row** and **column indices**:



states.columns

states.index

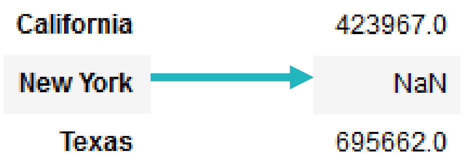
Out[7]:

	population	area
Alaska	NaN	1723337.0
California	38332521.0	423967.0
New York	19651127.0	NaN
Texas	26448193.0	695662.0

Data preparation: Basic Structures

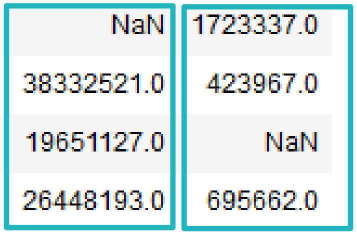
Other (simpler) data structures (cf. **EX 1** in **NB 1**):

- Python **dictionary**:
  Maps keys to values of **arbitrary** type.



California	423967.0
New York	NaN
Texas	695662.0

- Numpy **array [PHB pp. 33-96]**:
  Multi-dimensional **typed** Python array.

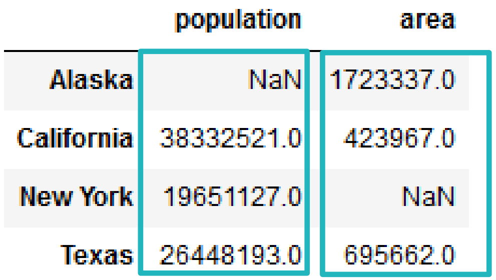


NaN	1723337.0
38332521.0	423967.0
19651127.0	NaN
26448193.0	695662.0

Data preparation: Basic Structures

- Pandas **Series [PHB pp. 97-110]**:
  1D-array of indexed, **typed** data with flexible **indices**.
  More efficient than a dictionary!

- Pandas **DataFrames**:
  Aligned Series sharing the **same row index:**
  `states.index` indices of the rows,
  `states.columns` indices of the columns.



	population	area
Alaska	NaN	1723337.0
California	38332521.0	423967.0
New York	19651127.0	NaN
Texas	26448193.0	695662.0

DataFrames support a variety of different data types (vectors, text, images and structured data) on which ML algorithms can be applied.

https://spark.apache.org/docs/latest/ml-pipeline.html

Data preparation: Tools

Why use **both Spark and Pandas**?

- Pandas*:*
  User-friendly, more flexible, better for visualisation.
- Spark: *Better for parallelism.*

→ In the **exercises**:

- **Spark** is the **default** framework,
- **Pandas** DataFrames will be marked with **_pd** *.*
- Most commands are quite intuitive. Helping text is provided.
  → Proficiency in these tools is not required.

Source Data and I/O: HDFS

HDFS was available on Urika GX. Vulcan: Work in progress.

- I/O files are (usually large) DataFrames which are read from and written to through the Hadoop Distributed File System (**HDFS**).
- HDFS allows for distributed storage in an **HDFS cluster** (provided in the Urika-GX systems).

+ : Speed of **parallel** execution

– : HDFS files cannot be handled as “normal“ files.

See:

https://hadoop.apache.org/docs/r2.8.5/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html

Source Data and I/O: Small HDFS guide

Basic commands to perform operations on these files:

https://hadoop.apache.org/docs/r2.8.5/hadoop-project-dist/hadoop-hdfs/HDFSCommands.html

and

https://hadoop.apache.org/docs/r2.8.5/hadoop-project-dist/hadoop-common/FileSystemShell.html

PRACTICAL (e.g. on the Urika system)

In particular:

```
> hadoop fs -ls
```

displays your local hdfs files (none or Trash folder).

Source Data and I/O: Small HDFS guide

PRACTICAL (e.g. on the Urika system)

Copy an HDFS file to your directory:

```
> hadoop fs -cp hdfs://192.168.0.1:8020/user/hpclzano/df_ts.csv df_ts2.csv
```

You can now see the file with (in chronological order)

```
> hadoop fs -ls -t
```

Check out more -ls options in https://hadoop.apache.org/docs/r2.8.5/hadoop-project-dist/hadoop-common/FileSystemShell.html

Details skipped

Status 2021

Source Data and I/O: Small HDFS guide

PRACTICAL (e.g. on the Urika system)

Remove the df_ts2.csv file with: hadoop fs -rm **[name_of_file]**

```
> hadoop fs -rm df_ts2.csv
```

ERROR! since hdfs files are treated like **directories**.

In fact:

```
> hadoop fs -ls df_ts2.csv
```

… shows?

Details skipped

Source Data and I/O: Small HDFS guide

PRACTICAL (e.g. on the Urika system)

… it shows the many smaller parts in which the file is now distributed, now as files:

```
…
-rw-r--r--   3 hpclzano s29931     470189 2020-03-18 14:28
df_proper.csv/part-… .csv
…
```

To properly remove the file:

```
>  hadoop  fs  -rm  -R  df_ts2.csv
```

Details skipped

Source Data and I/O: Small HDFS guide

Later in **Spark**, I/O is done in such a way:

```
df_test5_5.write.mode('overwrite').csv('df_test5_5.csv', header =
True)
```

```
df_train_classification = spark.read.option('header',
True).option('inferSchema', True)\
.csv('df_train_classification.csv').cache()
```

Writing/reading are done locally by default to/from HDFS files!

-> **Manually adapted** to Lustre I/O.

Requirements for the hands-on exercises

- **Login**: Sheet on your desk.
- This & all other files can be found in:

https://fs.hlrs.de/projects/par/events/2024/dl-hlrs

- **DL-HLRS-all-exercises.pdf** :
  - ➢ Create the workspace,
  - ➢ Download the data needed for all days (?).
- **DL-HLRS-day1-lectures.pdf** : **These slides**
- **DL-HLRS-day1-exercises.pdf** : **Exercise instructions**

We do this set-up now.
Careful with c/p from pdf!
Use right-click in terminal.

Focus on Pre-processing, Feature Engineering and Machine Learning

- Part II: Example on the Jupyter Notebooks
  - Pre-processing
  - Supervised learning techniques in a Machine Learning pipeline
    - « Notebook 1 (Ex.)
    - « Notebook 2 (Ex.)
    - « Notebook 3 (Ex.)
    - « Notebook 4 (Ex.)
    - « Notebook 5 (Ex.)

More learning outcomes

Main index

Part I
Part II
Part III

Notebook 1: **NB1_manipulation.ipynb**

- Jupyter Notebook how-to, see slides: https://fs.hlrs.de/projects/par/events/2024/dl-hlrs/DL-HLRS-day1-exercises.pdf
- A few slides to sum up the content of **this** Notebook follow.

Data preparation: Basic Structures exercise

EX 1:

Recap of basic structures:

Pandas DataFrame

↓

Pandas Series

↓

Dictionary

Data preparation: Objectives

- **Feature engineering**:
  - Deal with missing data, outliers, NaNs, …
  - Obtain derived features that have an impact on the model.
  - Vectorisation → NB 2, 3, 4
- **Explorative Data Analysis** → NB 2
  - Apply statistical tools
  - Data interpretation
  - Find out existing correlations

Data preparation

1. Weather data:

- We start with a .csv file:
  - Read in the file.
  - Create a usable Pandas DataFrame df_pd_weather (**EX 2**).

- Manipulation of the Pandas DataFrame:
  Drop columns, delete or replace rows with damaged entries (**EX 3**).

- The Pandas DataFrame is converted into a Spark DataFrame (df_weather), then manipulated further (**EX 4**).

Data preparation

2. S-Bahn data:

- They are split into
  - *“act“* (actual or real, containing delayed journeys), and
  - *“sched“* (scheduled, timetable) data.
- We merge them into **one** unique Spark DataFrame (**df_all** : **EX 5**).
- First manipulation of this Spark DataFrame : **EX 6**.

Data preparation

- **EX 7-8: Time operations**:

  E.g.: *2017-09-01 00:30:00*

  – Transform all “*act*“ and “*sched*“ times into **Unix-times**:

  – Unix-time = number of seconds since 01/01/1970:

  1.504.218.600

  → Unix-time allows to perform operations on the time entries:

  → e.g. compute the **delay** and the **duration** of travel.

https://www.unixtimestamp.com

Data preparation

- **EX 7-8: Time operations**:

  1.504.218.600

  – Further new features can be obtained, such as
  → month, day of week : *September*, *Friday*
  → weekday/-end, peak hour, holiday, which season, …
  are relevant to predict the delay!

https://www.unixtimestamp.com

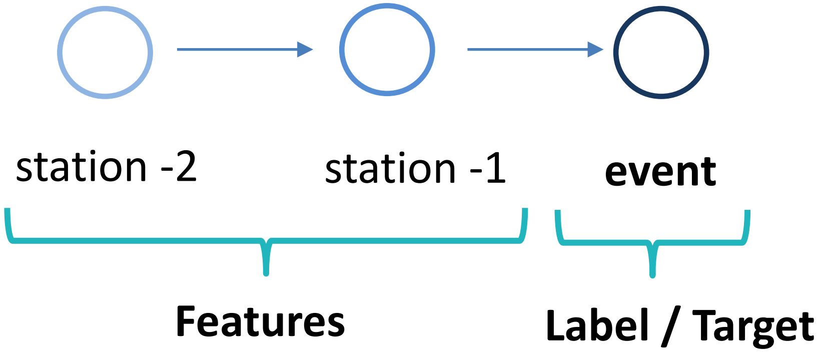
Data preparation

3. Merge **train and weather** data:

- Create a merged DataFrame (**df_proper**).
- Filter out delays that are not considered (**EX 9**):
  - negative delays (= train is early),
  - delays more than 3 hours (outliers).
- Add **delay** at stations -1 and -2, and create a new DataFrame (**df_ts**) with this information:

+ : Much higher probability of predicting the delay.

– : This information is available only **at very short notice**!



Data preparation

- Generate **df_ts_classification** (**EX 10**):
  - It includes a binary **{0, 1}** column = delay **{no, yes}**.
  - This is the label / target for the **classification algorithm**.
  - The threshold is set to 0 minutes, i.e. **all** delays are included.

Advanced methods to define the threshold in classification problems in DL (computer vision applications), e.g. the Area Under the Curve (AUC) score calculated from the Receiver Operating Characteristic (ROC).

Details skipped

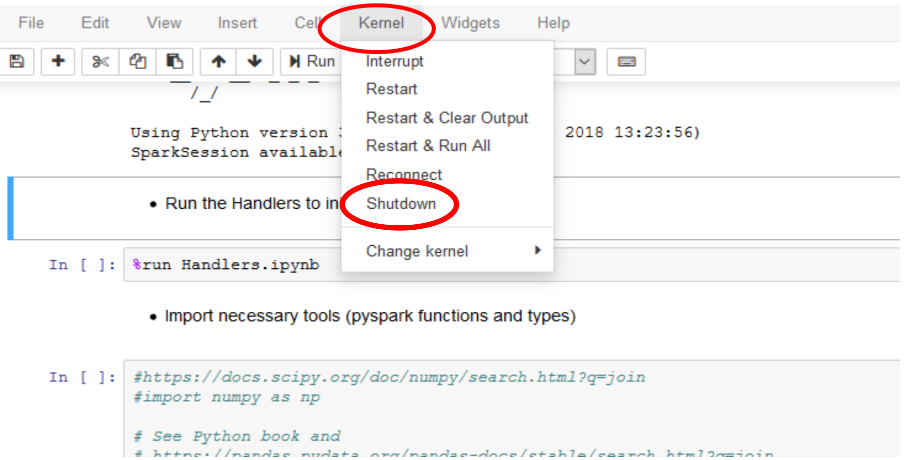
Data preparation

4. Split the DataFrames into **training** and **test**:

- Split df_ts and df_ts_classification for the ML pipeline later (using **random split**, **EX 11**):
  df_train, **df_train_classification** → 50% training
  df_test, **df_test_classification** → 50% test
- The **seed** for random split has been fixed at the beginning (to have reproducible results).

Hands-on

- Execute all **NB1_manipulation.ipynb** (Notebook 1):
  - EX1 → introduction
  - … until EX4: Pandas → "basic"
  - … until EX11: Spark → "advanced"
- Remember to shut down the kernel at the end.



```
   /_/

Using Python version                  2018 13:23:56)
SparkSession availabl
```

- Run the Handlers to in

```
In [ ]: %run Handlers.ipynb
```

- Import necessary tools (pyspark functions and types)

```
In [ ]: #https://docs.scipy.org/doc/numpy/search.html?q=join
        #import numpy as np

        # See Python book and
        # https://pandas.pydata.org/pandas-docs/stable/search.html?q=join
```

Focus on Pre-processing, Feature Engineering and Machine Learning

- Part II: Example on the Jupyter Notebooks
  - Pre-processing
  - Supervised learning techniques in a Machine Learning pipeline
    - « Notebook 1 (Ex.)
    - « Notebook 2 (Ex.)
    - « Notebook 3 (Ex.)
    - « Notebook 4 (Ex.)
    - « Notebook 5 (Ex.)

More learning outcomes

Main index

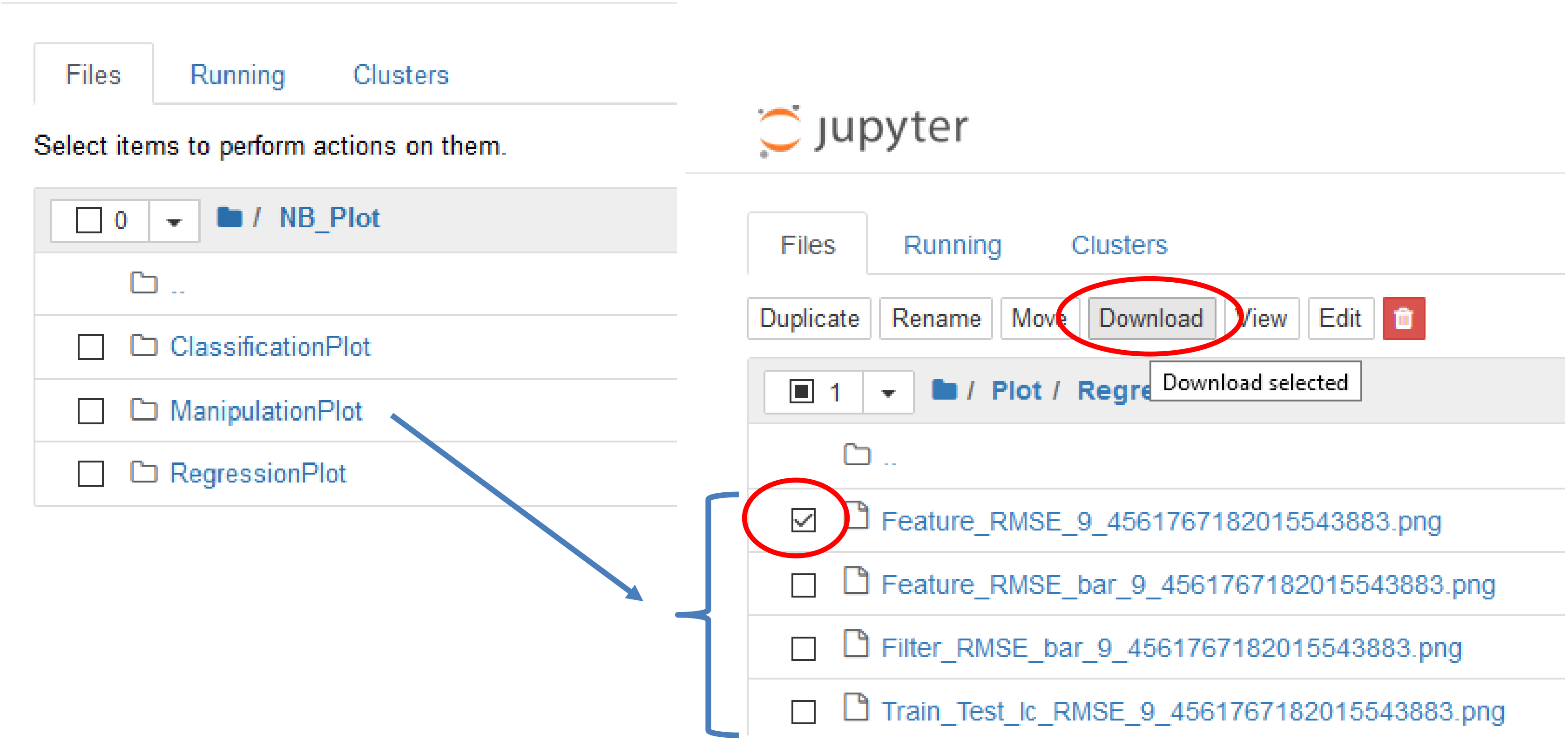
Part I
Part II
Part III

Notebook 2: **NB2_vis-man.ipynb**

- Jupyter Notebook how-to, see slides: https://fs.hlrs.de/projects/par/events/2024/dl-hlrs/DL-HLRS-day1-exercises.pdf
- A few slides to sum up the content of **this** Notebook follow.

JN: AFTER the exercise

You can download any created plot in the plot folder and sub-folders:



Files Running Clusters

Select items to perform actions on them.

□ 0 ▾ / NB_Plot

..

□ ClassificationPlot

□ ManipulationPlot

□ RegressionPlot

jupyter

Files Running Clusters

Duplicate Rename Move Download View Edit

Download selected

▣ 1 ▾ / Plot / Regre

..

☑ Feature_RMSE_9_4561767182015543883.png

□ Feature_RMSE_bar_9_4561767182015543883.png

□ Filter_RMSE_bar_9_4561767182015543883.png

□ Train_Test_lc_RMSE_9_4561767182015543883.png

Data Visualisation

Main **tools** for **visualisation** of manipulated data:

- **Matplotlib** (as *plt*):
  A comprehensive library for creating visualisations in Python:
  https://matplotlib.org/

- **Seaborn** (as *sns*):
  “A Python data visualization library based on matplotlib.
  It provides a high-level interface for drawing […] **statistical** graphics”:
  https://seaborn.pydata.org/

Data Visualisation

1. Compute first statistical data and obtain a **pie-plot**:

- Read-in the **training** Spark DataFrame **df_train**, with manipulated:
  - S-Bahn time-features,
  - Weather-features,
  - Delays at the stations 0, -1, -2 :
    - in minutes
    - as {0, 1} classification

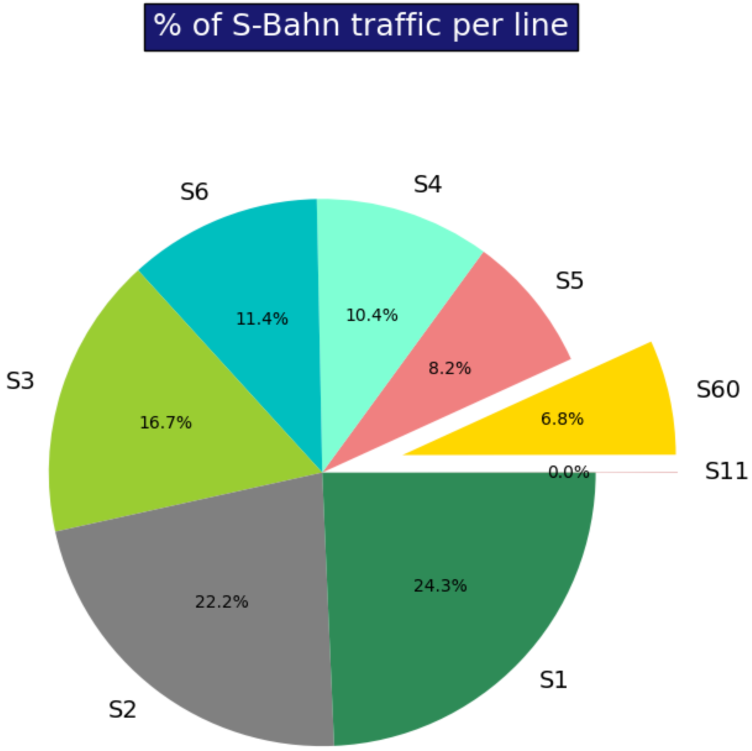
Data Visualisation

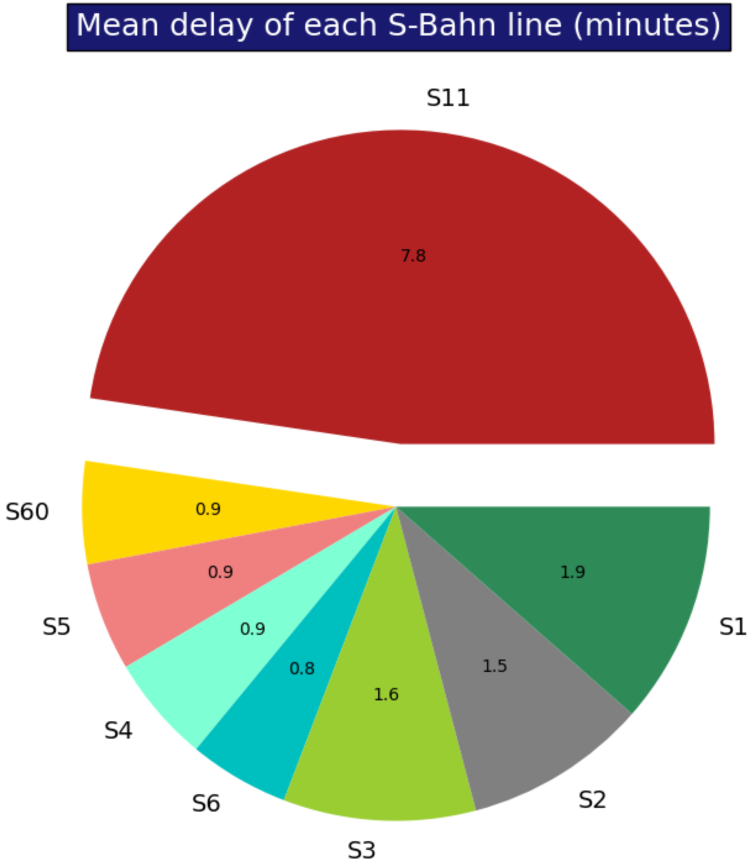
1. Compute first statistical data and obtain a **pie-plot**:

EX 1: Obtain and visualise the Pandas DataFrame **df_train_pd**.

- **df_pd_stats** contains:
  - statistical information on the **number of events** (*count*) and the **delay** (*max, mean, min*)
  - … clustered by S-Bahn line number.
- Plot the ***count*** and ***mean*** information into two pie-plots.

Data Visualisation





Data Visualisation

2. Frequency with a **bar-plot**

- **Goal**: Highlight the **frequency** of three types of delays (in **minutes**):

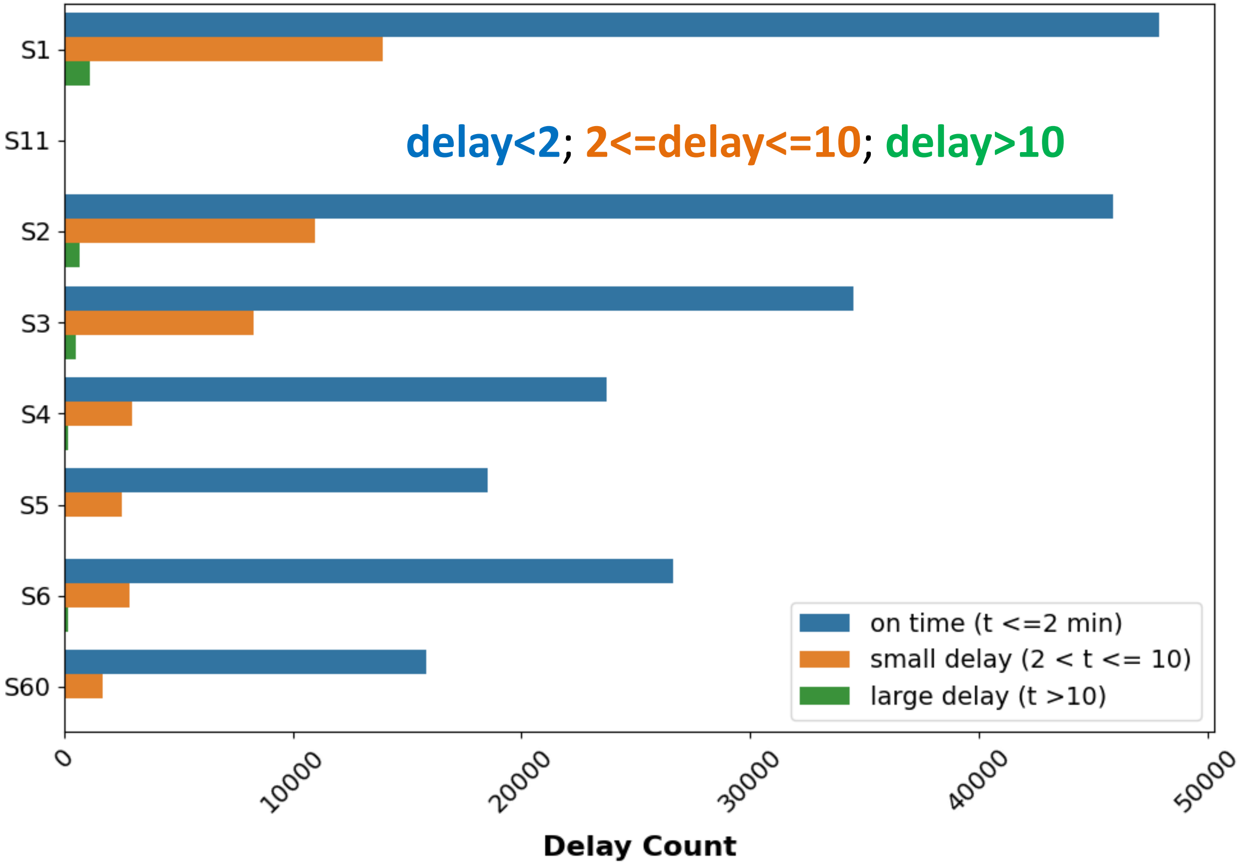
  delay<2; **2<=delay<=10**; **delay>10**

- …using a **Lambda function**:
  a compact anonymous function that does not need an identifier.
- A numbered code is assigned to each category of delay:

  0; **1**; **2**

Data Visualisation

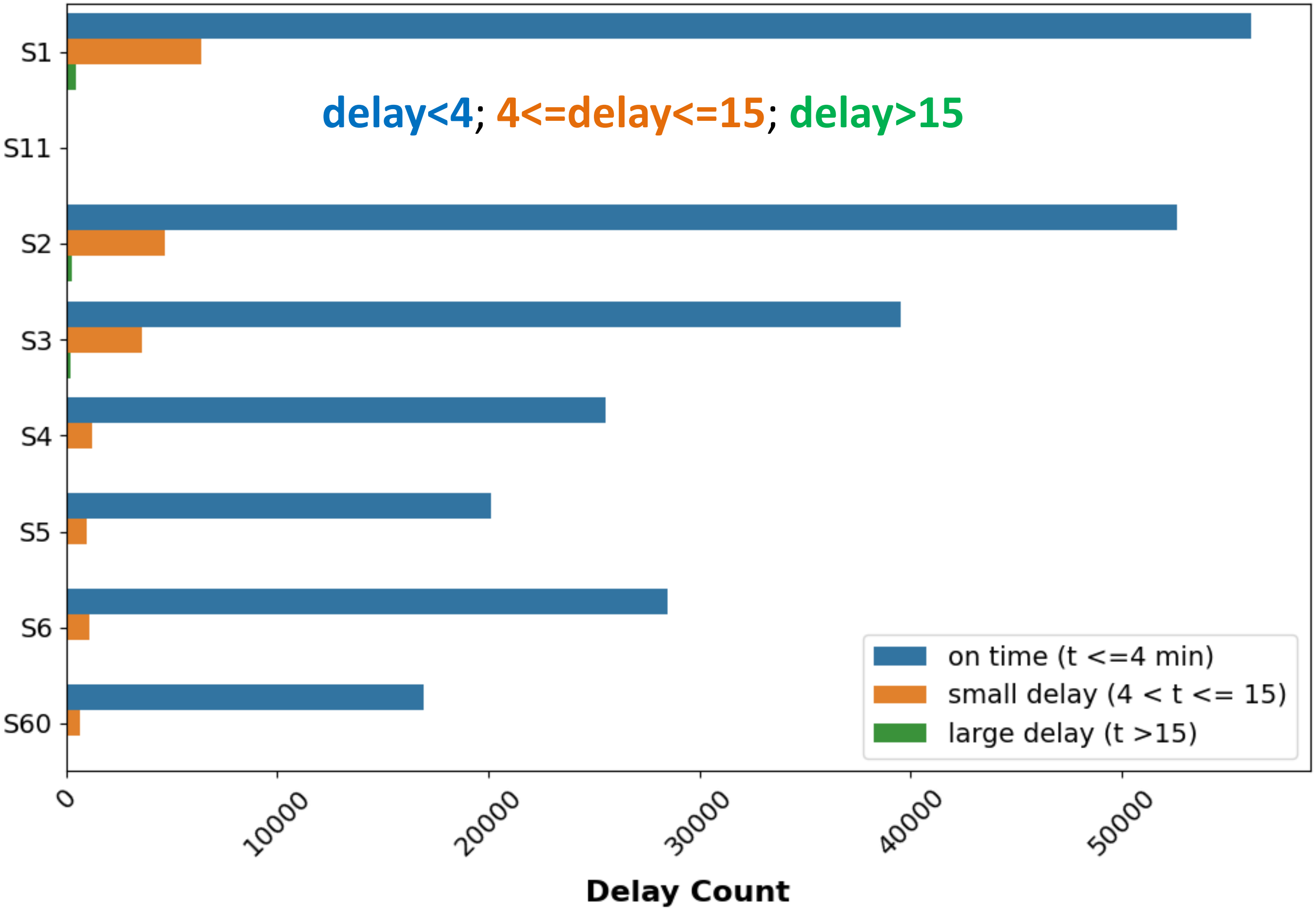
- Apply the Lambda function to the **delay column** of the Pandas DataFrame (to obtain a column of {0, 1, 2}),
- and plot using Seaborn.



delay<2; 2<=delay<=10; delay>10

Data Visualisation

EX 2: Change the threshold delays and generate a new plot.



Data Visualisation

3. Correlation map

+

- Find out **linear relations** between the features and the **target** (delay)…
- …to identify a **subset of features** on which the ML algorithms could run successfully.

–

- **Nonlinear** relations are **not** detected!
- Some potentially relevant features are missing from the catalogue:
  - Number of passengers,
  - Delay of the following train,
  - …

Data Visualisation

- Additional tools for the correlation map:
  - Scikit-learn: ML library with pre-processing tools,
  - … such as the `LabelEncoder` for vectorisation (**EX 3**): Transform categorical features into **numerical features** (see also NB 3, with Spark).

https://scikit-learn.org

Data Visualisation

- We consider **4 sets of feature columns** to observe a possible **correlation to the delay**:
  - Features of the S-Bahn journeys (original dataset)
  - … including derived features from time manipulation
  - … including the delay at the previous stations (-1, -2) → **EX 5**
  - Only weather features

- Accordingly, **4 Pandas DataFrames** are generated.

Data Visualisation

The **Pearson’s correlation coefficient** is used by default:

$$r = \frac{\sum(\boldsymbol{y_i} - \bar{y})(\boldsymbol{x_i} - \bar{x})}{\sqrt{\sum(\boldsymbol{y_i} - \bar{y})^2(\boldsymbol{x_i} - \bar{x})^2}}$$

feat. vectors

- **Where do we see a correlation to the delay?**
  Green or **purple**: Pearson’s coefficient is close to `+1` or `−1` (perfect `direct` and `inverse` correlation).

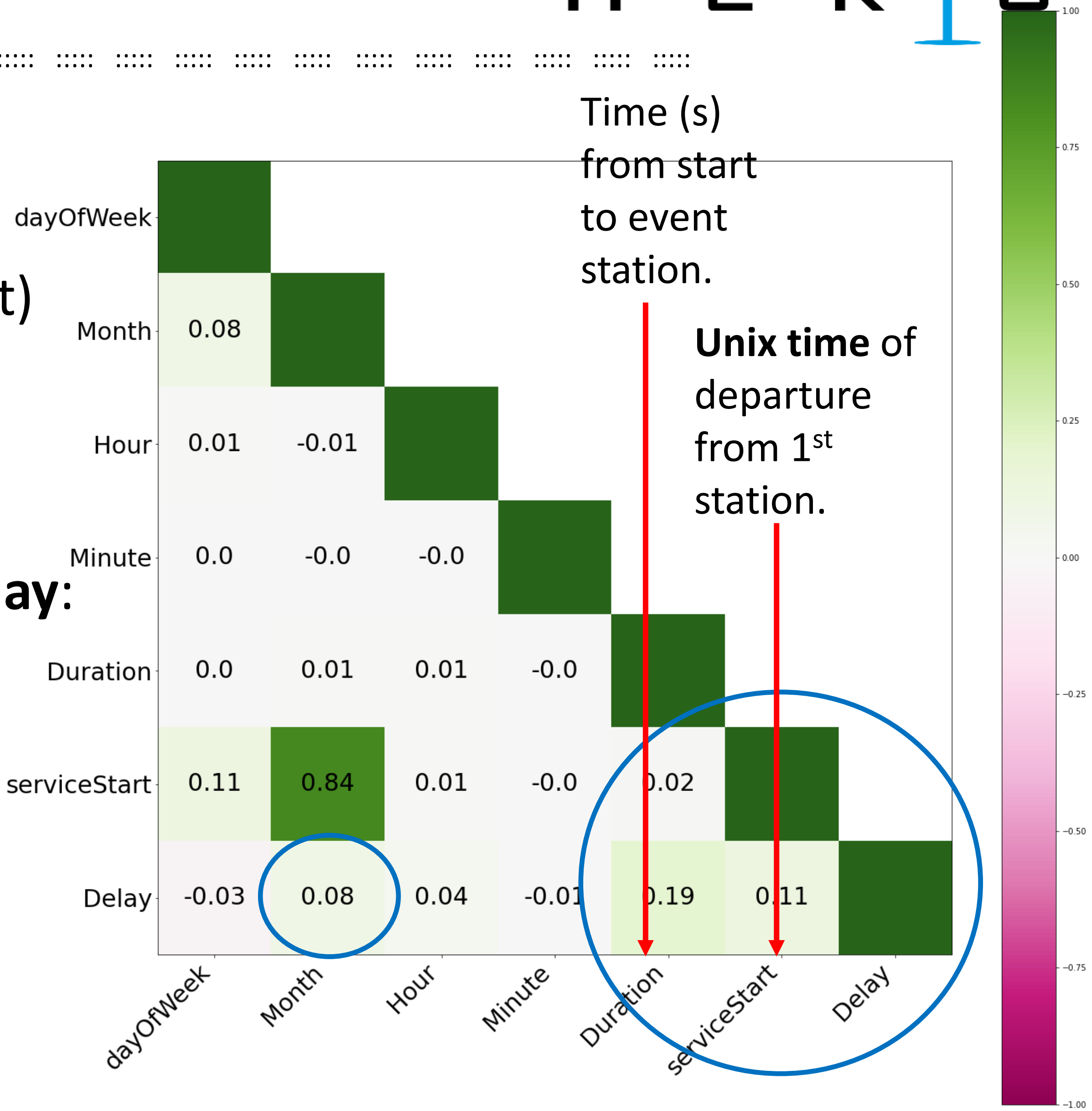
- **EX 4**: How to choose a **different** correlation coefficient?
- **EX 5**: Produce **one** correlation plot, then the plots for **all four** feature combinations.

Data Visualisation

Features of the S-Bahn journeys (original dataset) + features from time manipulation:

(Slight) correlation to **delay**:

- Start of service,
- Duration of travel,
- Month.



	dayOfWeek	Month	Hour	Minute	Duration	serviceStart	Delay
dayOfWeek							
Month	0.08						
Hour	0.01	-0.01					
Minute	0.0	-0.0	-0.0				
Duration	0.0	0.01	0.01	-0.0			
serviceStart	0.11	0.84	0.01	-0.0	0.02		
Delay	-0.03	0.08	0.04	-0.01	0.19	0.11	

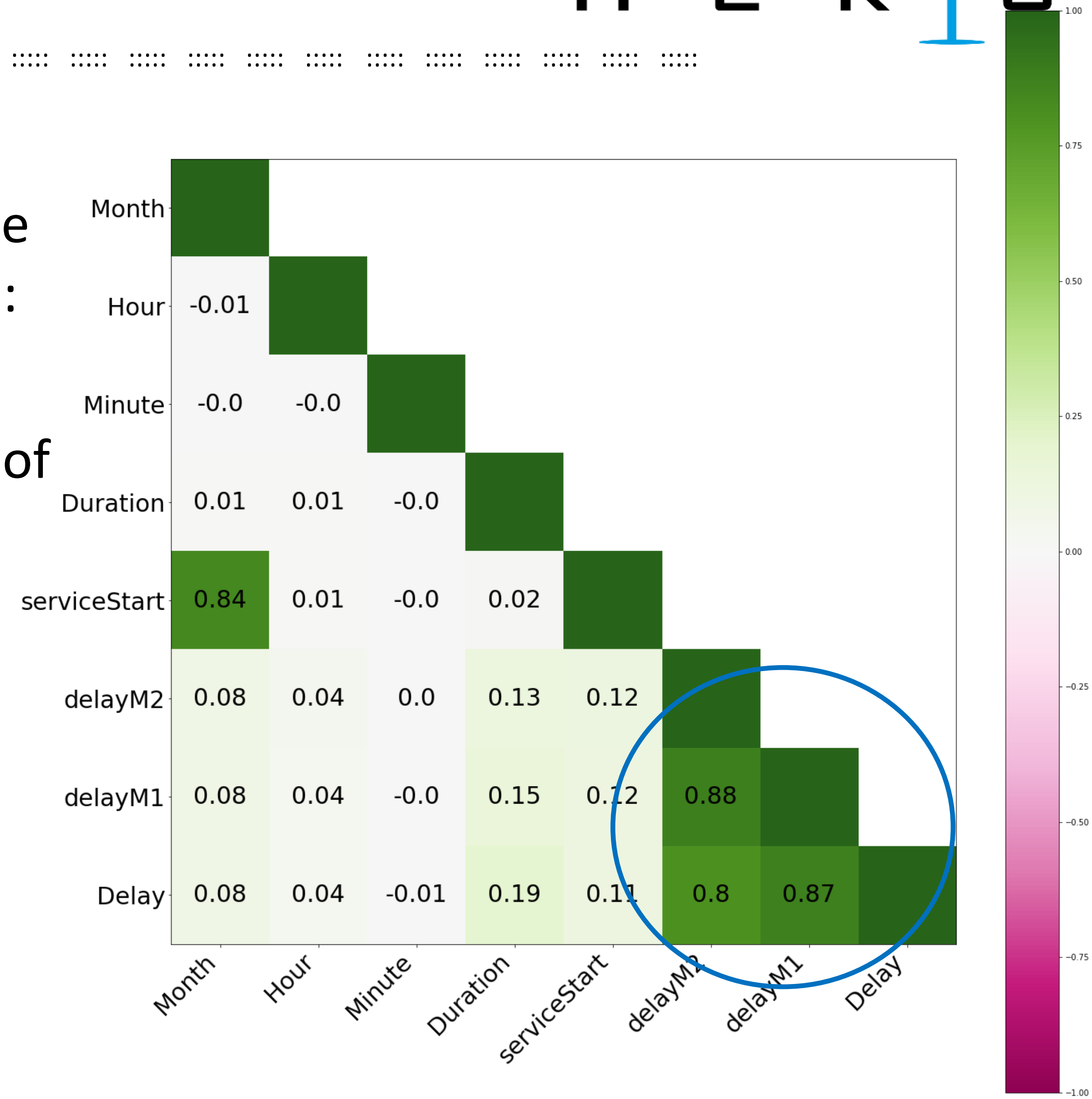
Time (s) from start to event station.

Unix time of departure from 1st station.

Data Visualisation

Including the delay at the previous stations (-1, -2):

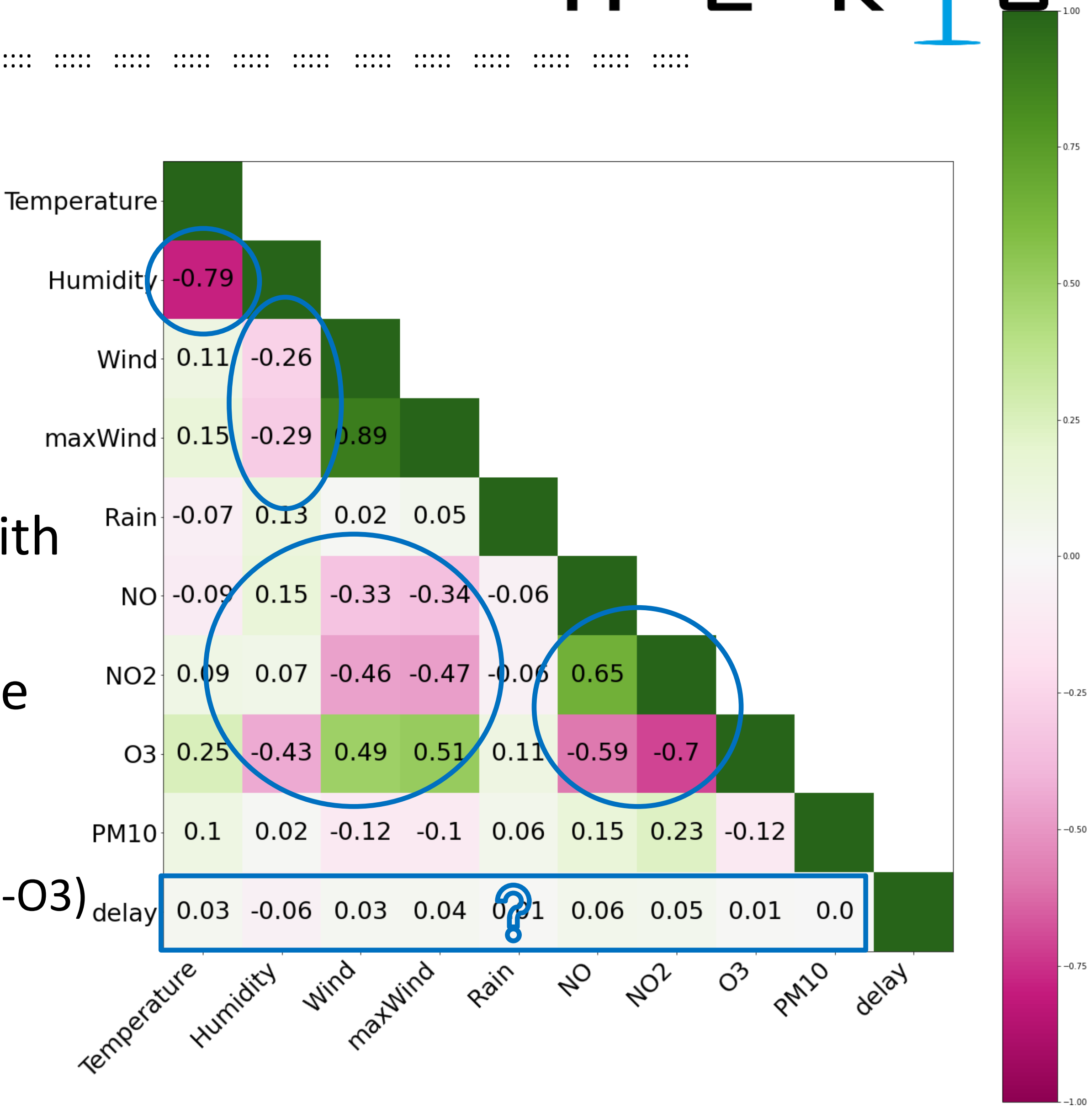
- Very high correlation of **delay** and the delay at previous stations (-1, -2)!



Data Visualisation

Only weather features:

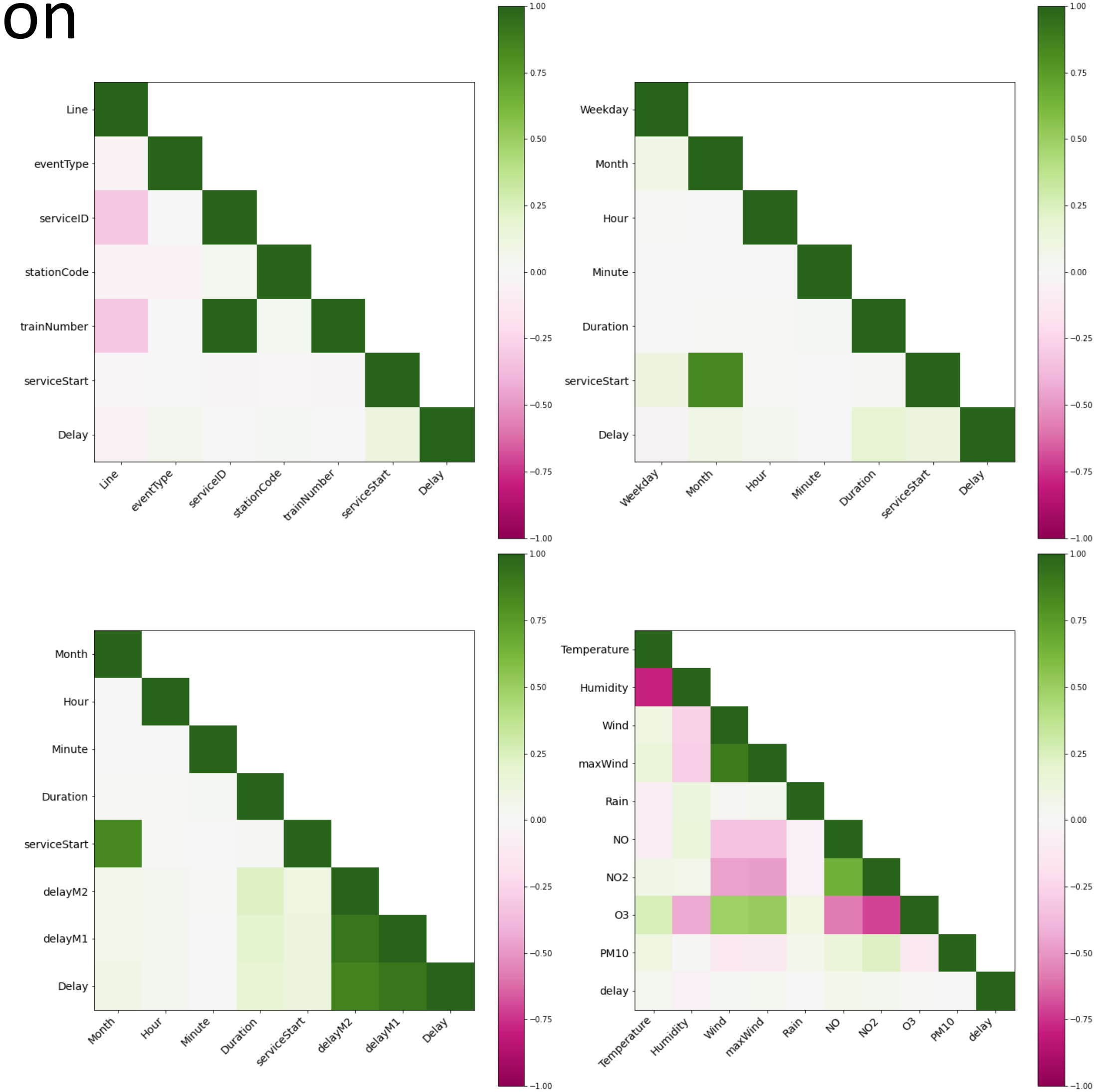
- No clearly visible correlation of **delay** with weather data,
- Some weather data are highly correlated among themselves (wind, humidity, NO1-NO2-O3)



	Temperature	Humidity	Wind	maxWind	Rain	NO	NO2	O3	PM10	delay
Temperature										
Humidity	-0.79									
Wind	0.11	-0.26								
maxWind	0.15	-0.29	0.89							
Rain	-0.07	0.13	0.02	0.05						
NO	-0.09	0.15	-0.33	-0.34	-0.06					
NO2	0.09	0.07	-0.46	-0.47	-0.06	0.65				
O3	0.25	-0.43	0.49	0.51	0.11	-0.59	-0.7			
PM10	0.1	0.02	-0.12	-0.1	0.06	0.15	0.23	-0.12		
delay	0.03	-0.06	0.03	0.04	0.01	0.06	0.05	0.01	0.0	

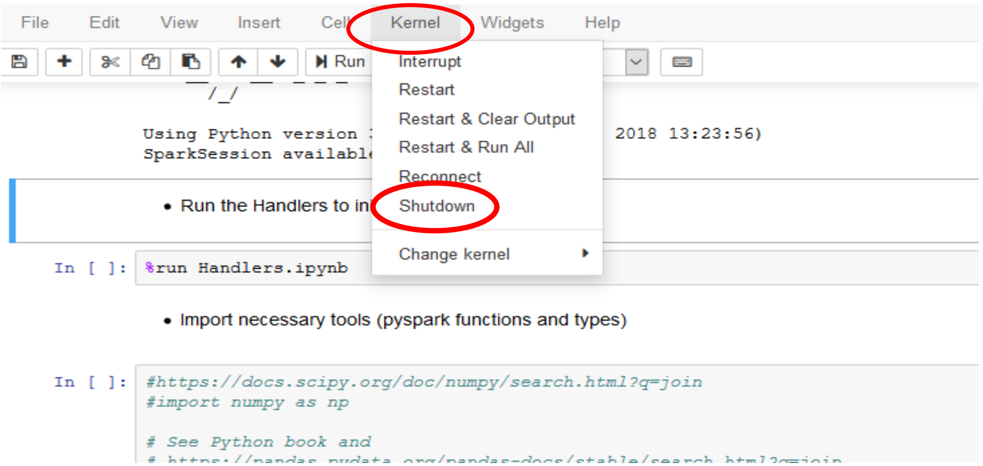
Data Visualisation

All 4 combinations…



Hands-on

- Execute all **NB2_vis-man.ipynb** (Notebook 2)
  - EX 1-3 → “basic”
  - EX 4-5 → “advanced”
- You can **download the plots** obtained.
- Remember to shut down the kernel at the end.



```
Using Python version                2018 13:23:56)
SparkSession available
```

- Run the Handlers to in

```
In [ ]: %run Handlers.ipynb
```

- Import necessary tools (pyspark functions and types)

```
In [ ]: #https://docs.scipy.org/doc/numpy/search.html?q=join
        #import numpy as np

        # See Python book and
        # https://pandas.pydata.org/pandas-docs/stable/search.html?q=join
```

Focus on Pre-processing, Feature Engineering and Machine Learning

- Part II: Example on the Jupyter Notebooks
  - Pre-processing
  - Supervised learning techniques in a Machine Learning pipeline
    - « Notebook 1 (Ex.)
    - « Notebook 2 (Ex.)
    - « Notebook 3 (Ex.)
    - « Notebook 4 (Ex.)
    - « Notebook 5 (Ex.)

More learning outcomes

Main index

Part I
Part II
Part III

Notebook 3: **NB3_linreg.ipynb**

- Jupyter Notebook how-to, see slides: https://fs.hlrs.de/projects/par/events/2024/dl-hlrs/DL-HLRS-day1-exercises.pdf
- A few slides to sum up the content of **this** Notebook follow.

Steps until "Please stop here (introduction)"

Machine Learning LinReg

1. Read-in the data from manipulation

DataFrames after Manipulation:

- **Training**: *df_train* (**50%** of all data)
- **Test**: *df_test* (remaining **50%**)

Machine Learning LinReg

2. Run the **regression** algorithm

- Define the combinations of **feature columns** to run the model:
  - 5 combinations: details later
  - Notice that the feature columns are listed twice in the Notebook:
    - `cols_to_`**`inx`** = feature columns as **input** to the ML pipeline
    - `cols_`**`inx`** = output of **StringIndexer** in the ML pipeline
- The execution of the **ML pipeline** is explained step by step in the next slides.

→ Open **handlers.ipynb** Notebook and go to the function **linReg (Linear Regression Pipeline)**.

You do not need to modify/execute the handlers.

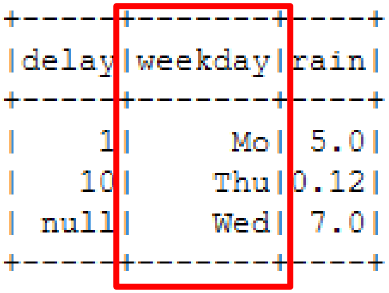
The ML workflow corresponds to the steps in the handlers.

Machine Learning LinReg

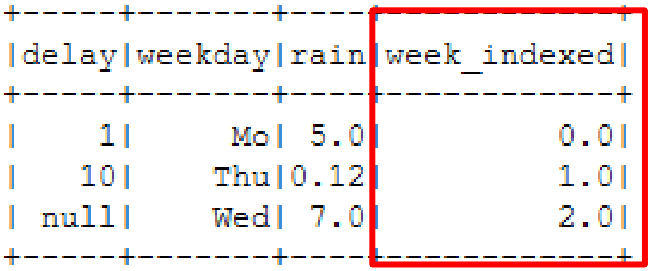
Feature Engineering steps:

- **Vectorisation** with **StringIndexer**:
  Encodes a **string column** of categorical features to a column of (**numerical**) indices {1,2,3, …}

  - Similarly, **one-hot-encoding** is used in DL: **next slide**



```
+-----+-------+----+
|delay|weekday|rain|
+-----+-------+----+
|    1|     Mo| 5.0|
|   10|    Thu|0.12|
| null|    Wed| 7.0|
+-----+-------+----+
```



```
+-----+-------+----+------------+
|delay|weekday|rain|week_indexed|
+-----+-------+----+------------+
|    1|     Mo| 5.0|         0.0|
|   10|    Thu|0.12|         1.0|
| null|    Wed| 7.0|         2.0|
+-----+-------+----+------------+
```

https://spark.apache.org/docs/latest/ml-features

Machine Learning LinReg

One-hot encoding: A **text** is turned into a sparse {0, 1} matrix:

- **Dictionary**

```
small_dict = ['EOS', 'a', 'my', 'jumps', 'on', 'squirrel',
'chicken', 'desk', 'sequoia']
```

- **Input as numerical arrays**

```
X=np.array([[2,6,3,4,2,8,0],[1,5,3,4,7,9,0]],dtype=np.int32)
```

- **One-hot encoded input for** `X[1]` **(the first sentence):**

```
array([[0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0.]])
```

In Spark: Use OneHotEncoder or OneHotEncoderEstimator (version 2.4).

Machine Learning LinReg

One-hot encoding (cont'd):

Example in Natural Language Processing (NLP) :
Predict a **new word in a sequence**.

- In DL, this would be done with a Neural Network with **a one-hot encoded matrix** as input.
- Let us have a **random** prediction instead:
  `x = np.random.rand(9)`
  as an array of real numbers as large as the dictionary.
- One more step is needed to use this result!
- The *last layer* is the *softmax* function to obtain from `x` a **probability vector**:

```
y = np.exp(x)/sum(np.exp(x))
```

Machine Learning LinReg

One-hot encoding (cont'd):

Example in Natural Language Processing (NLP) :
Predict a **new word in a sequence**.

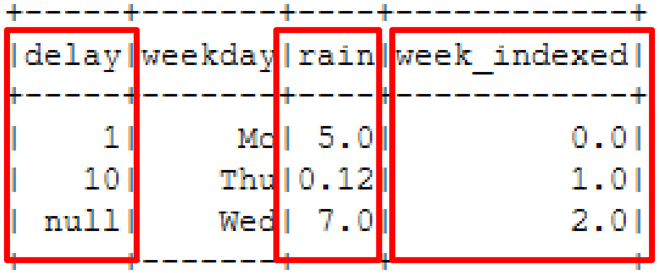
- The entries of the resulting probability vector `y` sum to 1:

```
>>> y
array([0.12889101, 0.07738281, 0.1064542 , 0.11356852,
0.16539288,
       0.11980596, 0.07341816, 0.13642974, 0.07865672])
```

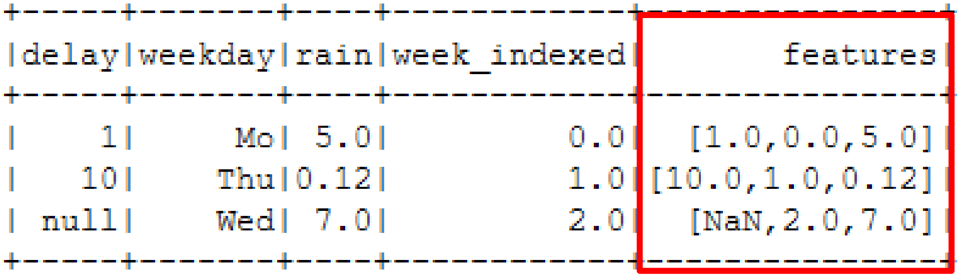
- `np.argmax(y)`
  provides the dictionary position of the predicted word → "on"

Machine Learning LinReg

- **VectorAssembler**:
  - All needed features are condensed into **a single vector** for each sample.
  - The resulting vector is appended to the training DataFrame.



```
+-----+-------+----+------------+
|delay|weekday|rain|week_indexed|
+-----+-------+----+------------+
|    1|     Mo| 5.0|         0.0|
|   10|    Thu|0.12|         1.0|
| null|    Wed| 7.0|         2.0|
+-----+-------+----+------------+
```



```
+-----+-------+----+------------+---------------+
|delay|weekday|rain|week_indexed|       features|
+-----+-------+----+------------+---------------+
|    1|     Mo| 5.0|         0.0|  [1.0,0.0,5.0]|
|   10|    Thu|0.12|         1.0|[10.0,1.0,0.12]|
| null|    Wed| 7.0|         2.0|  [NaN,2.0,7.0]|
+-----+-------+----+------------+---------------+
```

Machine Learning LinReg

- **Normalisation:**
  - **Each feature vector** is normalised to have **unit norm**
  - It takes a parameter $p \in [1, \infty]$: which specifies the L^p-norm used for normalization ($p = 2$ by default, changed to $p = 1$):

$$\boldsymbol{v}_{norm} = \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|_p}$$

$$\|\boldsymbol{v}\|_p = \left(\sum |v_i|^p\right)^{1/p}$$

In computer vision applications:
The pre-processing usually includes **decoding**, **resizing**, and **normalizing** to a standardized format accepted by the neural network [DLI].

Details skipped

Machine Learning LinReg

Why normalisation?

- Normalisation aids generalisation, **standardises the input data** and improves the behaviour of the learning algorithm.
- **DL:** Neural Networks can contain several intermediate normalisation layers (see e.g. Local Response Normalization for CNN). [Krizhevsky et al., ImageNet Classification with Deep Convolutional Neural Networks, 2012]
- **DL:** Batch normalisation in NN
  - Normalise the inputs to all layers in every batch [arXiv:1502.03167]
  - **Q**: Can Spark also normalise across mini-batches or on the whole dataset?

Details skipped

[https://spark.apache.org/docs/1.4.1/ml-features.html#normalizer]

Machine Learning LinReg

Define the **estimator (= the architecture of the model):**

https://spark.apache.org/docs/latest/ml-classification-regression

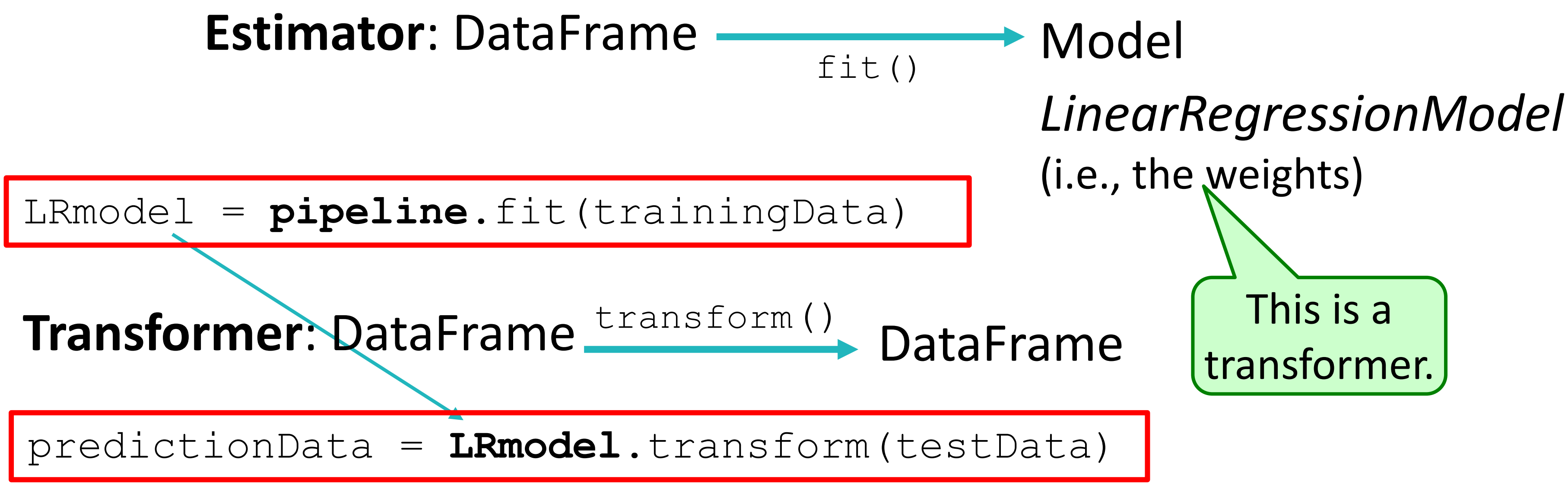
- Define the model **class** as:
  - *LinearRegression*
  - with the model **hyperparameters** (here: *maxIter*, *regParam*),
  - specifying the **features** and the **label / target**.

- All steps so far (feature engineering + model definition) are collected into a **pipeline (next slide)**.

https://spark.apache.org/docs/latest/mllib-linear-methods.html

Machine Learning LinReg

Estimator vs. Transformer



Estimator: DataFrame $\xrightarrow{\texttt{fit()}}$ Model

LinearRegressionModel

(i.e., the weights)

```
LRmodel = pipeline.fit(trainingData)
```

Transformer: DataFrame $\xrightarrow{\texttt{transform()}}$ DataFrame

This is a transformer.

```
predictionData = LRmodel.transform(testData)
```

The training `Pipeline` is an Estimator → ML model

The `PipelineModel` is a Transformer → inference dataframe

→ Same as the pre-processing of data (sequence of transformers).

https://spark.apache.org/docs/latest/ml-pipeline.html

Machine Learning LinReg

Define the **evaluator:**

- The fitted ML model can now make predictions.
- A suitable **metric** is defined to **evaluate the model performance** (and assess the quality) **after training**:
  - Mean Absolute Error (MAE)
  - Mean Squared Error (MSE)
  - Root Mean Squared Error (**RMSE**)
  - Polynomial approaches …

Large deviations are emphasised through **S**, then the original scale restored through **R**

https://spark.apache.org/docs/latest/mllib-evaluation-metrics.html#regression-model-evaluation

Machine Learning LinReg

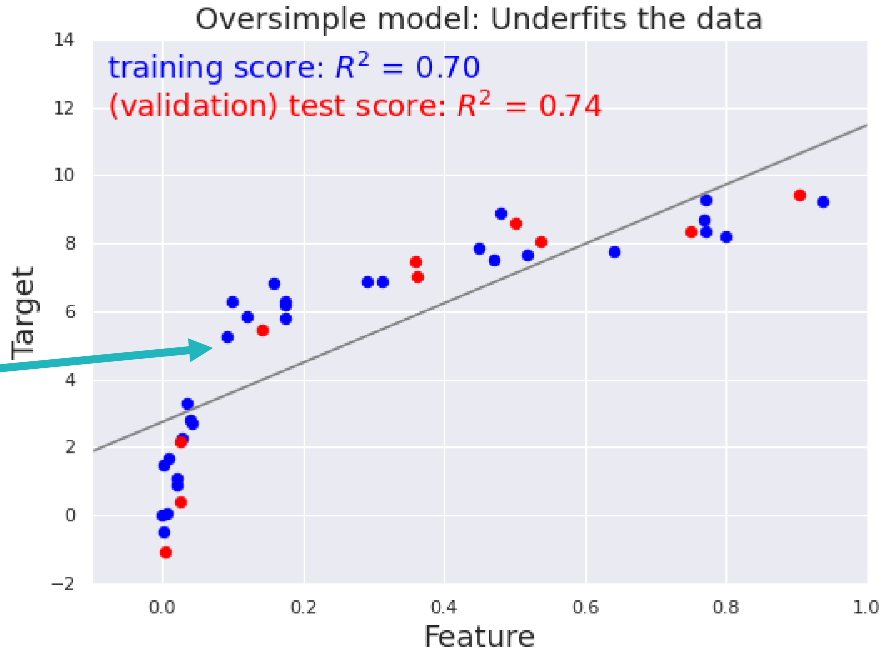
Details of the ML algorithm:

- Basic idea of **supervised learning** is to learn a function:

$$y = f(x; w)$$

where **w** are the parameters of the model (weights).

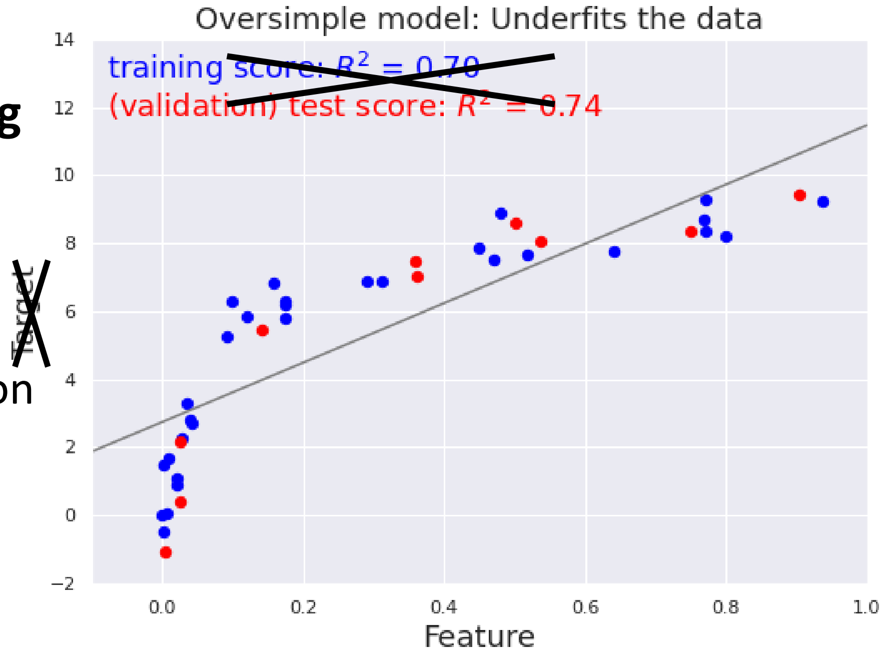
- The **training** samples $\{x_i, y_i\}$ (blue points) are used to build the model.



[code adapted from: PHB Notebooks]

Machine Learning LinReg

- The idea of linear regression is similar to (linear) **dimension reduction** in **unsupervised learning**
- Embedding in a *latent space*:
  - Maintain topological properties (e.g., distance)
  - Minimise the loss at reconstruction
- Here: 2D →1D, both dimensions are features



[code adapted from: PHB Notebooks]

Machine Learning LinReg

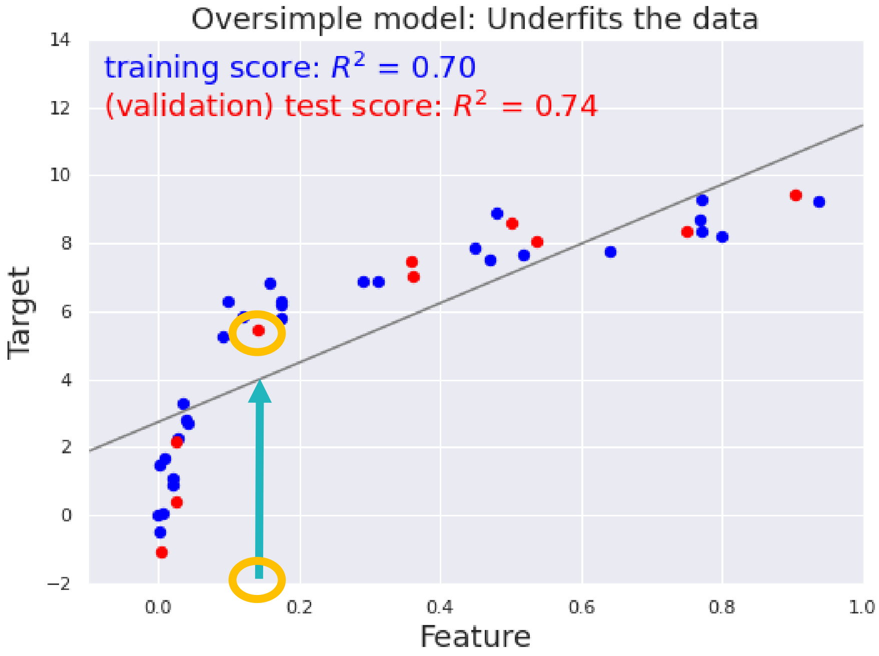
Linear regression:

- Basic idea: Fit to a line in 2D, to a plane in 3D, or to a hyperplane in *n*-D (**affine subspaces**).

  - 1D feature and 1D target
  → 1D polynomial regression (line)

  - Score $R^2 \in [1, -\infty)$: next slide

  - This model is not very accurate!
  We will see the opposite case.



[code adapted from: PHB Notebooks]

Machine Learning LinReg

true

$$R^2 = 1 - \frac{\sum_{i=0}^{N-1}(y_i - \hat{y}_i)^2}{\sum_{i=0}^{N-1}(\hat{y}_i - \langle\hat{y}\rangle)^2}$$

prediction

$$R^2 = 1$$

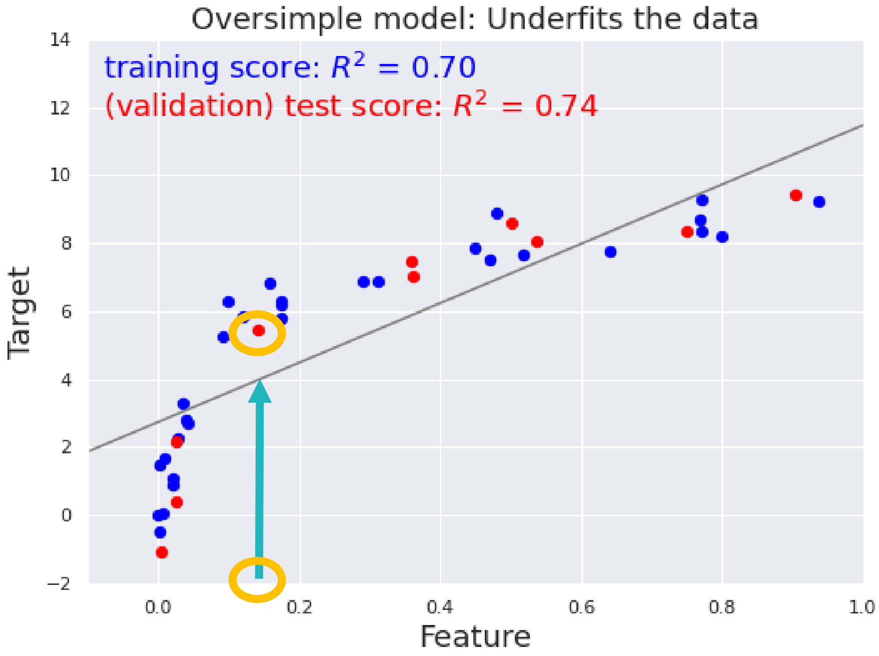
- Perfect match observed value – prediction.

$$R^2 = 0$$

- Prediction matches the mean of all true values.

$$R^2 \in (0, -\infty)$$

- Prediction arbitrarily bad.



https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html?highlight=score#sklearn.linear_model.LinearRegression.score

[code adapted from: PHB Notebooks]

Machine Learning LinReg

Linear regression:

- In our case:

Feature combinations with each:

5, 11, **1** (**line**), **2** (**plane**), 8 features

↓ **to predict**

1D **label**

(range of the regression function: delay in minutes)

Q: Does LinReg always predict a **linear relation (line, plane, …)** between the label and the features?

Machine Learning LinReg

The **input vector** can be a **nonlinear function** of the **feature vector**:

- E.g., the polynomial mapping of a feature vector $\boldsymbol{x}_i$ (1D case):

$$y_i = w_0 + w_1 x_i + w_2 x_i^2 + \cdots$$

- Transformation of **multi-dimensional** features, e.g. $\left(x_{ij}\right)^j$
- **Interaction** among multi-dimensional features, e.g. cross-products $(x_{i1} \cdot x_{i5})$
(e.g. the day of week X humidity).

The index i refers to the observation!

https://webis.de/downloads/lecturenotes/machine-learning/unit-en-linear-regression.pdf

Machine Learning LinReg

In the prediction…

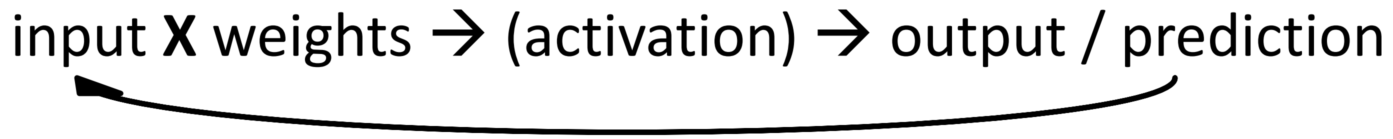
$$y_i = \sigma(\boldsymbol{w}^T \hat{\boldsymbol{x}}_i)$$

… **linearity** must be preserved in the **coefficients** or **weights** $(w_0, w_1, w_2, \ldots)$, not in the feature transformations $\hat{\boldsymbol{x}}_i$:

For **one** observation i:

- y_i: real-valued prediction of this **linear** function,
- $\hat{\boldsymbol{x}}_i$: is the **input** vector of the transformations of the feature vector $\boldsymbol{x}_i$ for each sample i.

→ Corresponds to **one** (dense) **layer** in a **Neural Network:**



input **X** weights → (activation) → output / prediction

https://webis.de/downloads/lecturenotes/machine-learning/unit-en-linear-regression.pdf

Machine Learning LinReg

In practice, linear regression **in Spark**:

- The predicted output
  - follows a **Gaussian distribution**,
  - is a **realisation** of the normal function.

For each **observation** i, the prediction has the normal distribution:

expected value

$$f(y_i) = {}^{1}\!/_{\sqrt{2\pi\sigma^2}} \exp\left(- \frac{\frac{1}{2}(y_i - \boxed{\mu_i})^2}{\sigma^2} \right)$$

The Gaussian model assumes **mutually independent observations**:

→ This is in our case a **significant simplification**, since the train journeys do influence each other!

Machine Learning LinReg

- The relation between the **expected value** μ_i and the features depends on the chosen distribution.
- This relation can be expressed through the **link function** $g(\cdot)$ between the expected value μ_i and the linear predictor η_i :

$$g(\mu_i) = \eta_i \coloneqq \boldsymbol{x}_i^T \boldsymbol{w}$$

- The canonical link function for the **Gauss distribution** is the identity:

$$\mu_i = w_1 x_{i1} + w_2 x_{i2} + \cdots$$

→ which yields **a fully linear model.**

Bias-term omitted

Rodríguez, G. (2007). Lecture Notes on Generalized Linear Models.
URL: https://grodri.github.io/glms/notes/
https://data.princeton.edu/wws509/notes/c2.pdf

Machine Learning LinReg

Variations:

- Simple (1D feature) vs. multiple (*n*-D features) Linear Regression
- Generalised Linear Regression:
  The response variable follows a **different distribution**
  (e.g. Binomial, Poisson, Gamma, Tweedie functions).

https://spark.apache.org/docs/latest/ml-classification-regression.html#generalized-linear-regression ,
https://en.wikipedia.org/wiki/Generalized_linear_model

Machine Learning LinReg

Once the model is defined, **we look for the best combination of weights.**

Define the **validation**:

- Define the **hyperparameter grid** to:
  - **add** or **overwrite** already defined hyperparameters
  - fit a **new model** for each “point of the grid”
  - here: `2x2` **hyperparameters** (`regParam`, `elasticNetParam`)

Search in the grid is done through **cross validation** → next slides.

Careful :

» parameters = weights (esp. in DL context).

» **hyper**parameters = knobs to tweak for tuning the model.

Overview of all the regularisation hyperparameters in Spark: https://spark.apache.org/docs/latest/ml-classification-regression.html#linear-methods

Machine Learning LinReg

ML as minimisation problem:

- The LinReg algorithm is the **minimisation** of a (convex) function (objective) of the weight vector $\boldsymbol{w}$:

$$\min_{\boldsymbol{w}} f(\boldsymbol{w})$$

This minimum balances:

- The **loss function (next slide)** → Guarantees that the model is correct, i.e. each **training sample** is closely mapped to the correct label (**effectiveness**)

$$f(\boldsymbol{w}) := \frac{1}{n}\sum_{i=1}^{N} \boxed{L(\boldsymbol{w}; \boldsymbol{x}_i, y_i)} + \lambda \boxed{R(\boldsymbol{w})}$$

- … and the **regularisation function**, i.e. a penalisation term to minimise the model **complexity** → Avoid **overfitting (next slides)**.

https://webis.de/downloads/lecturenotes/machine-learning/unit-en-regularization.pdf

Machine Learning LinReg

For a **linear model**, the **loss function** is the squared **residual**, computed for every sample in the dataset:

$$L_i = \frac{1}{2}(\boldsymbol{w}^T\hat{\boldsymbol{x}}_i - \hat{y}_i)^2$$

y =weights X features
Prediction of the model
(**estimated value**)

known label for
training and test data
(**observed value**)

The **regression error** (or misclassification) for validation is the Root Mean Squared Error:

$$\textbf{RMSE} = \sqrt{\sum_{i=0}^{N-1}\frac{(y_i-\hat{y}_i)^2}{N}}$$

Total
number of
samples

.

https://webis.de/downloads/lecturenotes/machine-learning/unit-en-linear-regression.pdf

Machine Learning LinReg

We will actually minimise f

Minimising the loss function:

The minimisation of squared residuals is a **least squares** problem.

It can be solved through:

- Direct methods (computing the pseudo-inverse):
  - Normal equation (numerically unstable, not for big data!)
  - Methods based on the QR decomposition and the singular value decomposition (SVD)
- **Iterative methods: Gradient Descent → next slides**.

https://webis.de/downloads/lecturenotes/machine-learning/unit-en-regularization.pdf

Machine Learning LinReg

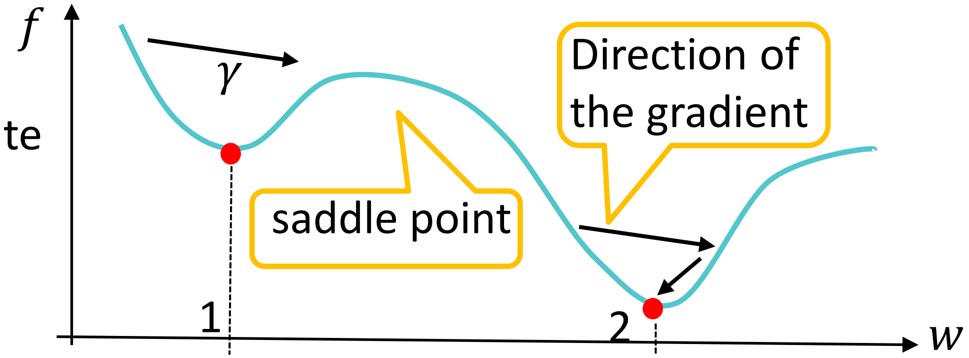
Gradient Descent:

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \gamma f'_{\boldsymbol{w}^{(t)}}$$

- t denotes the **iteration number**
- $\boldsymbol{w}$ are the weights before and after the update
- f'_w is the **gradient** of the loss function (in DL: result of **backpropagation**)
- γ is the **step size** (**learning rate** in DL literature):
  In Spark, it has both a fixed (s) and a variable component: $\gamma = {}^{s}/_{\sqrt{t}}$
  → While t increases, the step size is reduced.

next slides

Adjustable learning rate to escape local minima!



Adjustable learning rate to catch the global minimum!

https://spark.apache.org/docs/latest/mllib-optimization.html

Machine Learning LinReg

Backpropagation:

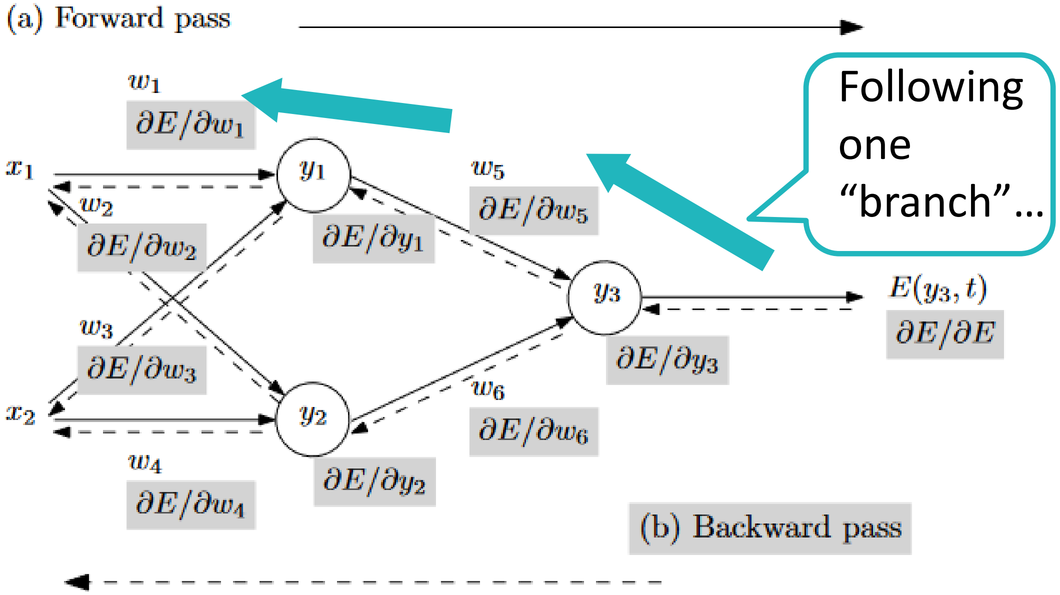
- Training input x_i are fed forward, generating corresponding activations y_i.
- E is the error between the final output (y_3) and the target ($\widehat{y_3}$, in the paper: t), same as the loss function.
- Through the chain rule:

$$\frac{\partial E}{\partial w_5} = \underline{\frac{\partial E}{\partial y_3}\frac{\partial y_3}{\partial w_5}}$$

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial w_1} =$$

$$= \underline{\left(\frac{\partial E}{\partial y_3}\frac{\partial y_3}{\partial y_1}\right)}\frac{\partial y_1}{\partial w_1}$$

…


(a) Forward pass
w_1
$\partial E/\partial w_1$
x_1
y_1
w_5
$\partial E/\partial w_5$
w_2
$\partial E/\partial w_2$
$\partial E/\partial y_1$
y_3
$E(y_3, t)$
$\partial E/\partial E$
w_3
$\partial E/\partial w_3$
$\partial E/\partial y_3$
x_2
y_2
w_6
$\partial E/\partial w_6$
w_4
$\partial E/\partial w_4$
$\partial E/\partial y_2$
(b) Backward pass

Following one “branch”…

Baydin et al., Automatic Differentiation in Machine Learning: a Survey, 2018
Mathieu et al., Fast Training of Convolutional Networks through FFTs, 2014

Machine Learning LinReg

Backpropagation:

- … we get the final gradient with respect to all weights:

$$\nabla_{\boldsymbol{w}} E = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \dots, \frac{\partial E}{\partial w_6} \right)$$

- The gradient can be subsequently used in a **Gradient Descent** procedure.

- In the context of **Physics Informed Neural Networks** (PINN):
  The gradient with respect to the inputs $\nabla_{\boldsymbol{x}} E$ can be also computed in the same backward pass.

Details skipped

Baydin et al., Automatic Differentiation in Machine Learning: a Survey, 2018
Mathieu et al., Fast Training of Convolutional Networks through FFTs, 2014

Machine Learning LinReg

Very important for optimisation, also on (multi-)GPU architectures:
How often are the weights updated **for each training dataset**?

- **Stochastic** Gradient Descent (SGD):
  Update at **each sample in the training** set: The gradient of the loss function is computed $\nabla_{\boldsymbol{w}} L(\boldsymbol{w}; \boldsymbol{x}_i, y_i)$ and the weights are updated $\boldsymbol{N}$ **times**.

- Gradient Descent:
  The gradient is computed only **once for the averaged loss function**:

$$\nabla_{\boldsymbol{w}} \left(\frac{1}{N} \sum_{i=1}^{N} L(\boldsymbol{w}; \boldsymbol{x}_i, y_i) \right)$$

https://spark.apache.org/docs/latest/mllib-optimization.html

Machine Learning LinReg

- **Mini-batch** : One update for every batch S (intermediate scenario):
  - $|S| = miniBatchFraction \cdot N$

$$\nabla_{\boldsymbol{w}} \left(\frac{1}{|S|} \sum_{i \in S} L(\boldsymbol{w}; \boldsymbol{x}_i, y_i) \right)$$

    « The default *miniBatchFraction* is **1**:
    One update for the complete training set (GD method).

    « A **small** *miniBatchFraction* $1/N$ corresponds to the SGD method:
    Update for each sample.

  - A **stochastic** component is introduced through the choice of mini-batches S.

[http://ufldl.stanford.edu/tutorial/supervised/OptimizationStochasticGradientDescent/]

Machine Learning LinReg

Training with GD:

- One **iteration** or **step**
  = One update of the weights.
- One **epoch** (in DL-literature)
  = One prediction (*forward* pass) and one gradient compute (*backward* pass) of **all** training samples.
  Small batches => Many steps per epoch.
  Large batches => Risk of overfitting [arXiv:1609.04836]
  Linear scaling of batch size & learning rate [arXiv:1404.5997]

More advanced methods of (stochastic) optimisation: AdaGrad, RMSProp, **Adam**
[Kingma and Ba, Adam: A Method for stochastic optimization, 2015]

What should we actually minimize?

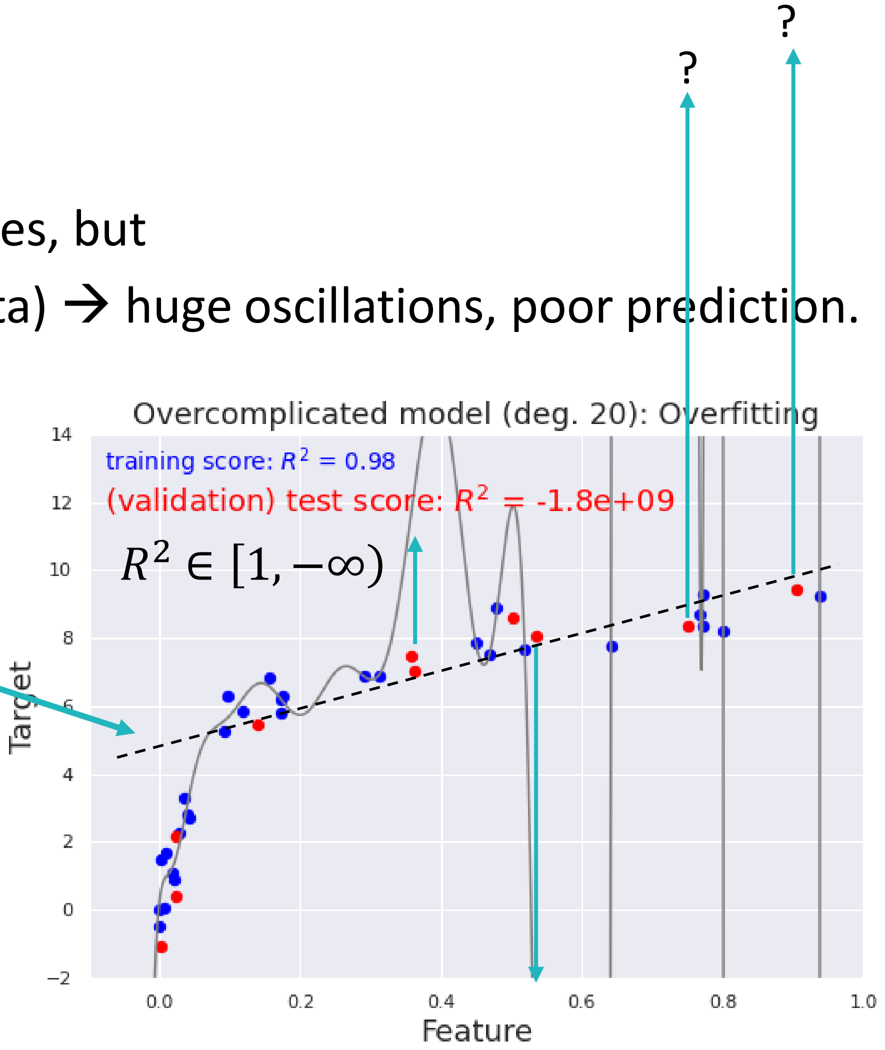
→ The minimisation of the **loss function** could lead to **overfitting** .

Machine Learning LinReg

Overfitting: The model is

- **overly precise** for the training samples, but
- **poorly precise** for new data (test data) → huge oscillations, poor prediction.

- 1D feature and 1D target
→ **20D** polynomial regression vs. 1D regression (line).



[code adapted from: PHB Notebooks / link to R^2]

Machine Learning LinReg

Reasons for **Overfitting**

- Overly complex models (“exploding” absolute values of the weights),
- … but also the **training data**:
  - A lot of noisy / incorrect data,
  - Dominant biased data, non representative outliers *(e.g., the event line S11, traffic during COVID lock-down or Christmas)*,
  - Too small set with data properties that do not reflect the general behaviour *(e.g. too few journeys)*.

https://webis.de/downloads/lecturenotes/machine-learning/unit-en-regularization.pdf

Machine Learning LinReg

How to prevent overfitting:

- Increase the quality of the training data for more robust models: *Artificially* increase the size of the dataset, e.g.:
  - Extract random patches, apply translations, rotations…
  - Alter the intensity of RGB channels

  [Krizhevsky et al., ImageNet Classification with Deep Convolutional Neural Networks, 2012]

computer vision in DL

- Use **regularisation**

→ Build a penalised, **sub-optimal** model: The loss function will be only "partially minimised".

→ In practice, the loss function is augmented .

$$f(\mathbf{w}) := \frac{1}{N}\sum_{i=1}^{N} L(\mathbf{w}; \mathbf{x}_i, y_i) + \boxed{\lambda R(\mathbf{w})}$$

Machine Learning LinReg

Regularisation in practice: $\lambda R(\boldsymbol{w})$ is made of:

- $\boxed{\lambda}$: Fixed regularisation parameter to tune the impact of $R(\boldsymbol{w})$, **times**
- $R\ (\boldsymbol{w})$: **Elastic Net** function:

$$R(\boldsymbol{w}) = \alpha \parallel \boldsymbol{w} \parallel_1 + (1-\alpha)\frac{1}{2} \parallel \boldsymbol{w} \parallel_2^2$$

which combines:

- LASSO regression: L^1 regularisation for sparsity in the weights (simpler models with some $w_j = 0$ → some features are neglected)
- Ridge regression: L^2 regularisation for a **smoother** function

The two approaches influence the **trajectory** towards the minimum of $f\,(\boldsymbol{w})$.

The **coefficient** $\boxed{\alpha}$ tunes the impact of either L^1 or L^2 regularisation.

https://webis.de/downloads/lecturenotes/machine-learning/unit-en-regularization.pdf

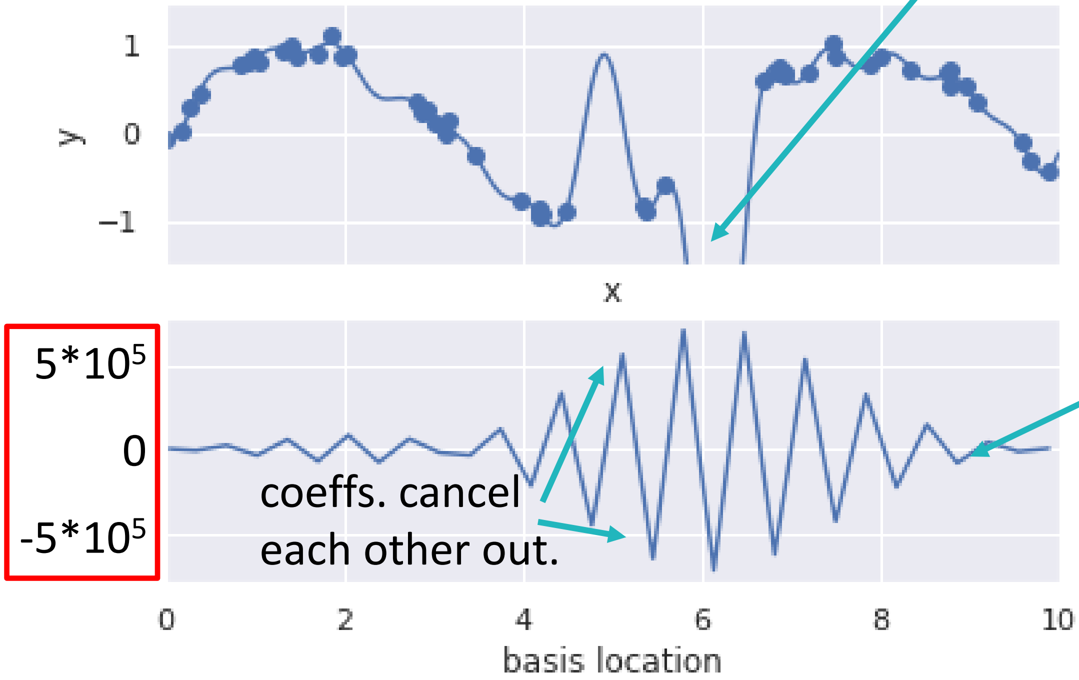
Machine Learning LinReg

Basis functions / polynomial approach different than in Spark.

Regularisation in practice:

1D-feature prediction through **30** basis functions (Gaussian) without regularisation:

Overfitting!



coeffs. cancel each other out.

Coefficients w_j at each Gaussian basis centre.

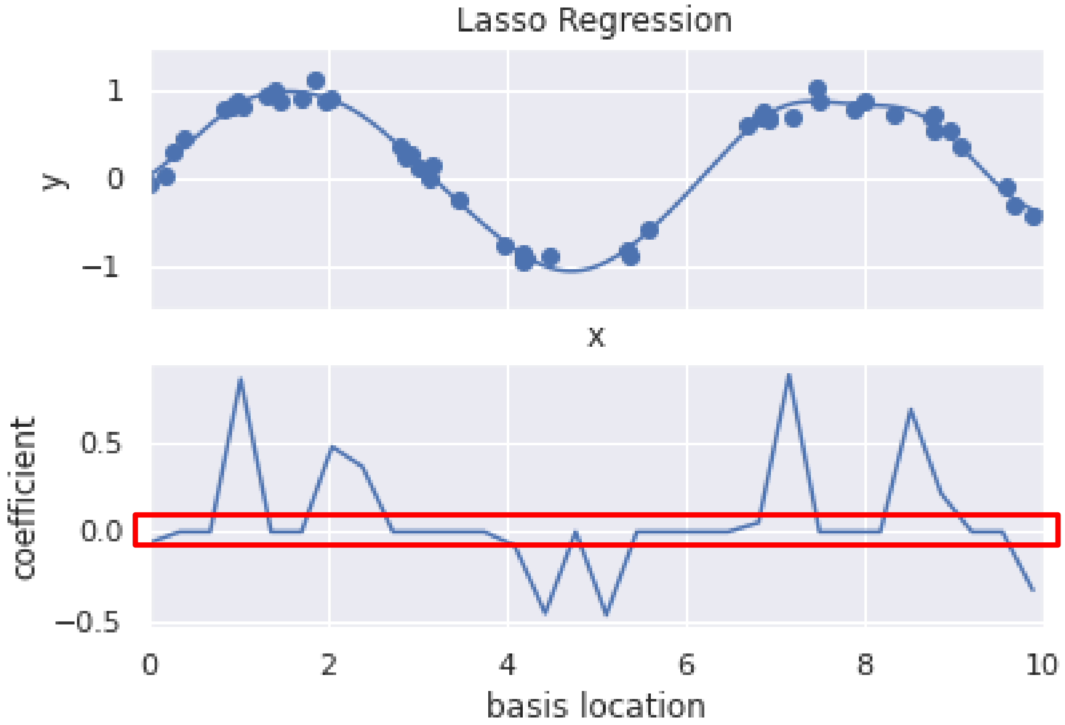
“Exploding” coefficients w_i

[code adapted from: PHB Notebooks]

Machine Learning LinReg

Regularisation in practice:

30 basis functions (Gaussian) with Lasso (L^1) regularisation:



Regression on **12 bases**

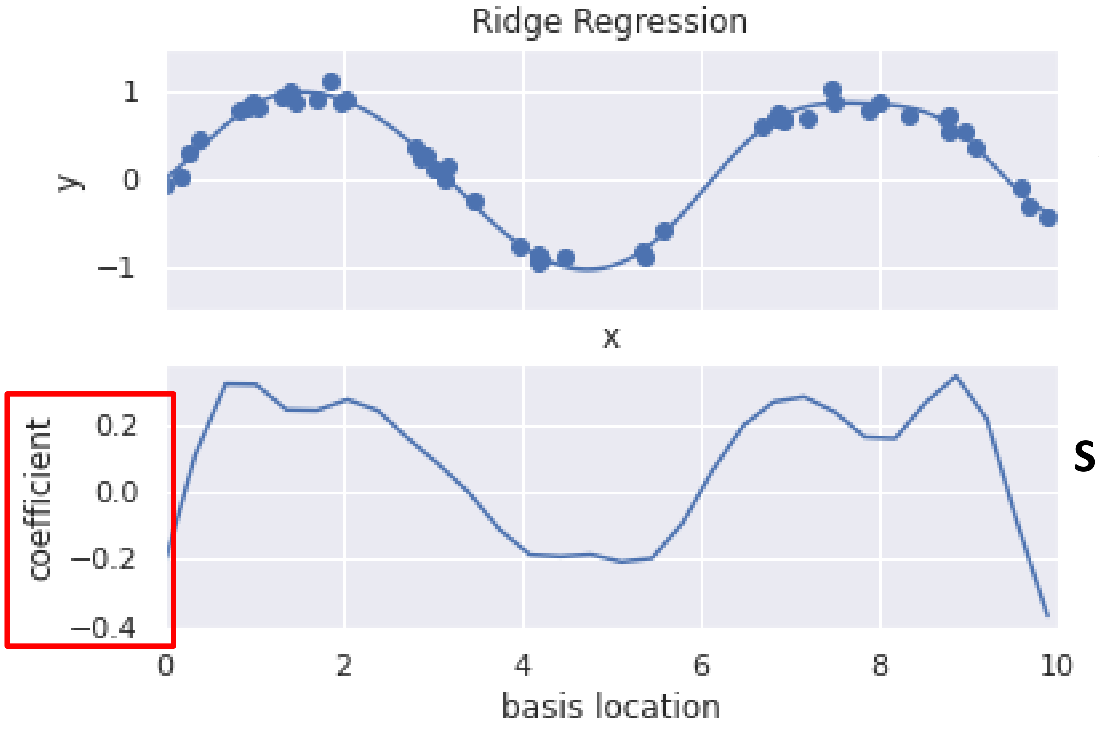
Sparse model with 18x $w_j = 0$

[code adapted from: PHB Notebooks]

Machine Learning LinReg

Regularisation in practice:

30 basis functions (Gaussian) with Ridge (L^2) regularisation:



Very similar result to L^1.

Smooth model with $w_j \neq 0 \; \forall j$

[code adapted from: PHB Notebooks]

Machine Learning LinReg

Regularisation techniques in **Deep Learning**:

- **Normalisation layers**: *already discussed*.
- **Dropout**:
  "Switch off" neurons (i.e. weights) at a random rate. This way, there will be no co-adaptations of neurons (or ***lazy* weights**).
  Compensates for overfitting!
  The final model for prediction will use **all** weights.
- **Pooling**:
  Reduces the size of images at intermediate layers in different ways, helping generalisation.
  Compensates for shifts.

Details skipped

[Krizhevsky et al., ImageNet Classification with Deep Convolutional Neural Networks, 2012]

Machine Learning LinReg

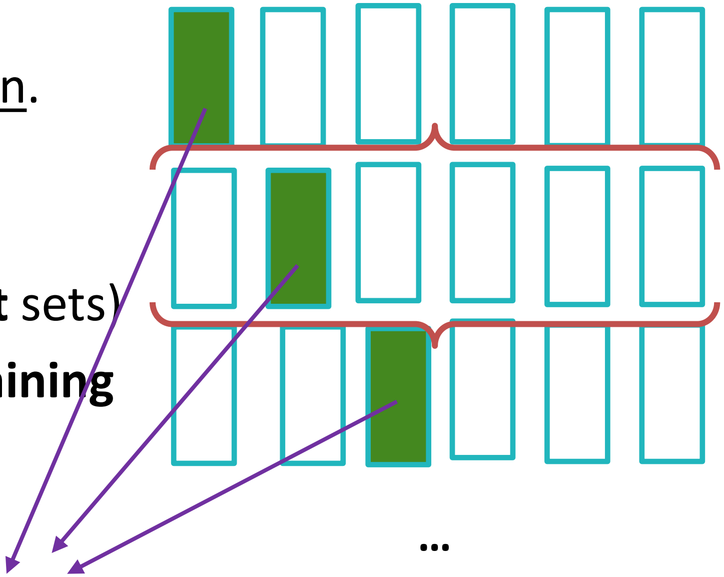
How the parameter grid is used in practice.

Define the **cross validation:**

Idea: Improve training with continuous validation.

For each hyperparameter combination:

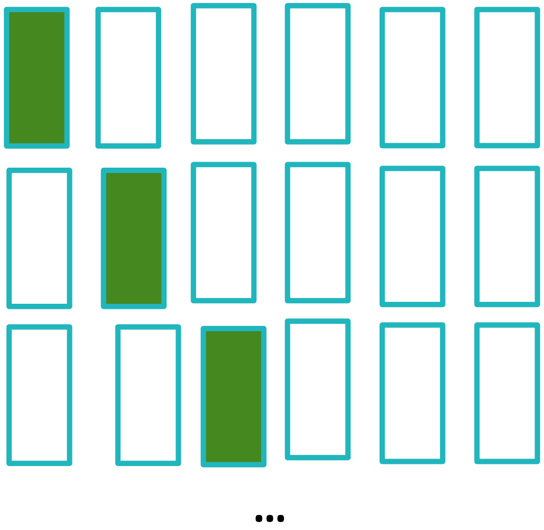
- The original **training dataset** is partitioned into ***k*** non-overlapping subsets (**development** sets)
- One subset (*hold-out* set) is left **out of the training** and used for **validation**
- The **training is repeated *k*** times
- The **evaluation metric** is computed on **all hold-out sets** and **averaged**.



In the code, the **CrossValidator** object contains the whole pipeline (estimator, parameter grid, evaluator) and calls the **fit** over the training data.

Machine Learning LinReg

Cross validation:



In our case:

→ **(2x2) x 10 = 40 models** are being trained and evaluated

folding of the training dataset

grid search over hyperparamers

→ The best combination of hyperparameters is chosen (lowest averaged error)

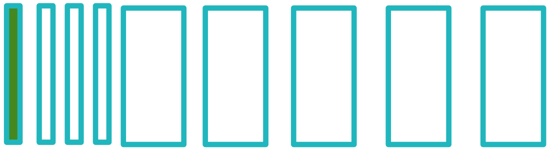
→ The estimator is finally re-fitted **on the whole training dataset** to determine the final **weights**.

Test vs. validation:
https://machinelearningmastery.com/difference-test-validation-datasets/

Machine Learning LinReg

Cross validation:

- **Extreme case**: *k* is close to the total number or samples:
  - "Leave-one-out" and singleton-tests.
  - All samples but one are trained in each trial (cf. minibatches of size 1 in DL context)



Tuning of hyperparameters through cross validation can be done **in parallel**.

```
https://spark.apache.org/docs/latest/ml-tuning.html#cross-validation
```

Machine Learning LinReg

For this example, the **RMSE**.

Cross validation:

To evaluate a model, we can consider prediction errors (the **evaluation metric**) on different datasets. For the **same model**, it *typically* holds:

- The error on the **training set** on which the model is finally trained $<$
- The error of the cross-validation (as **average** of several **holdout sets**) $\lesssim$
- The error on **one holdout set** $<$
- The unknown **truth error** on **test**, new data.

→ The first 3 errors are lower bounds of the truth error.

Requires the "ideal classifier".

https://webis.de/downloads/lecturenotes/machine-learning/unit-en-evaluating-effectiveness.pdf

Machine Learning LinReg

From the **main Notebook**:
Call the **ML pipeline** defined in handlers (*linReg*) to

- **fit** the model class to the **training** data and
- … return the obtained **model** (i.e., the weights) and the **evaluator** (i.e., RMSE)

→ This corresponds to **training** the model.

Not executed!

Since it takes long, **pre-trained models** have already been loaded for you to use.

Machine Learning LinReg

Now you can:

- Define the evaluator outside the pipeline (**EX 1**).
- Load the pre-trained ML model (**EX 2**).
- Call the **ML pipeline** *linReg**Test*** defined in the Handlers to
  - **Apply the model to predict the delays** on the test dataset,
  - **Evaluate** the quality of the prediction by computing the RMSE.

EX 2 is repeated twice: Over 1 feature set, and as a loop over 5 feature sets.

Hands-on = EX 1-2.
Stop at "End of ex. 2" (bar-plot).

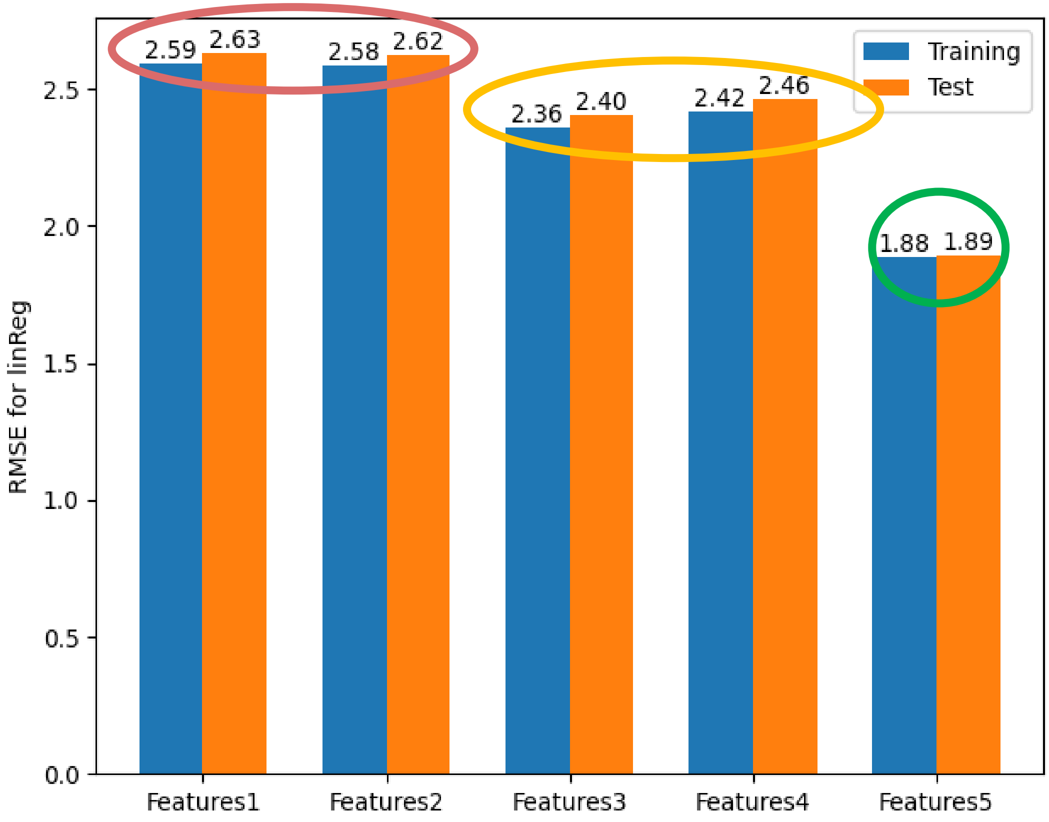
Machine Learning LinReg

3. **Evaluate** the regression algorithm (LR)

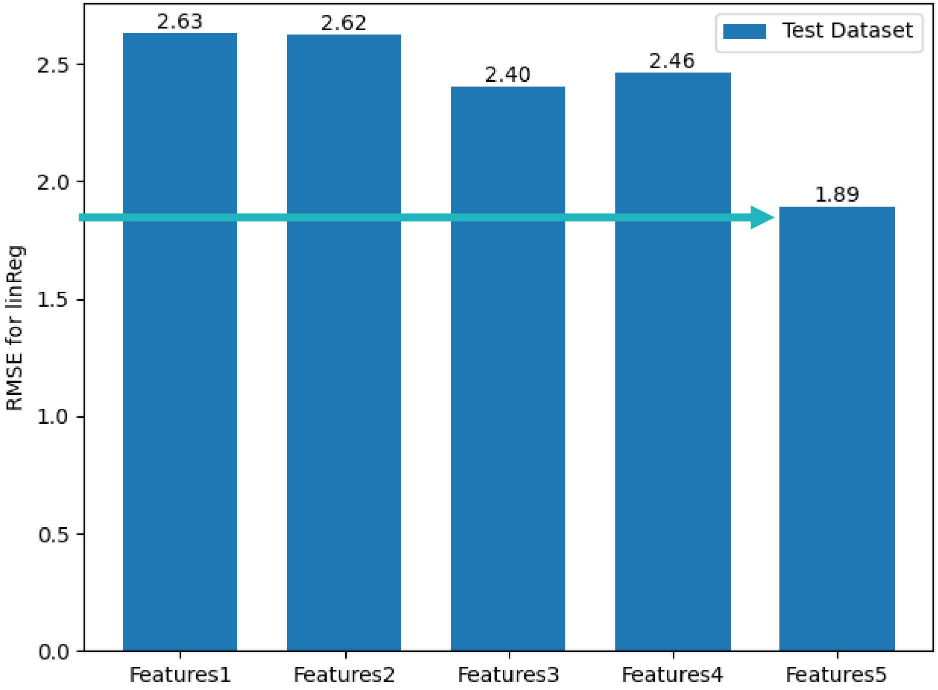
Which features give the best/worse RMSE?

1) Basic information of the original train dataset (5)
2) Basic information, weather data (11)
3) Delay at station -1 (derived features) (1)
4) Delay at stations -1,-2 (derived features) (2)
5) Basic information, delays -1, -2, and duration (derived features), weather data (8)

Number of feature columns

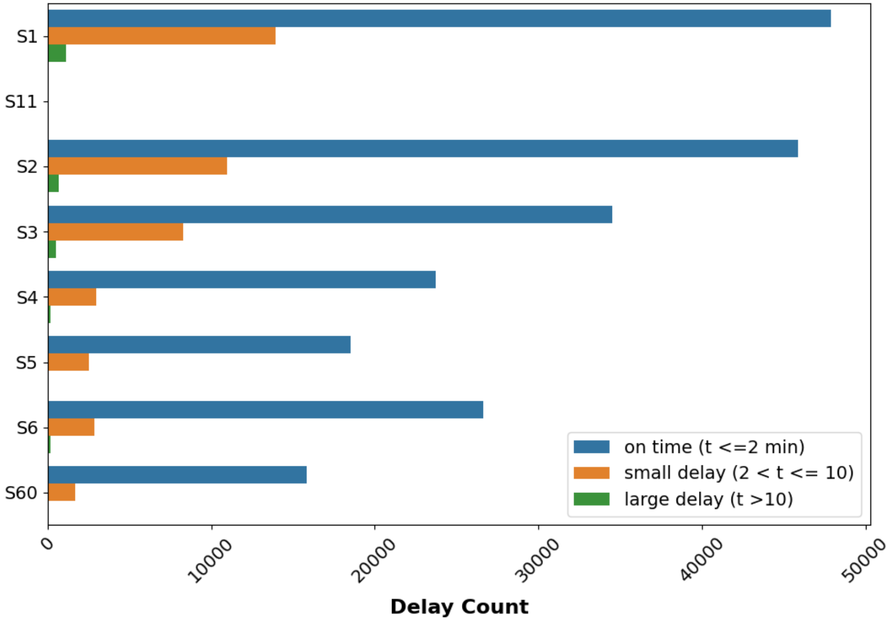


Machine Learning LinReg



- ML prediction: Best average error of ca. 1,5 minutes
- Most delays **below 2 minutes**

=> ☹



Machine Learning LinReg

Strategies to evaluate/optimise the ML algorithm (1)

- Choose different combinations of **features** → done
- Analyse the influence of ratio **training vs. test data** → **learning curve**
- Modify (e.g., filter) the samples → **feature engineering**

next slides

Next exercise

Machine Learning LinReg

Strategies to evaluate/optimise the ML algorithm (2)

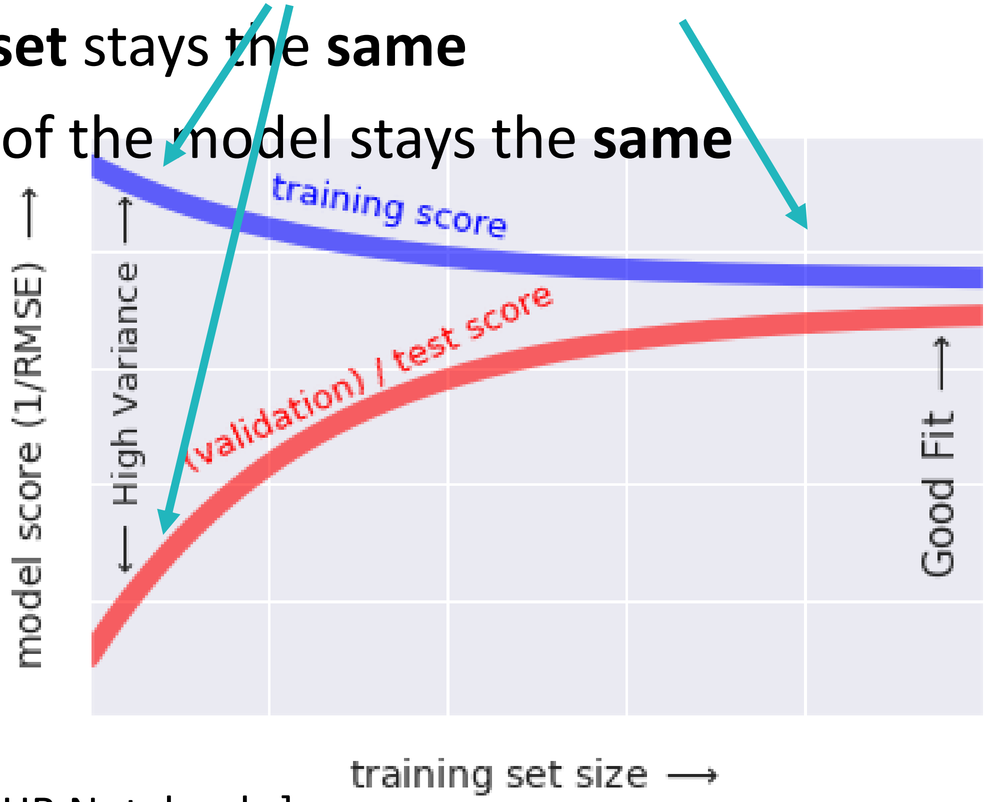
- Modify the architecture, e.g. by using different regression algorithms.
- Use a more appropriate **evaluator** (MSE, MAE, …) specific for the problem.
- Tune the **hyperparameters**: Regularisation parameters, number of iterations, …
  - **Heuristic** approach or cross-validation.
  - More complex models could lead to better results or to **overfitting**.

Machine Learning LinReg

Learning curve:

Ideal behaviour: From overfitting to convergence

- **Size of test set** stays the **same**
- **Complexity** of the model stays the **same**



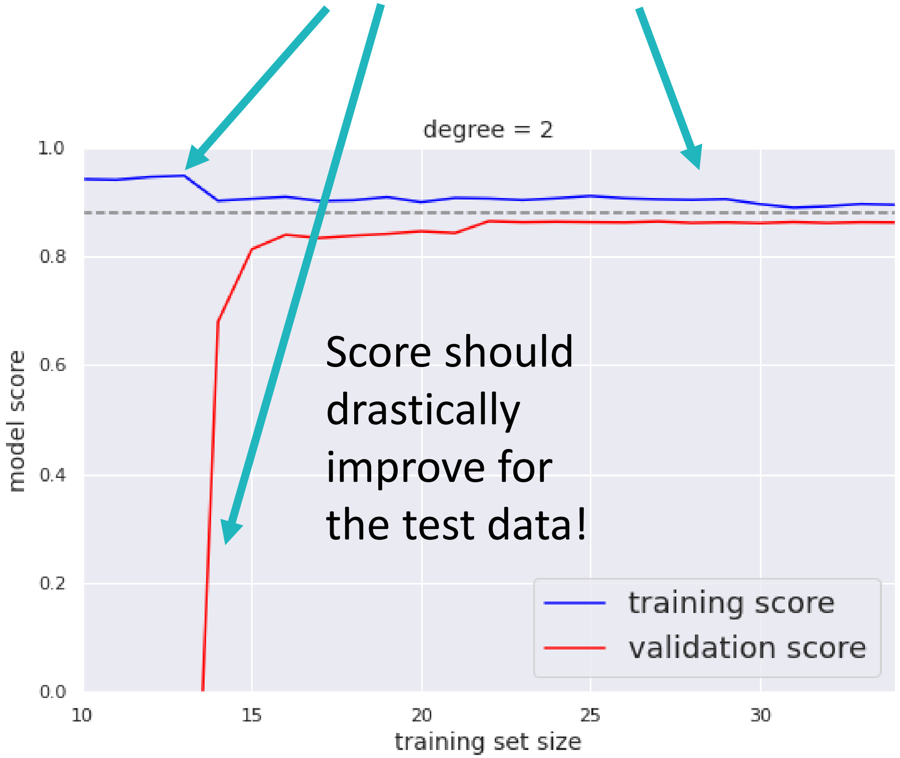
Evaluation metric: **Score** as inverse of the error (RMSE).

[code adapted from: PHB Notebooks]

Machine Learning LinReg

Learning curve:

A simple example: From overfitting to convergence



Score should drastically improve for the test data!

[code adapted from: PHB Notebooks]

Machine Learning LinReg

Learning curve:

In our example:

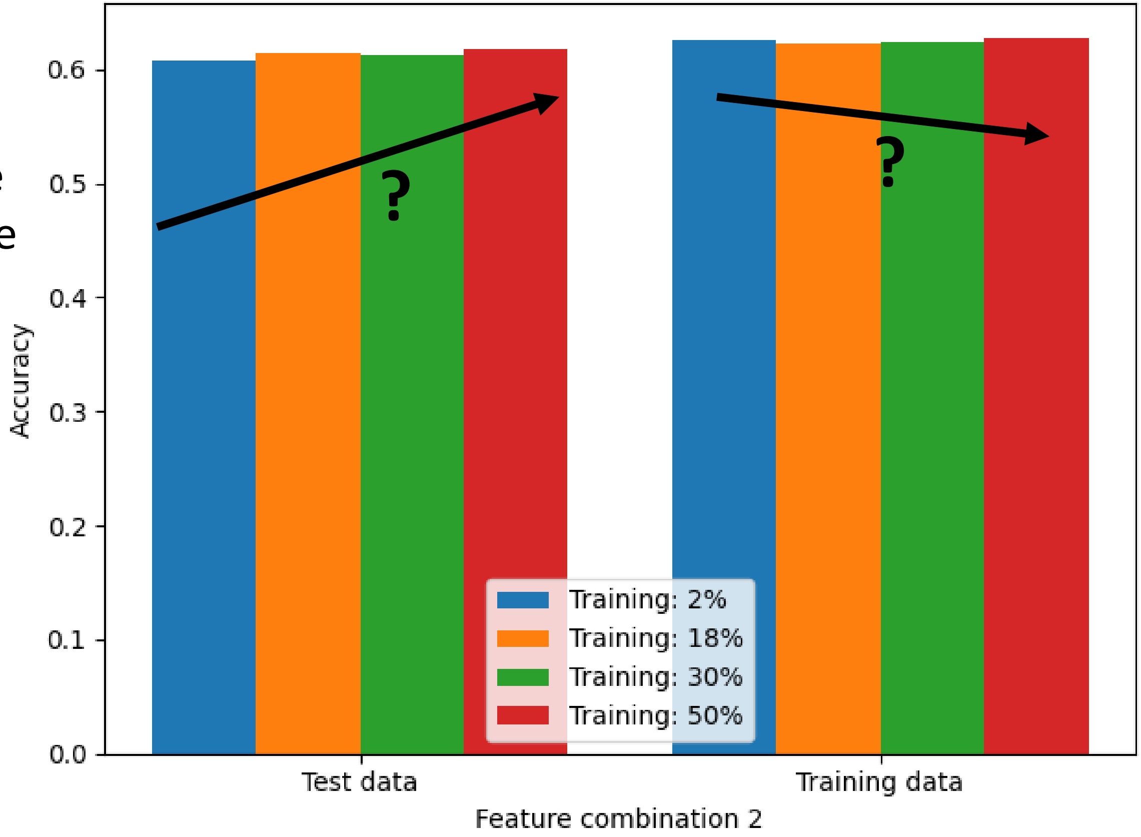
- **Test** set is fixed to 50% of the total data.
- **Training slices**:
  - From 50% of the total data, **training subsets** are obtained as further random splits.
  - They correspond to 2, 18, 30, 50% of the total data.
- The **random seed** is fixed for reproducibility.

Machine Learning LinReg

Learning curve:

Results for the <u>**classification** algorithm</u> (feature set 2):

The arrows represent the ideal behaviour, which we do not see!



Test Dataset: **Fixed** as 50% of the total data.
Random seed = 4561767182015543883

Machine Learning LinReg

Details mostly skipped

About the **learning curve** of training and test sets:

- **Loss function** during training could have unexpectedly(?) **higher values** than during validation (data unseen from the model).

Reasons in DL, e.g. **after each epoch**:

- **Values of the weights**: The training loss is computed at each batch, then averaged (the model improves with batches). The validation loss is computed only with the final model.
- **Structure of the model**: Regularisation mechanisms are turned on only at training time (dropout: some weights are turned off to zero).

https://keras.io/getting_started/faq/#why-is-my-training-loss-much-higher-than-my-testing-loss

Machine Learning LinReg

Details mostly skipped

In ML with Spark:

- After fitting with cross validation, the **bestModel** of CrossValidatorModel is available for prediction, I/O of the best fitted model etc.
- Collecting **other sub-models** while training at cross validation is also possible. To do that, switch on the attribute **collectSubModels** of the CrossValidator estimator.
  This may cause large memory consumption!

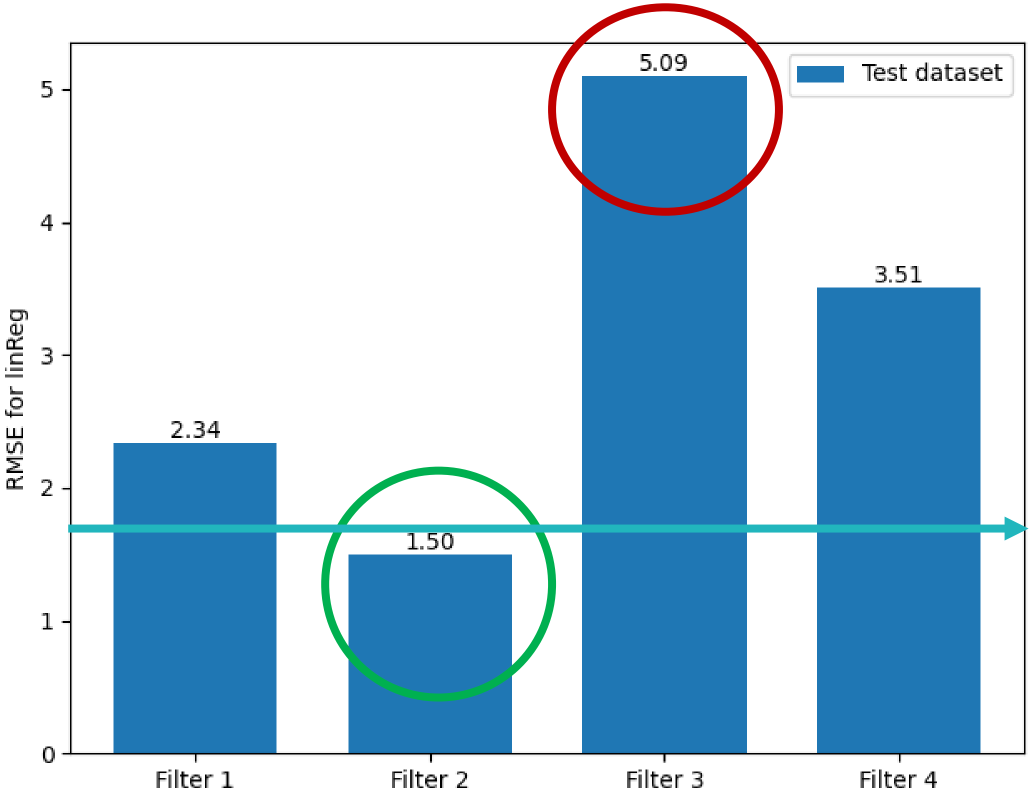
→ We only use the resulting **bestModel** in the examples.

https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.tuning.CrossValidator.html

Machine Learning LinReg

Modify (e.g., filter) the samples:

- Filter both the training and test datasets: **EX 3** → **You can use the solution!**
  - Filter 1: S-Bahn line 1 only
  - Filter 2: Only Wednesday
  - Filter 3: Delay > 5 mins
  - Filter 4: Combination of F1-3
- (Train or) read-in 4 new models.
- **Inference** on the test data set.
- **Plot** the prediction error: **EX 4**.



Focus on Pre-processing, Feature Engineering and Machine Learning

- Part II: Example on the Jupyter Notebooks
  - Pre-processing
  - Supervised learning techniques in a Machine Learning pipeline
    - « Notebook 1 (Ex.)
    - « Notebook 2 (Ex.)
    - « Notebook 3 (Ex.)
    - « Notebook 4 (Ex.)
    - « Notebook 5 (Ex.)

More learning outcomes

Main index

Part I
Part II
Part III

Notebook 4: **NB4_class.ipynb**

- Jupyter Notebook how-to, see slides: https://fs.hlrs.de/projects/par/events/2024/dl-hlrs/DL-HLRS-day1-exercises.pdf
- A few slides to sum up the content of **this** Notebook follow.

Machine Learning RandomForest

Through linear regression, we predicted the delay:

- in minutes,
- on the dataset as a whole.

We aim now to predict:

- the delay **as class** {yes, no},
- on the data
  - of the line **S1** (first splitting block in the Notebook)…
  - clustered by **station** (second splitting block in the Notebook).
- The **training** will be done on the complete dataset.

→ **Goal:**

Can we predict a delayed departure from **a particular station** of the line S1?

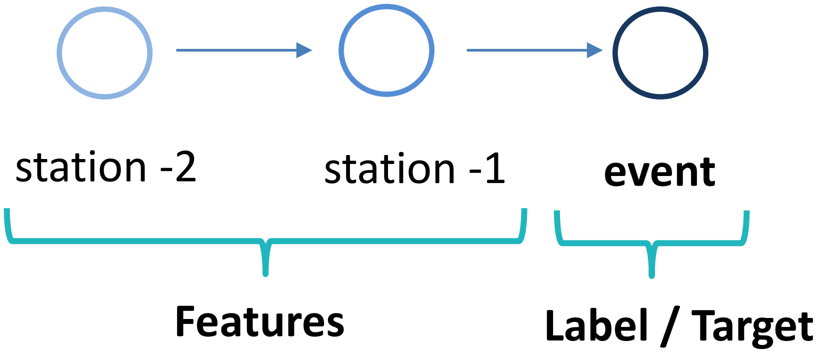
Machine Learning RandomForest

1. Read-in the data from manipulation

DataFrames after Manipulation:

- **Training**: *df_train_classification* (**50%** of all data)
- **Test**: *df_test_classification* (remaining **50%**)

which contain the columns delay {y, n} at the stations:



Machine Learning RandomForest

1. Read-in the data from manipulation

For the evaluation after training:

- Select the **data corresponding to the line S1**: *df_test_S1, df_train_S1*
- Create a **list of DataFrames**, one DF for each station: *df_arr_test_ds, df_arr_train_ds*

EX 1

All lines in the dataset: work in progress…

Machine Learning RandomForest

2. Run the **classification** algorithm

- Define the **feature columns** and
  the target, different than in LinearRegression:
  ~~delay in minutes~~ → delay {y, n} at the stations (0, -1, -2)
- Execute the **ML pipeline**:
  Similar to the LinearRegression one
  → step by step in the next slides.

Machine Learning RandomForest

→ Please do EX1 & all steps until “Please stop here (introduction)”

→ Then, you can open the **handlers.ipynb** Notebook and go to the function **clfcTrain (Classification Pipeline)**.

You do not need to modify/execute the handlers.
The ML workflow corresponds to the steps in the handlers.

Machine Learning RandomForest

The **ML pipeline** for classification is defined in the handlers (*clfcTrain*) and takes as **arguments** the training DataFrame and the feature columns.

Feature Engineering steps:

https://spark.apache.org/docs/latest/ml-features

- **StringIndexer**
- **VectorAssembler**
- **Normaliser**

} same as LinearRegression

Machine Learning RandomForest

Define the **estimator (= the architecture of the model):**

https://spark.apache.org/docs/latest/ml-classification-regression.html#random-forest-classifier

https://spark.apache.org/docs/latest/mllib-ensembles.html#random-forests

- Define the model **class** as:
  - `RandomForestClassifier`
  - with the model **hyperparameters** (`numTrees`: see next slides),
  - specifying the **features** and the **label / target**.

- All steps so far (feature engineering + model definition) are collected into a **pipeline**.

Machine Learning RandomForest

Define the **evaluator:**

- **MulticlassClassificationEvaluator**:
  Evaluation Metric is the **accuracy**:

$$\frac{1}{N}\sum_{i=1}^{N}\delta_{0(y_i-\hat{y}_i)},$$

1 if the prediction is correct, **0** otherwise

where:

- $\mathbf{y}$: predictions vs. $\hat{\mathbf{y}}$: true output (one-dimensional vectors of integers)
- with N entries: N **number of samples**
  (e.g., for the whole dataset, or for each station of the S1 line).

Machine Learning RandomForest

- The **MulticlassClassificationEvaluator** can be used in case of multiple choices, e.g. digits {1, 2, ..., 10}, or as binary evaluator {0, 1} .

https://spark.apache.org/docs/latest/mllib-evaluation-metrics.html

Machine Learning RandomForest

Define the **evaluator:**

- Through the **accuracy**, we do not seek to perform better in evaluating either late or on-time.
- For particular applications, one tries to improve how the model classifies **one class** only (e.g. undetected **defected** artefacts: false negatives)
- Based on the confusion matrix, other **evaluation measures** are defined:

$$\text{precision} = \frac{\text{TP}}{\text{TP+FP}} \qquad \text{recall} = \frac{\text{TP}}{\text{TP+FN}}$$

		CLASSIFIER	
		Positive	Negative
ACTUAL	Positive	**True positive**	**False negative**
	Negative	**False positive**	**True negative**

Machine Learning RandomForest

Details of the ML algorithm: **Decision Tree**

Ideal/True classifier: E.g. the combination of an automatic inspection machine and a human annotator (costs, reduced throughput, …).

Training dataset

Weather	Day of Week	Line	Duration	Pollution	Delay
Rainy	Monday	S2	35	Low	Delayed
Storm	Tuesday	S3	40	High	On time
Sunny	Thursday	S60	30	Medium	On time

…

$\mathbb{X}$

Set of **classes** ($\mathbb{C}$)

On time

Delayed

On time

Set of **feature vectors** ($\mathbb{X}$)

Ideal classifier

Decision Tree

- Ideal classifier ($\mathbf{x}$,c)
- A decision tree is one **approximation** of the ideal classifier.

https://webis.de/downloads/lecturenotes/machine-learning/unit-en-decision-trees-basics.pdf

Machine Learning RandomForest

Decision Tree: Feature-based Splitting

- The set $\mathbb{X}$ of feature vectors is decomposed into disjoint / unique sets.
- The **root** and each **non-leaf node** defines a (non-)**binary** splitting of **a feature** of $\mathbb{X}$ (e.g. all *Storm+Rainy* **and** *Sunny*, …).
- For each feature vector $\mathbf{x}$, there is a **unique path** from the root to a leaf node.
- **Monothetic tree**: One feature at a time is evaluated at non-leaf nodes.

Weather	Day of Week	Line	Duration	Pollution	Delay
Rainy	Monday	S2	35	Low	Delayed
Storm	Tuesday	S3	40	High	On time
Sunny	Thursday	S60	30	Medium	On time

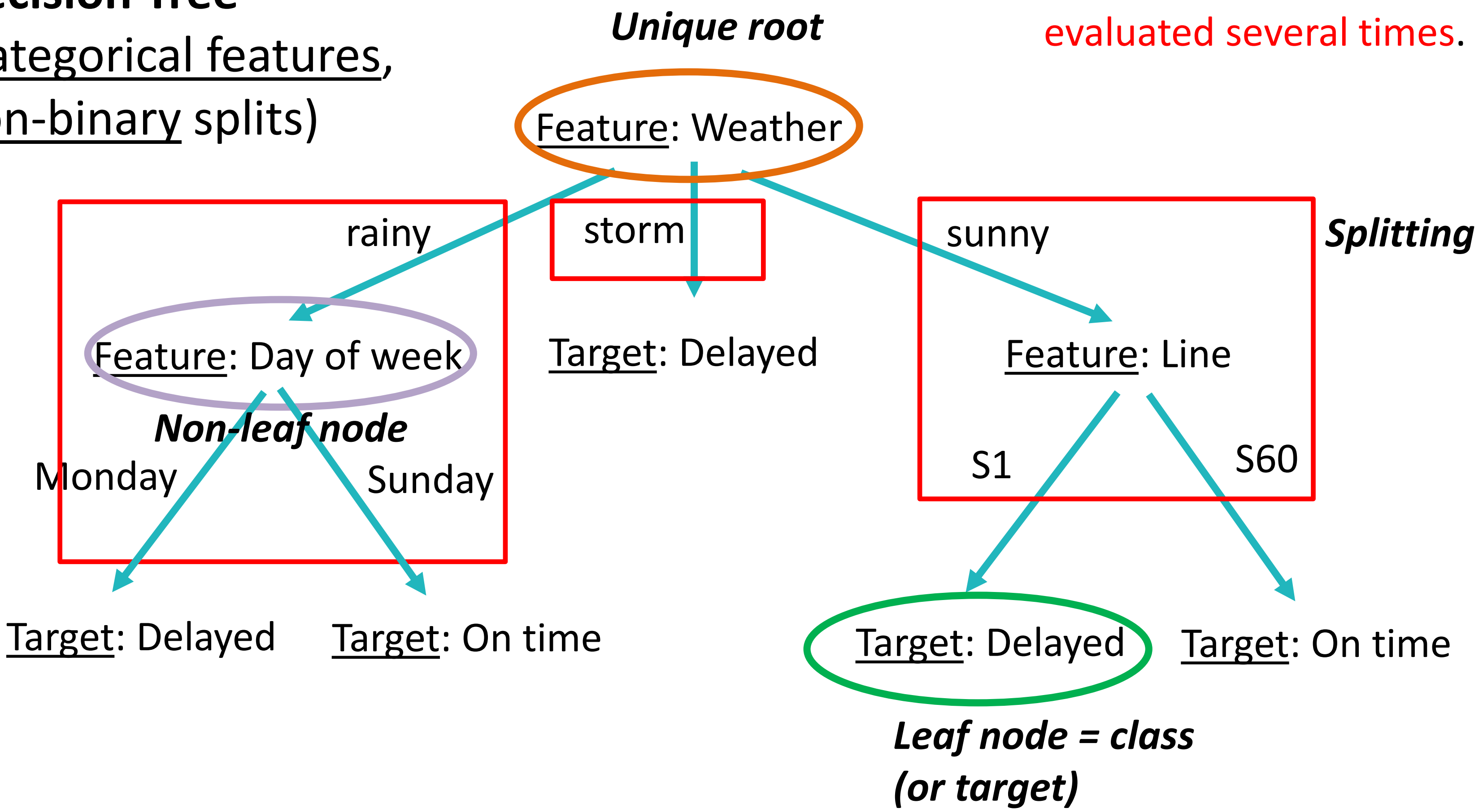
…

https://webis.de/downloads/lecturenotes/machine-learning/unit-en-decision-trees-basics.pdf

Machine Learning RandomForest

- In Spark, the split is always binary.
- The same feature can be evaluated several times.

Decision Tree
(categorical features, non-binary splits)



Machine Learning RandomForest

Decision Tree

- **Hypothesis space**: Given a set of examples (training set of **feature vectors** + corresp. **classes**) → set of possible decision trees.
- How to **evaluate a tree?** Minimize:
  1. Classification error (→ correct association of feature vector and class)
  2. Size of the tree

Criteria for the **size** of the tree:

- number of leaf nodes (5)
- tree height (= number of evaluations: 3)
- (weighted) path length: sum of all lengths of all paths between the root and any leaf (2*4 +1 = 9)
- **depth**: maximum path length (2)

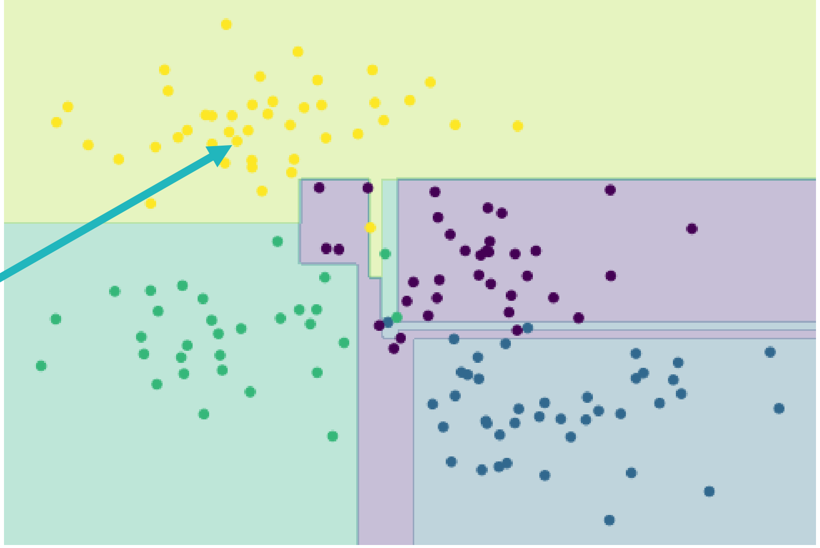
https://webis.de/downloads/lecturenotes/machine-learning/unit-en-decision-trees-basics.pdf

Machine Learning RandomForest

Decision Tree: Overfitting

Continuous features, binary splits:

- Each decision tree is constructed during the **training phase**.



- Each **point** in the picture (sample) = **one feature vector** `(x,y)` with their class (colour).
- The space of feature vectors (all points) is **iteratively** split according to the coordinate values.

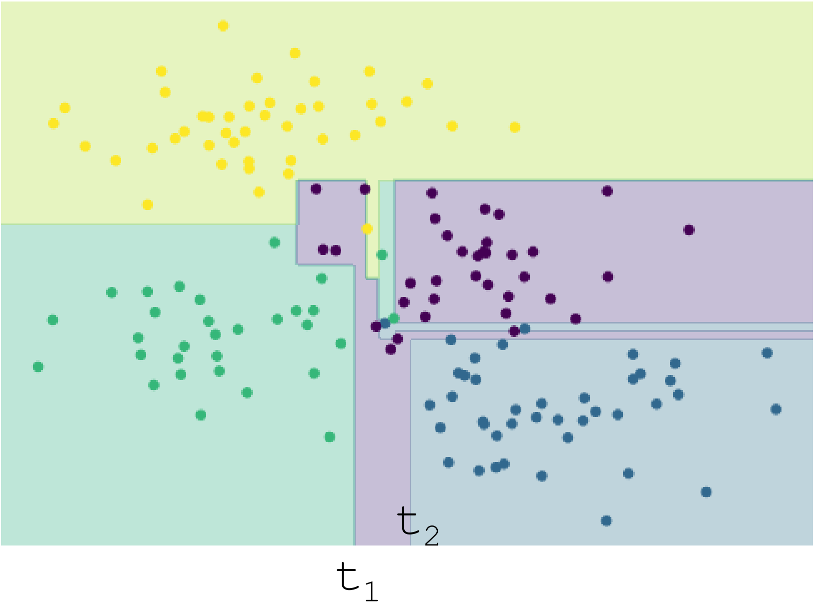
[T. Hastie et al., The Elements of Statistical Learning, Springer]

[code adapted from: PHB Notebooks]

Machine Learning RandomForest

Decision Tree: Overfitting

- Features with continuous values can be evaluated many times in the tree, e.g.
  $x \leq t_1$,
  $t_1 < x \leq t_2, x > t_2$ …



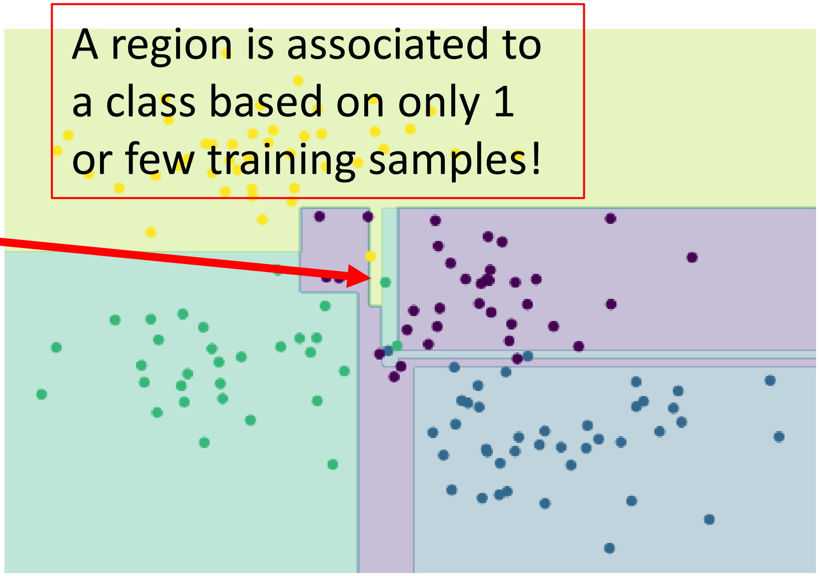
- At the end of the splitting, a **class** is associated to each region (in the picture: 4 classes = colours; in our case: delay: {yes, no})

[code adapted from: PHB Notebooks]

Machine Learning RandomForest

Decision Tree: Overfitting

- After **some** iterations, noisy or non-representative data are considered (e.g., an **overcrowded journey on a peak-hour on a rainy day,** that is **on time**).
- This generates **overfitting**!
- It is hard to establish **how many iterations** are needed before overfitting is reached!



A region is associated to a class based on only 1 or few training samples!

- Improvement is needed where the classification is **non-unambiguous**, i.e. where the clusters get close together.

[code adapted from: PHB Notebooks]

Machine Learning RandomForest

Decision trees: Weak classifiers

Not only overfitting…:

- Decision trees with fixed number of leaves are **unstable**: A small change in the training data implies a significant change in the resulting classifier.
- From the same training data, several decision trees can be derived as classifiers with comparable accuracy (**computational problem**).
- The learning procedure can hardly detect the **optimal classifier**.
- Decision trees cannot reach an acceptable degree of **accuracy**.

https://webis.de/downloads/lecturenotes/machine-learning/unit-de-ensemble-methods-basics.pdf

Machine Learning RandomForest

Ensemble Methods

Goal:
Counterbalance the disadvantages of individual classifiers and improve their performance using the information of a **group** of classifiers **of the same kind**.

Challenges:

- During training, the **same algorithm** to construct a decision tree will produce **the same tree** for the **same dataset**.
  → **Different datasets** are needed to have **different classifiers**.
- How to extract **one decision** from many classifiers?

https://webis.de/downloads/lecturenotes/machine-learning/unit-de-ensemble-methods-basics.pdf

Machine Learning RandomForest

Ensemble Methods

Solution :

Details skipped

- Produce different training sets through ***bootstrap aggregating*** (non-overlapping subsets of the main set, cf. cross-validation).
- The final decision is taken by a ***majority vote***.

next slide

Bagging

Other ensemble methods (different: training sets and final decision):

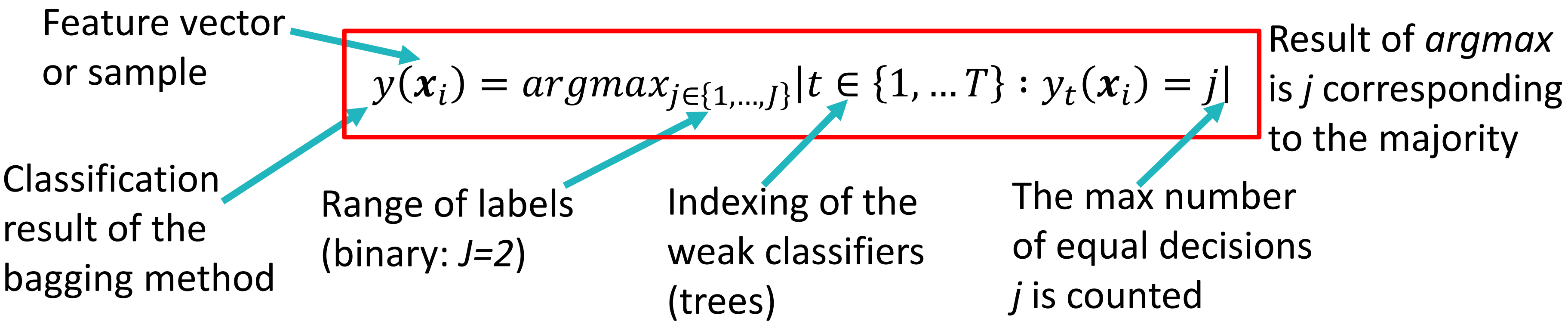
Adaptive Boosting, Cascading.

https://webis.de/downloads/lecturenotes/machine-learning/unit-de-ensemble-methods-basics.pdf

Machine Learning RandomForest

Prediction by majority vote:

Weath.	W-day	Line	Dura-tion	Poll.	Delay	Tree1	Tree2	Tree3	Prediction
Rainy	Mo	S2	35	Low	**Delayed**	D	O	O	67% O
Storm	Tue	S3	40	High	**On time**	O	O	D	67% O
Sunny	Thu	S60	30	Med	**On time**	O	O	O	100% O



Feature vector or sample

$$\boxed{y(\boldsymbol{x}_i) = argmax_{j\in\{1,\ldots,J\}} |t \in \{1,\ldots T\} : y_t(\boldsymbol{x}_i) = j|}$$

Result of *argmax* is *j* corresponding to the majority

Classification result of the bagging method

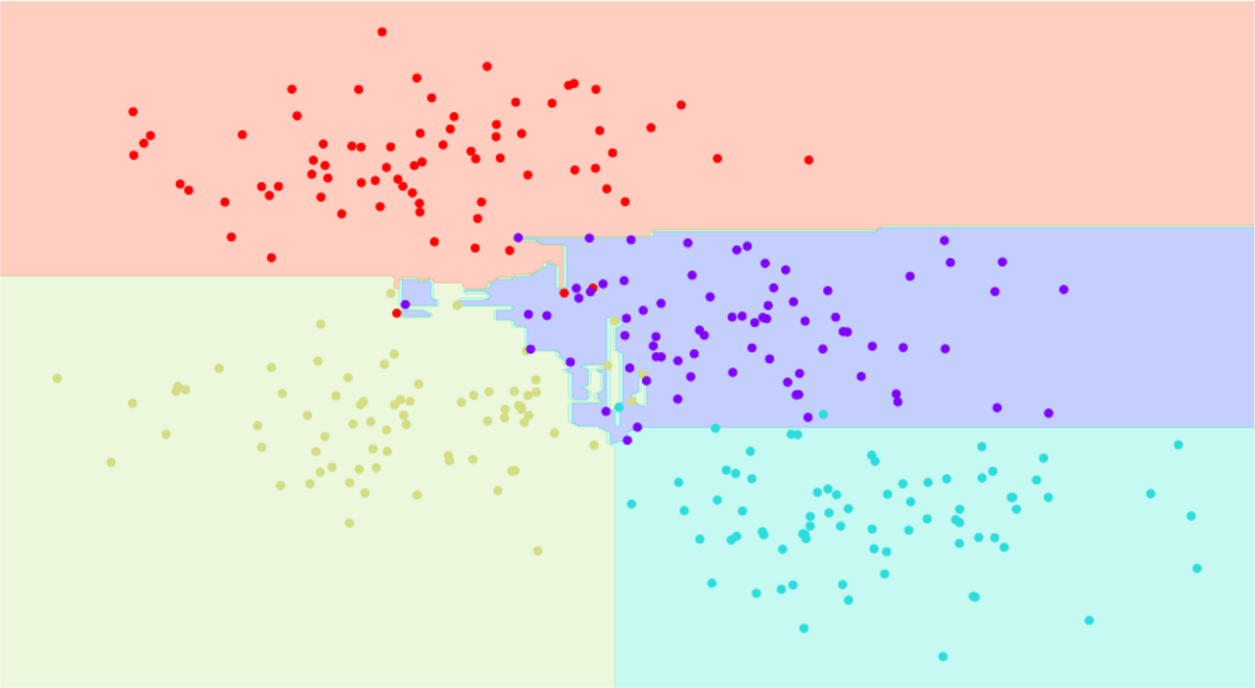
Range of labels (binary: *J=2*)

Indexing of the weak classifiers (trees)

The max number of equal decisions *j* is counted

Machine Learning RandomForest

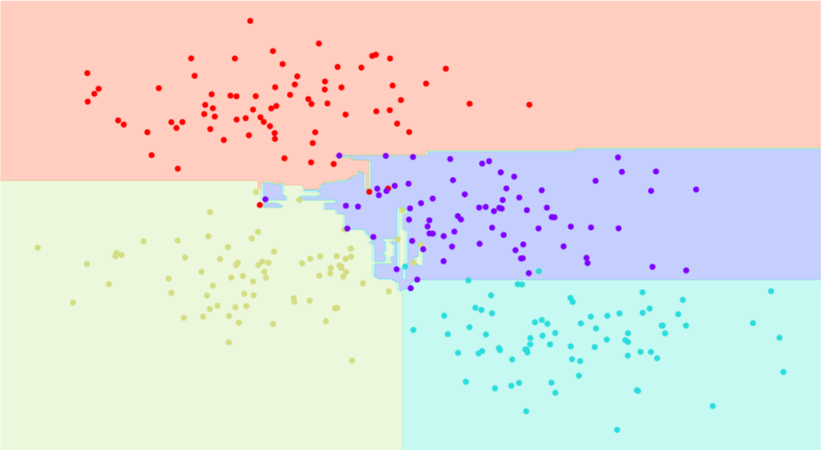
- **Bagging**:
  - An ensemble of trees are trained (possibly in **parallel**),
  - Classification of the samples through *majority vote*.



[PHB pp. 421-432 + code adapted from: PHB Notebooks]

Machine Learning RandomForest

Random Forest



- A **stochastic component** is crucial for a good outcome of **training**:

A (random) **subset** of → **the training dataset** is fitted by each tree.

A (random) **subset** of → **features** is considered for splitting by each tree.

- Both are done automatically in **RandomForest**:
  - **Fast** method (training and prediction), given the simplicity of decision trees.
  - Available as both RandomForest**Classifier** and RandomForest**Regressor**.

[PHB pp. 421-432 + code adapted from: PHB Notebooks]

Details skipped

Machine Learning RandomForest

Additional topics…

- **Algorithms** for decision trees.
- **Which problems** are suitable for decision trees / random forests?
- Other **ensemble methods**.
- …

https://webis.de/downloads/lecturenotes/machine-learning/unit-de-ensemble-methods-basics.pdf
https://webis.de/downloads/lecturenotes/machine-learning/unit-en-decision-trees-algorithms.pdf

Machine Learning RandomForest

In the handlers Notebook…

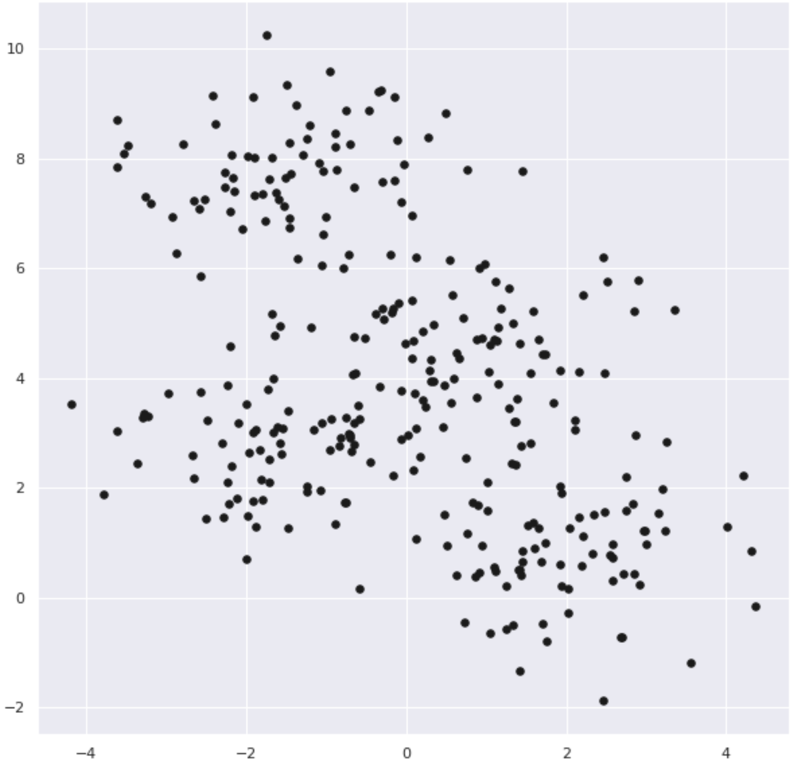
Define the **validation:**

- Define the **parameter grid: none** in this case
  - **`numTrees`** (here: 100):
    Number of trees increase→ (Linear) increase of compute time & accuracy
  - **`maxDepth`** (here: 5 (default)):
    Max depth of each tree can be tuned to produce some degree of overfitting.

Machine Learning RandomForest

Digression: **Unsupervised learning →** Generative classifier

Additional hyperparameters as the **number of clusters** can be set in classification tasks (e.g. in partitioning, hierarchical, and spectral algorithms).



→ Same constellation as in the previous examples.
→ How many clusters?

- Other **hyperparameters**: size and radius of clusters, …

Machine Learning RandomForest

Define the **cross validation:**

- Define the **cross validation**, to cross-validate `N`-folds over the training dataset (cf. Linear Regression). In our case, `N=`**`10`** **models** are being trained.

In the code, the **`CrossValidator`** object contains the whole pipeline (estimator, parameter grid, evaluator) and calls the **`fit`** over the training data.

→ This step could be skipped without hyperparameter map!

Machine Learning RandomForest

In the main Notebook…

From the **main Notebook**:

Call the **ML pipeline** defined in the handlers (`clfcTrain`) to

- **fit** the model class to the **training** data, and
- … return the obtained **model** (the weights) and the **evaluator** (the accuracy).

→ This corresponds to the **training** of the model (full training dataset).

Not executed!

Since it takes long, **pre-trained models** have already been loaded for you to use.

Machine Learning RandomForest

Now you can:

- Define the evaluator outside the pipeline (**EX 2**).
- Load the pre-trained ML model.
- Call the **ML pipeline** `clfcTest` defined in the Handlers to:
  - **Apply the model to predict the delay classification** on the test dataset,
  - **Evaluate** the quality of the prediction by computing the accuracy.

→ The last step corresponds to the **evaluation** of the model (one S1 DataFrame for each train station).

Machine Learning RandomForest

Hands-on

- EX2 → “basic”
- EX3 → “advanced” (Python)

Proceed until the **end of the Notebook** (bar-plot).

Machine Learning RandomForest

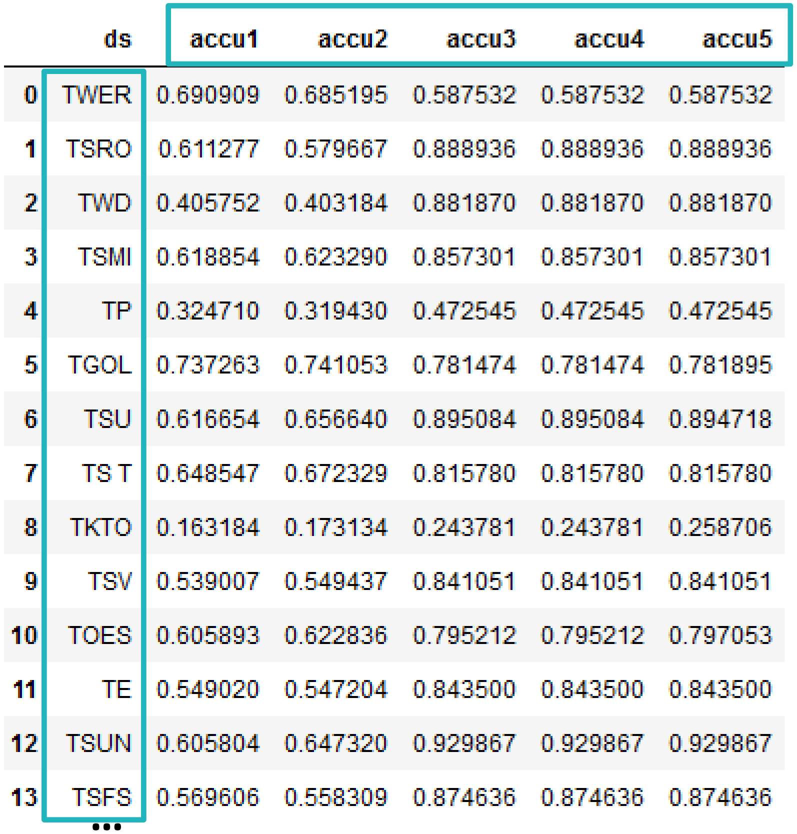
3. Evaluate the **classification** algorithm

EX 3 → option to use the solution

Goal: Collect the accuracy results (a **2D list**) in a DataFrame ordered by

- **feature combination** (columns),
- **train station** (rows),

for the **test** dataset.



	ds	accu1	accu2	accu3	accu4	accu5
0	TWER	0.690909	0.685195	0.587532	0.587532	0.587532
1	TSRO	0.611277	0.579667	0.888936	0.888936	0.888936
2	TWD	0.405752	0.403184	0.881870	0.881870	0.881870
3	TSMI	0.618854	0.623290	0.857301	0.857301	0.857301
4	TP	0.324710	0.319430	0.472545	0.472545	0.472545
5	TGOL	0.737263	0.741053	0.781474	0.781474	0.781895
6	TSU	0.616654	0.656640	0.895084	0.895084	0.894718
7	TS T	0.648547	0.672329	0.815780	0.815780	0.815780
8	TKTO	0.163184	0.173134	0.243781	0.243781	0.258706
9	TSV	0.539007	0.549437	0.841051	0.841051	0.841051
10	TOES	0.605893	0.622836	0.795212	0.795212	0.797053
11	TE	0.549020	0.547204	0.843500	0.843500	0.843500
12	TSUN	0.605804	0.647320	0.929867	0.929867	0.929867
13	TSFS	0.569606	0.558309	0.874636	0.874636	0.874636

…

Machine Learning RandomForest

Visualise the results of the classification **accuracy**:

- **Averaged** accuracy of the **test data** for the 5 different feature sets (→ next slide);
- **Clustered by stations**, on a geographic map: dedicated visualisation section in Notebook 5.

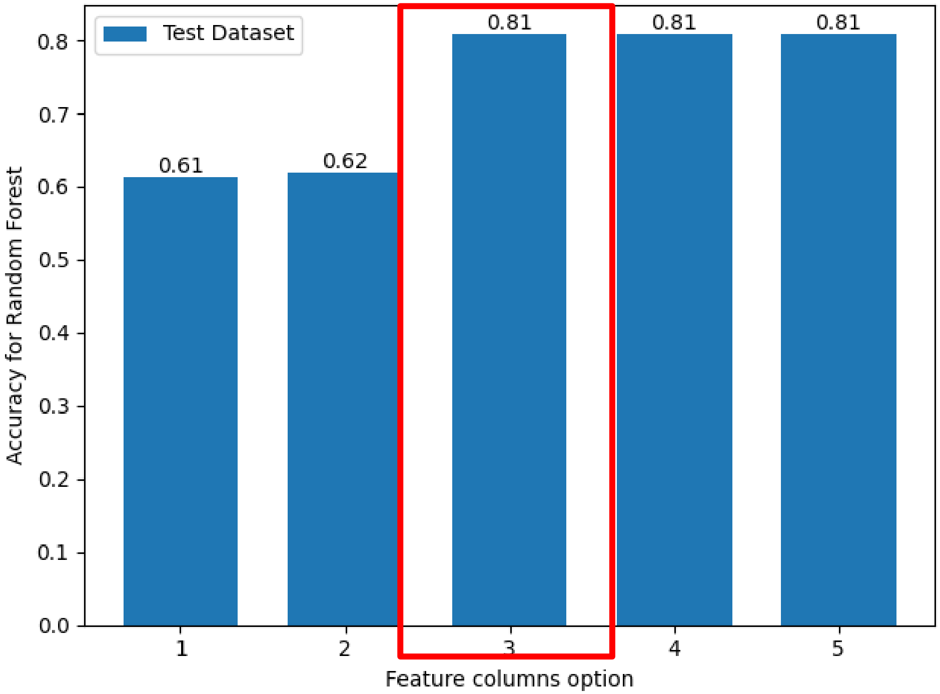
Machine Learning RandomForest

We selected 5 sets of **feature columns** to run the model in several combinations:

1) Basic information of the original train dataset (5)
2) Basic information, weather data (11)
3) Delay at station -1 (derived features) (1)
4) Delay at stations -1, -2 (derived features) (2)
5) Basic information, delays and duration (derived features), weather data (8)

Machine Learning RandomForest

- Options 3, 4, 5 give **very similar** results, better than options 1, 2.
- Options 3, 4, 5 include the delay at the previous station(s)…
  - … which allows only for **inference at short-term**!
- Option 3 contains 1 feature column (= delay at station -1). It is therefore the “cheapest” (training runtime and memory).



Machine Learning RandomForest

Outlook in the choice of the **model class**:

- **ML**: More advanced (and comp. expensive) regression models or classifiers such as SVM [PHB from p. 405].
- **DL**:
  - LSTM (Long Short-Term Memory) Networks for prediction of the continuous delay,
  - Convolutional and Recurrent Neural Networks for classification.

For **this example**, DL results were not significantly better than ML ones and will not be discussed.

~~Work in progress~~

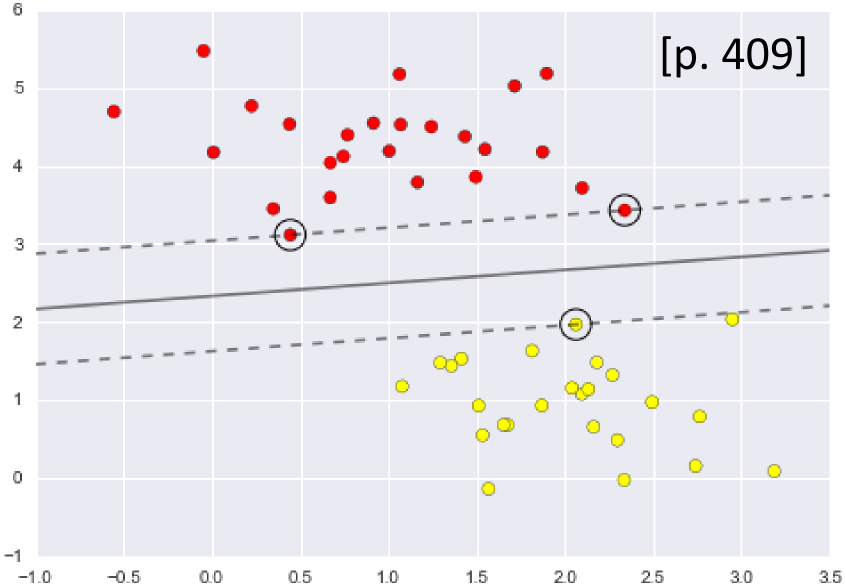
DL Tools → Day 2

Tests run in 2019.

SVM

Support Vector Machines:

- Powerful classification method
- 2D: Maximise the margin between two sets
- Points lying on the margins: support vectors
- Points far from the margins do not contribute to the loss function:
  - *compact* model, fast *prediction*
  - suitable for high dimension



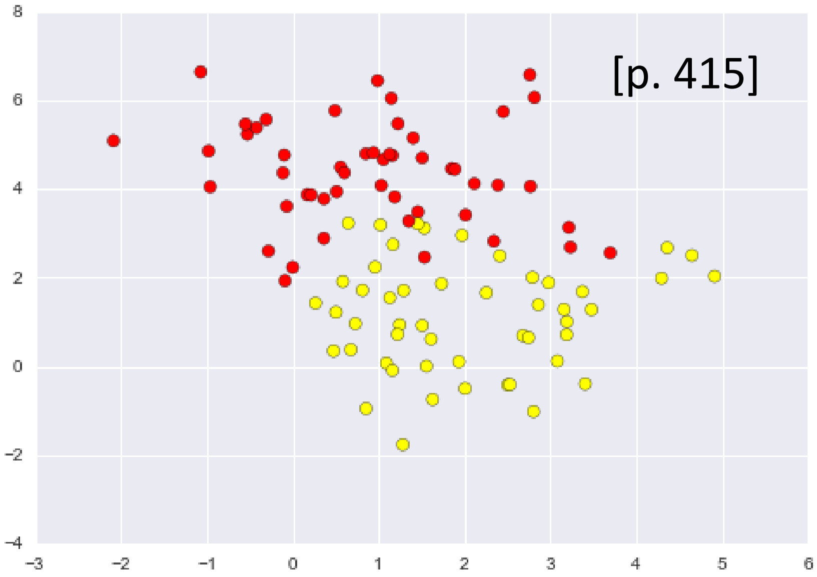
[p. 409]

Details skipped

SVM

Support Vector Machines:

- Can also be applied to *overlapping* datasets (i.e. similar features, different label): Softening parameter



[p. 415]

- Can be combined with *kernels* to go beyond the linear case (and still be efficient)
- Training phase: expensive, cross-validation necessary!

Details mostly skipped

Machine Learning RandomForest

(Digression) Interpretability: Coupling of ML and DL methods:

- Problem of NN as “**black boxes**“:
  - Examples: selection processes (personnel or at the bank…) or grading.
  - Make NN more transparent via ML methods (e.g., extract a decision tree from a NN).
- Transparency from data collection and manipulation to visualisation?

https://www.iff.uni-stuttgart.de/

Roscher et al., “Explainable ML for Scientific Discoveries”, 2020.

Machine Learning Scalability

Goal: Parallelise and improve the scalability of ML algorithms.

Performance on Urika-GX (system shut down on Feb 1st, 2021):

- Hadoop YARN resource manager
- Parallel I/O with HDFS
- Local disk space (scratch) for Spark communication

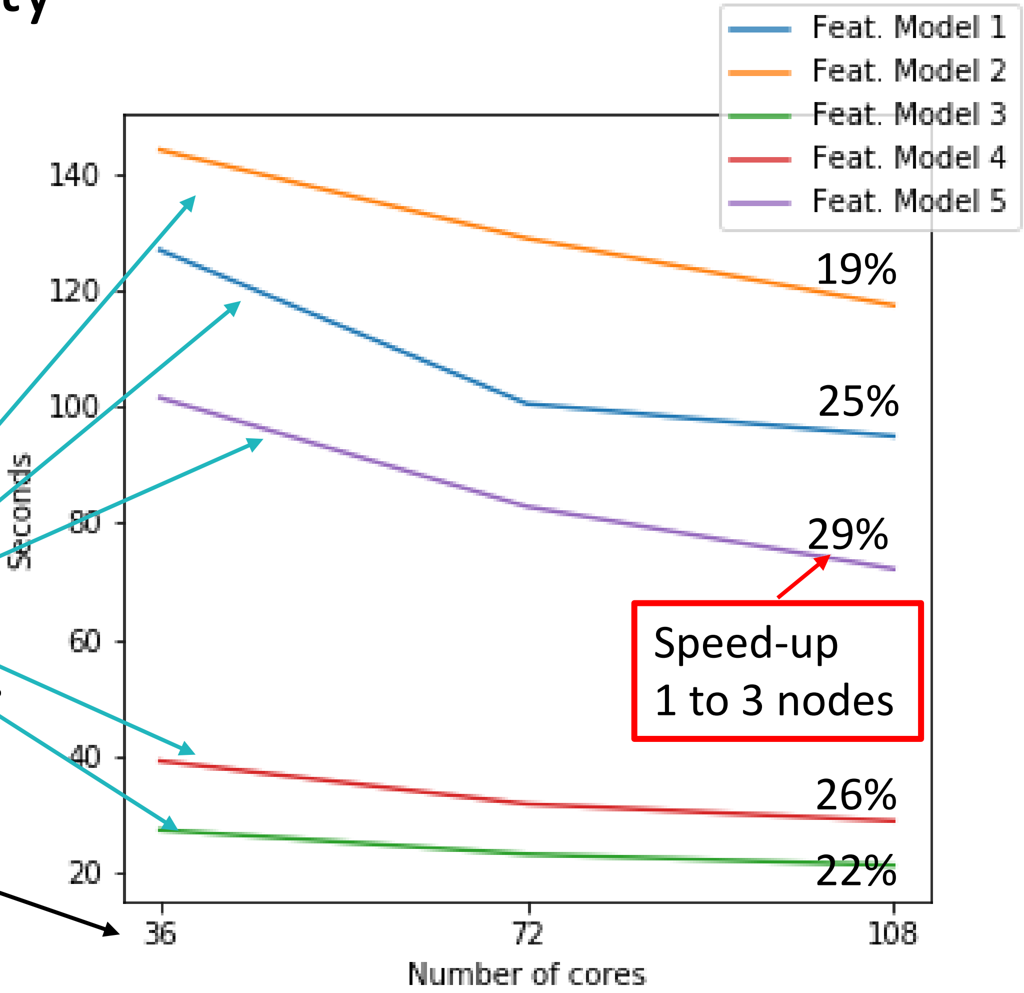
Optimised: No bottlenecks and successful scalability.

- **3, 2, 1** nodes à **36 cores** have been used respectively to **train the random forest**:
  - %spark **108** 450g
  - %spark **72** 450g
  - %spark **36** 450g

Machine Learning Scalability

Performance (strong scaling) of the classification algorithm:

- Average time (sec.) of 3x **training** on the same data,
- for 5 feature combinations.
- **1,2,3 nodes** in parallel with Spark.



Speed-up
1 to 3 nodes

Results obtained with Urika-GX Q1/2020

Machine Learning Scalability

Disclaimer on this benchmark:
The performance depends on the number of cores in a Spark session, but also…

- on the **communication overhead** between the **nodes**:
  max. efficiency by using all cores inside each node (**this case**),
- on the **balance** between dimension of the dataset and number of threads,
- on the **local scratch** available for Spark.

`https://researchcomputing.princeton.edu/faq/how-do-i-use-local-scratc`

Machine Learning Scalability

… Using instead **compute nodes** on **Vulcan/Training Cluster** :

- **clx-25** or **-21**: CascadeLake (Intel) 2x20 core-CPU
- **hsw**: Haswell (Intel) 2x10 or 12 core-CPU

on the **TC**:

- **skl**: Skylake (Intel) 2x20 core-CPU
- **clx-ai**: 2x18 core-CPU, and **GPUs**

clx-21 and **clx-ai** are CS-Storm nodes with larger local storage:
https://kb.hlrs.de/platforms/index.php/Urika_CS
https://www.hlrs.de/solutions/systems/cray-cs-storm

- No Hadoop YARN resource manager.
- No I/O with HDFS.
- Little local storage: Cf. this slide

Scalability results could not be reproduced.

Work in progress…

https://kb.hlrs.de/platforms/index.php/NEC_Cluster_Hardware_and_Architecture_(vulcan)

Focus on Pre-processing, Feature Engineering and Machine Learning

- Part II: Example on the Jupyter Notebooks
  - Pre-processing
  - Supervised learning techniques in a Machine Learning pipeline
    - « Notebook 1 (Ex.)
    - « Notebook 2 (Ex.)
    - « Notebook 3 (Ex.)
    - « Notebook 4 (Ex.)
    - « Notebook 5 (Ex.)

More learning outcomes

Main index

Part I
Part II
Part III

Notebook 5: **NB5_vis-class.ipynb**

- Jupyter Notebook how-to, see slides: https://fs.hlrs.de/projects/par/events/2024/dl-hlrs/DL-HLRS-day1-exercises.pdf
- A few slides to sum up the content of **this** Notebook follow.

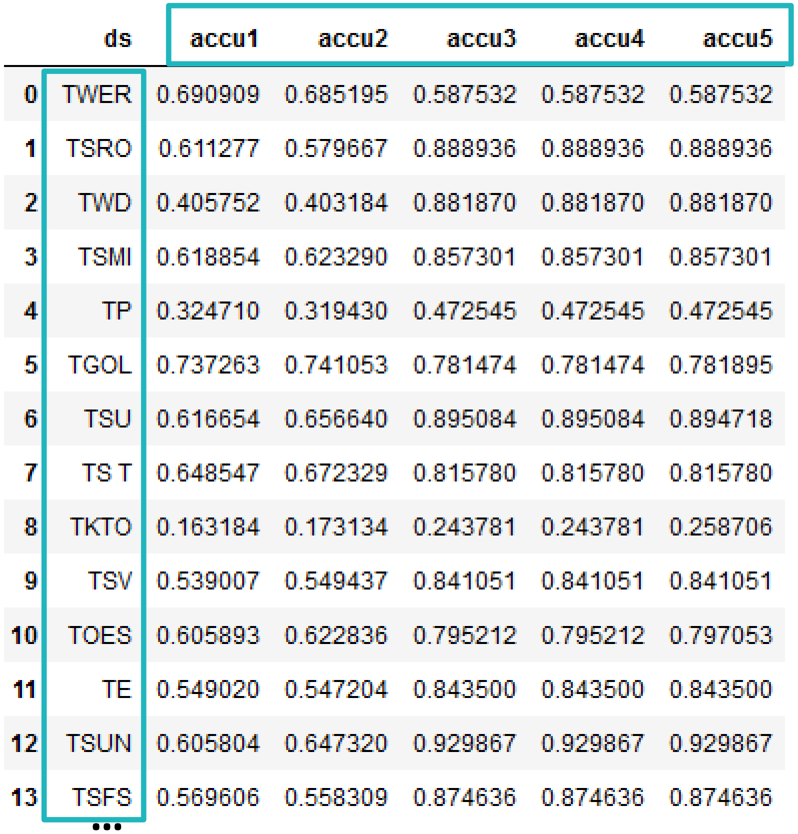
Visualisation of Classification Prediction

Goal: Visualise the classification results on a **map**.

- Additional plot packages:
  - **geopy**: Python client to locate coordinates using geocoders: https://geopy.readthedocs.io/en/stable/
  - **folium**: Library to visualise data on an interactive map (e.g., OpenStreetMap): https://python-visualization.github.io/folium/

Visualisation of Classification Prediction

- Read-in the **results of ML Classification** (prediction on the test dataset)…
- ... and convert this Spark DataFrame into a pandas DataFrame (**EX 1**)
- … which contains the **accuracy** of the predicted delay:
  - computed on all test samples of the **S1 test dataset**,
  - averaged **for each station** (ca. 30 accuracy values),
  - For the **five models** (sets of features),
  - … in order to have **five different maps**.



	ds	accu1	accu2	accu3	accu4	accu5
0	TWER	0.690909	0.685195	0.587532	0.587532	0.587532
1	TSRO	0.611277	0.579667	0.888936	0.888936	0.888936
2	TWD	0.405752	0.403184	0.881870	0.881870	0.881870
3	TSMI	0.618854	0.623290	0.857301	0.857301	0.857301
4	TP	0.324710	0.319430	0.472545	0.472545	0.472545
5	TGOL	0.737263	0.741053	0.781474	0.781474	0.781895
6	TSU	0.616654	0.656640	0.895084	0.895084	0.894718
7	TS T	0.648547	0.672329	0.815780	0.815780	0.815780
8	TKTO	0.163184	0.173134	0.243781	0.243781	0.258706
9	TSV	0.539007	0.549437	0.841051	0.841051	0.841051
10	TOES	0.605893	0.622836	0.795212	0.795212	0.797053
11	TE	0.549020	0.547204	0.843500	0.843500	0.843500
12	TSUN	0.605804	0.647320	0.929867	0.929867	0.929867
13	TSFS	0.569606	0.558309	0.874636	0.874636	0.874636

...

Visualisation of Classification Prediction

info not in the dataset

- Generate two lists of stations:
  - All **ordered** stations on the S1 line
  - A **non-ordered** list of S1 stations according to the accuracy DataFrame (**EX 2**)
- Produce a python **dictionary of stations**

  as *DS100 : station name*, e.g. *TB : Backnang*

- Define a **threshold** for the accuracy colour-code on the map:

  acceptable (>=0.8), borderline (0.5<= t < 0.8), poor (<0.5)

→ A colour is assigned to each station.

Visualisation of Classification Prediction

- The plot functions are defined in handlers.ipynb (class “PredictionVisualize”):
  - **getCoordsNoInt** and **getCoords**:
    - Read-in* (**EX 3**) or find with *geopy* (**EX 4**) the location of every station.
    - Associate a colour to each station based on the accuracy.
  - **drawMapDic**: Draw the map with *folium*.

optional exercise, since it cannot run

No general web access from cluster: This function **cannot** run.

* courtesy N. Güttler (Fraunhofer) and https://de.wikipedia.org/wiki/Liste_der_Stationen_der_S-Bahn_Stuttgart

Visualisation of Classification Prediction

5 sets of feature columns → 5 plots

1) Basic information of the original train dataset (5 features)
2) Basic information, weather data (11)
3) Delay at station -1 (derived features) (1)
4) Delay at stations -1, -2 (derived features) (2)
5) Basic information, delays and duration (derived features), weather data (8)

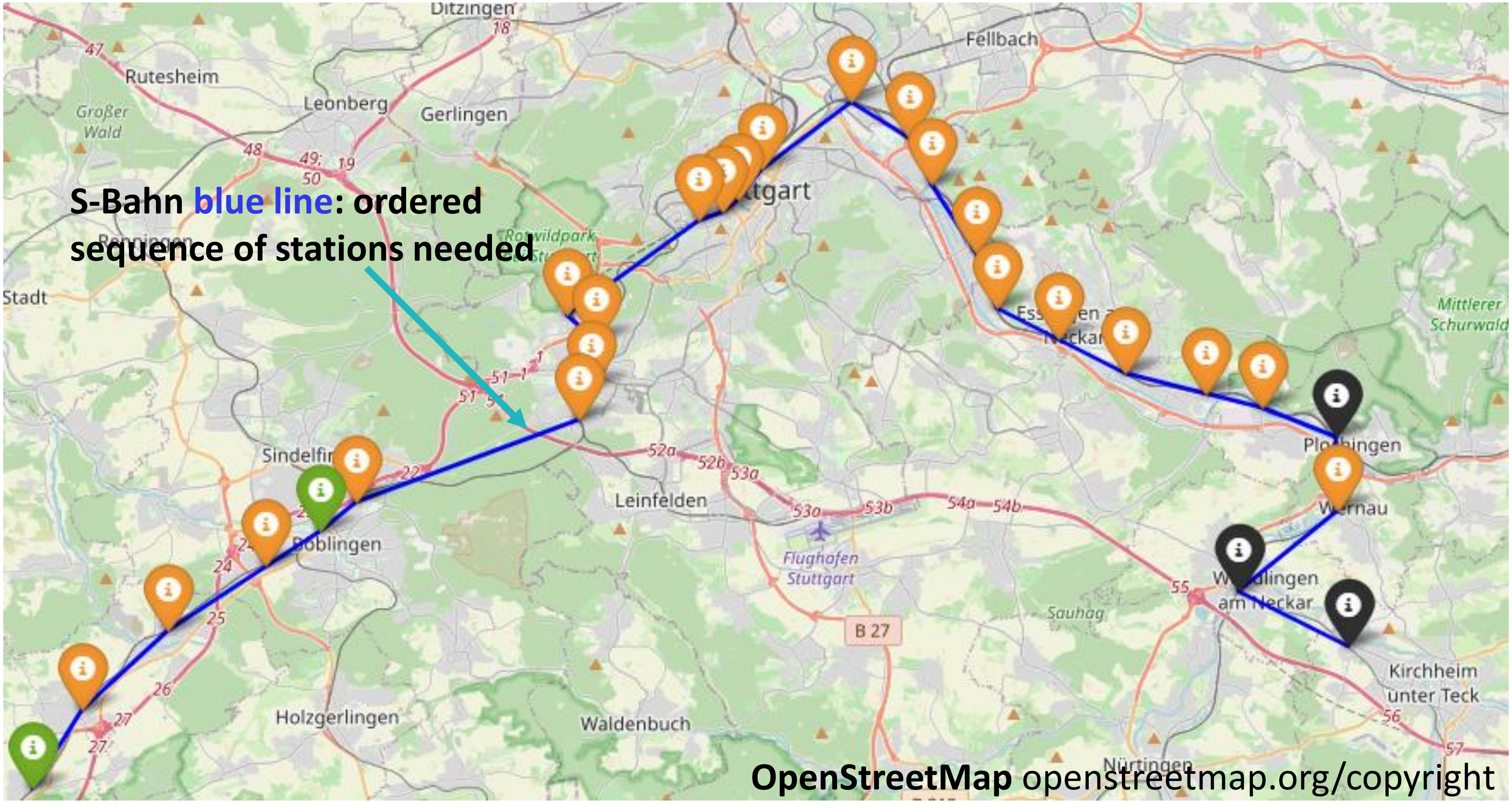
Visualisation of Classification Prediction

To open and see the plots:

next slide

- **Browse** to the folder `NB_plot/ClassificationPlot`
- **Download** the html files of the plots.
- Then, right-click on each plot file and “**Open with**” a browser (**not** the JN browser profile)!

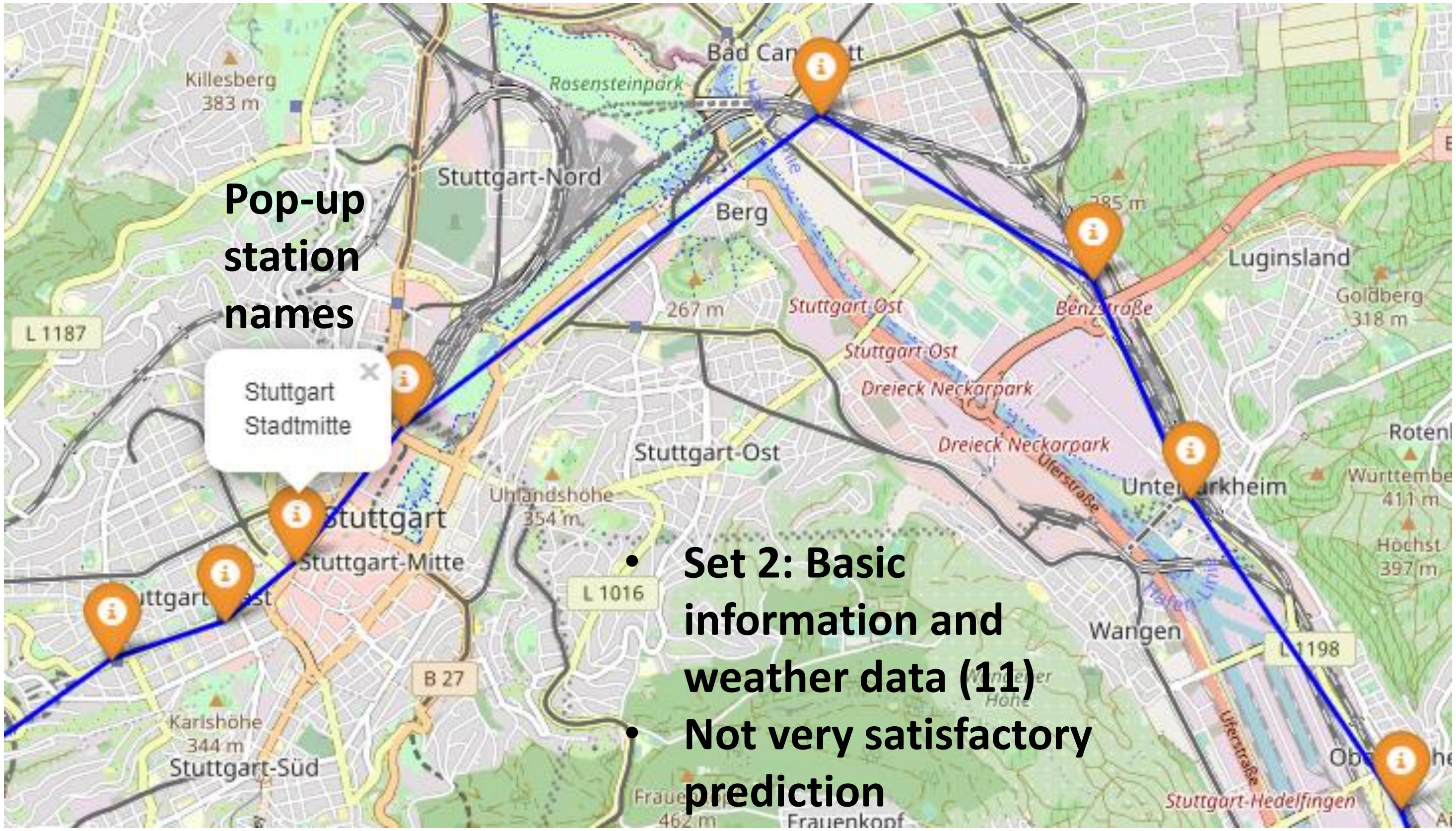
Visualisation of Classification Prediction: Feature set 2



S-Bahn blue line: ordered sequence of stations needed

OpenStreetMap openstreetmap.org/copyright

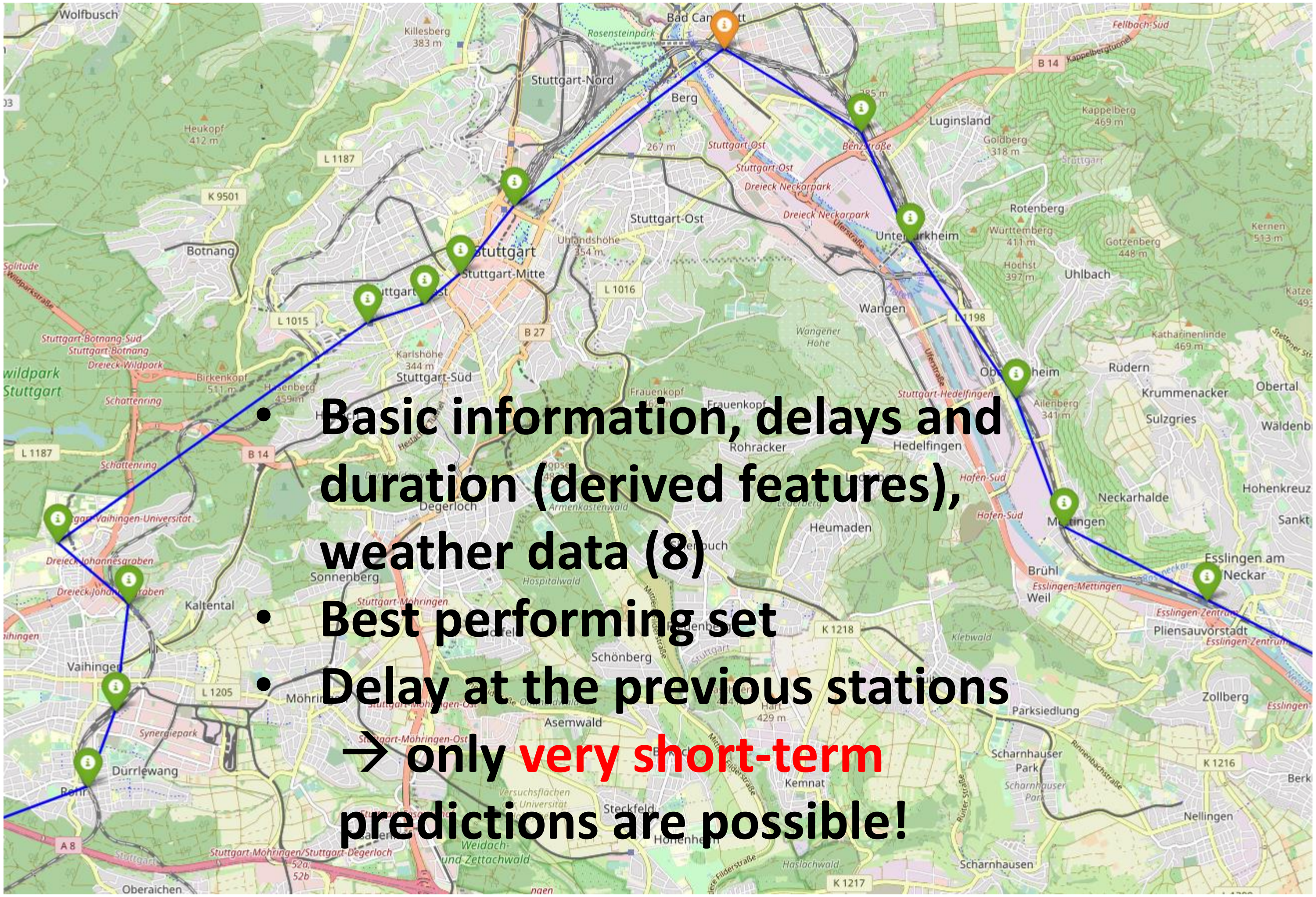
Visualisation of Classification Prediction: Feature set 2



Pop-up station names

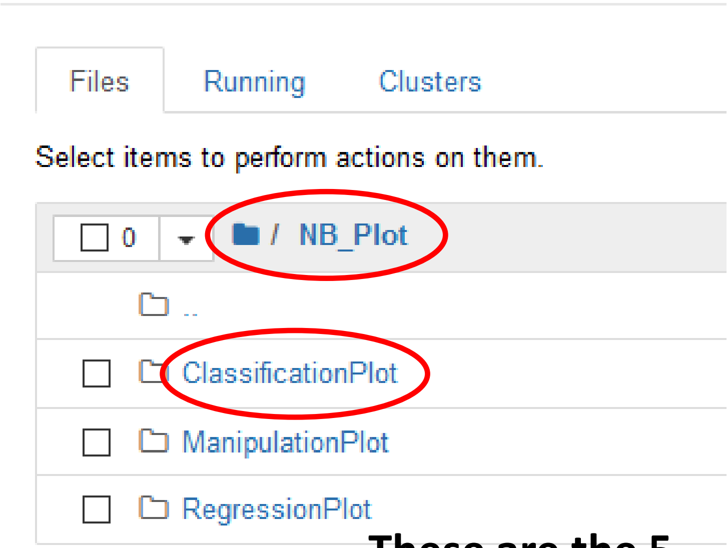
- **Set 2: Basic information and weather data (11)**
- **Not very satisfactory prediction**

Visualisation of Classification Prediction: Feature set 5



- Basic information, delays and duration (derived features), weather data (8)
- Best performing set
- Delay at the previous stations
  → only very short-term predictions are possible!

JN: AFTER the exercise



Files Running Clusters

Select items to perform actions on them.

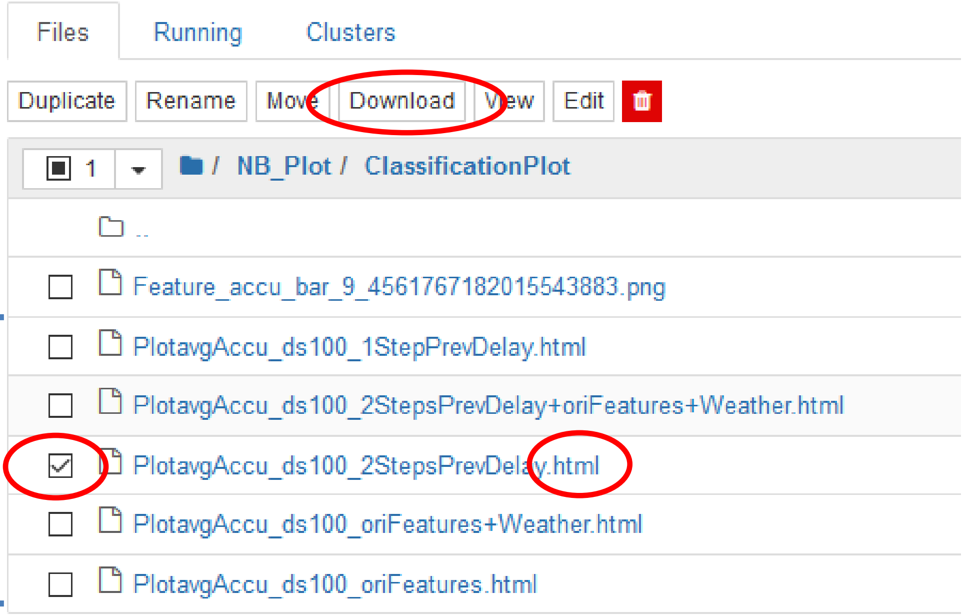
☐ 0 ▾ / NB_Plot

..

☐ ClassificationPlot

☐ ManipulationPlot

☐ RegressionPlot



Files Running Clusters

Duplicate Rename Move Download View Edit 🗑

■ 1 ▾ / NB_Plot / ClassificationPlot

..

☐ Feature_accu_bar_9_4561767182015543883.png

☐ PlotavgAccu_ds100_1StepPrevDelay.html

☐ PlotavgAccu_ds100_2StepsPrevDelay+oriFeatures+Weather.html

☑ PlotavgAccu_ds100_2StepsPrevDelay.html

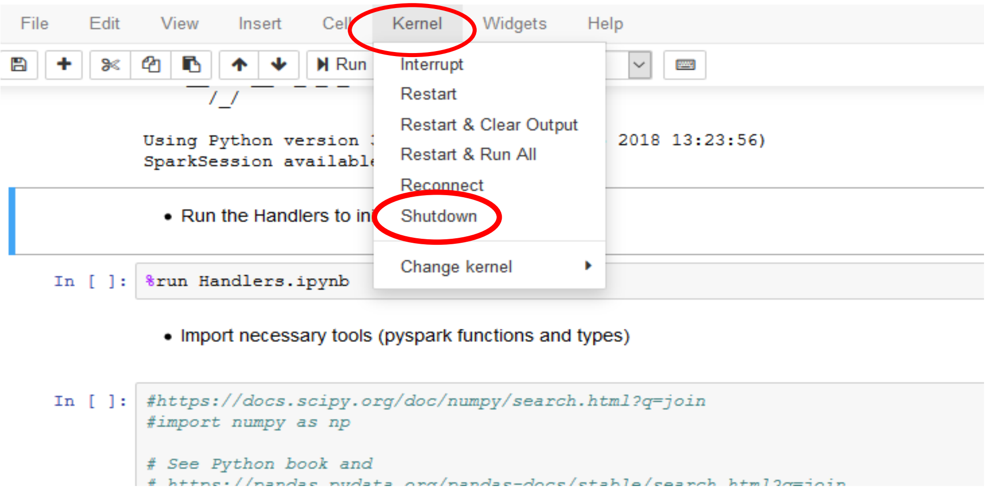
☐ PlotavgAccu_ds100_oriFeatures+Weather.html

☐ PlotavgAccu_ds100_oriFeatures.html

These are the 5 classification plots

Hands-on

- Execute all **NB5_vis-class.ipynb** (Notebook 5).
  - EX1-2 → "basic"
  - EX3-4 → "advanced"
- You can **download the plots** obtained as explained.
- Remember to shut down the kernel at the end.



```
  /_/

Using Python version                          2018 13:23:56)
SparkSession availabl
```

- Run the Handlers to in

```
In [ ]: %run Handlers.ipynb
```

- Import necessary tools (pyspark functions and types)

```
In [ ]: #https://docs.scipy.org/doc/numpy/search.html?q=join
        #import numpy as np

        # See Python book and
        # https://pandas.pydata.org/pandas-docs/stable/search.html?q=join
```

Focus on Pre-processing, Feature Engineering and Machine Learning

- Part III: HLRS Systems and Example as a Python script
  - Working on a cluster, parallel Spark
    - « HLRS Systems
    - ~~« Example on script (Ex.)~~
    - « Parallel Spark

More learning outcomes

Main index

Part I
Part II
Part III

Overview HLRS Systems

This example has been tested at HLRS on

- the **Vulcan NEC Cluster**, and
- the **Training Cluster** (TC).

Goal of this section:

- Overview of AI resources at HLRS,
- Get insights on using cluster resources,
- Get ready to run the example as a batch job.

Overview HLRS Systems

OLD SYSTEM!!!

HLRS **CPUs** for DataScience: Urika-GX:
https://kb.hlrs.de/platforms/index.php/Urika_GX

Two systems:

- **Gilgamesch**

Used by several HLRS partners in different projects.

48 Nodes in total (2 Login, 2 IO, 3 Service, **41 Compute Nodes**)

- **Enkidu**

Used for test and training.

16 Nodes in total (2 Login, 2 IO, 3 Service, **9 Compute Nodes**)

Jupyter Notebook with Spark kernel **and HDFS**

[oleksandr.shcherbakov@hlrs.de]

Overview HLRS Systems (for AI)

Big Data and AI at HLRS (Nov 2022 + update 2024):

Cray CS Storm and Cray CS500 [2] with ~~**Cray Urika CS software stack** [3]~~ on Vulcan / TC (nodes `clx-21` and `clx-ai`)

Hawk [4] and **Vulcan** [5] **clusters**

Hawk AI expansion [6] (nodes `rome-ai`)

If you need any support, feel free to ask:
rt-ai@hlrs.de - CS Storm/500 related issues;
rt@hlrs.de - any other topics;
or use our ticket submission form [7].

((1. https://kb.hlrs.de/platforms/index.php/Urika_GX)) → system shut down Feb 1, 2021
2. https://www.hlrs.de/solutions/systems/cray-cs-storm
3. ~~https://kb.hlrs.de/platforms/index.php/Urika_CS~~
4. https://www.hlrs.de/solutions/systems/hpe-apollo-hawk
5. https://www.hlrs.de/solutions/systems/vulcan/
6. https://www.hlrs.de/news/detail/hawk-upgrade-artificial-intelligence
7. https://www.hlrs.de/for-users/trouble-ticket-submission

See also:
https://kb.hlrs.de/platforms/index.php/Big_Data,_AI_Aplications_and_Frameworks

Overview HLRS Systems (for AI)

- The CS partition in detail:
  - CS-500 [clx-21] = 8 nodes (Vu) 2 x 20 core-CPU per node and **384gb** memory and local scratch
  - CS-Storm [clx-ai] = 4 (Vu) + 4 (TC) **nodes** 2 x 18 core-CPU + **8 GPUs** per node and **768gb** memory and local scratch

https://www.hlrs.de/training/2024/CUDA

coming soon

- Additionally in the CS partition :
  - Singularity container (currently not available: https://websrv.hlrs.de/cgi-bin/hwwweather)
  - … to start the (outdated) Urika-CS **container** (Cray) https://kb.hlrs.de/platforms/index.php/Urika_CS

Overview HLRS Systems

What is needed **for the eaxmple**:

1. Store and use source data.
2. Apply specific software.
3. Running a job either as a batch job (e.g. for a script) or as an interactive batch job (e.g. for Jupyter Notebook):
   - ***Frontend nodes:*** *are intended as single point of access to the entire cluster. Here you can set your environment, move your data, edit and compile your programs and* ***create batch scripts****. [Direct] interactive usage* ***like run your program*** *which leads to a high load is* ***NOT allowed*** *on the frontend/login nodes.*
   - ***Compute nodes*** *for running parallel jobs are only available through the* ***batch system.***

https://kb.hlrs.de/platforms/index.php/NEC_Cluster_access_(vulcan)

Vulcan / Training Cluster **DATA**

- Source data are stored in the **Lustre** filesystem NEC_lustre:
  https://kb.hlrs.de/platforms/index.php/NEC_Cluster_Disk_Storage_(vulcan)
- **On Vulcan**: **Lustre** accessible only via **workspaces**:
  https://kb.hlrs.de/platforms/index.php/Workspace_mechanism

  On TC: Lustre not available: NFS with workspaces.

Details skipped

Vulcan / Training Cluster **DATA**

- Workspaces…
  - allocate disk space for your jobs
  - have an identifier (a name)
- A workspace can be generated with **ws_allocate** and its path stored to an environmental variable:
  MYSCR=$(ws_allocate *workspaceFavouriteName #days*)
  echo $MYSCR
  (the workspace path is a **Lustre** path!)
- Workspaces **expire**! Can be extended, retrieved from trash, reminders can be sent automatically.

Vulcan only: DATA

- The tool **ws_exchange** allows for the flexible exchange of data among users **within their workspaces**: https://kb.hlrs.de/platforms/index.php/CAE_utilities#ws_exchange_procedure
  It creates by default:
  - a new **temporary** workspace with protected content
  - a **subdirectory** with random name (but public rwx permission).
- This is what we are going to use to exchange data.
- **ws_cp2exchange** is a special command which enables copying your data directly in the exchange subdirectory.

Details skipped

Vulcan only: DATA

PRACTICAL (optional)

```
> module  load  cae
```

- Create a sample file:

```
> cat  >  sample.txt
```

Input some text and type *ctrl+D* to quit

- Create the private exchange directory and public subdirectory (**id** is displayed as **exchange2020…**):

```
> ws_exchange
```

- Move sample.txt to exchange (replacing ***id*** with the corresponding output of ws_exchange):

```
> ws_cp2exchange  mv  sample.txt  id
```

Vulcan / Training Cluster **MODULES**

The **module** system:

https://kb.hlrs.de/platforms/index.php/NEC_Cluster_Software_Environment_(vulcan)

- Modules can be loaded / unloaded.
- The environmental setting (= loaded packages) will **not** be saved and will be **lost** for a new session:
  … A new session (login, new submitted job, **compute node**) will have the default environment.
- Modules support **multiple versions** of a software.

Vulcan / Training Cluster **MODULES**

- Display the modules available in the system:

```
> module avail
```

- Modules already loaded in your environment:

```
> module list
```

- Modules needed for the example are loaded through the `init_….sh` scripts.

- How to install python packages that need to be downloaded / are **not available** on the system?

Vulcan / Training Cluster **MODULES**

- General **Internet** is not available in the clusters!
- Instead, use an `ssh` tunnel to a local machine:

https://kb.hlrs.de/platforms/index.php/Secure_Shell_ssh

- … for `pip install`, see in particular:

https://kb.hlrs.de/platforms/index.php/Secure_Shell_ssh#pip_.28Python_package_installer.29

- The additional packages will be **locally** available on the python **module** used for the `pip install` (e.g. python/3.6).

Vulcan / Training Cluster **Compute nodes**

- **Compute nodes** have 4 main characteristics:
  - *node_type*: node ID
  - *node_type_cpu*: CPU name
  - *node_type_mem*: memory on this node
  - *node_type_core*: number of cores on this node
- **Vulcan**: Allocated nodes will not be shared **with other jobs**!
  vs.
  Training Cluster: Node-sharing is possible (`-q smp`).
- How to monitor jobs running on the compute nodes:

https://kb.hlrs.de/platforms/index.php/Batch_System_PBSPro_(vulcan)

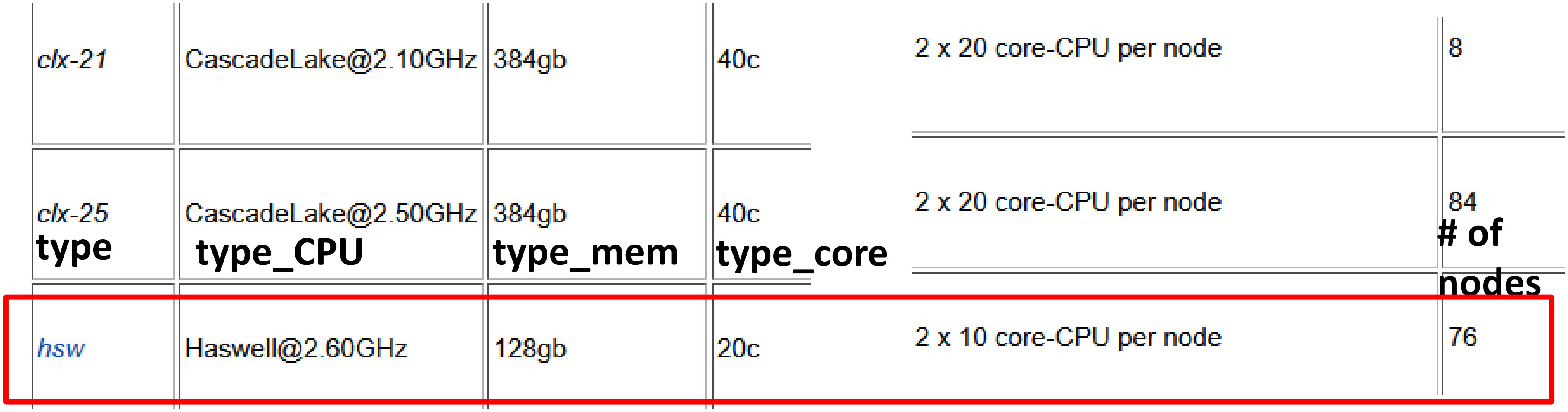
Vulcan / Training Cluster **Compute nodes**

- At least three features must be specified:
  - **Number** of nodes
  - **At least one** node variable (of the four above)
  - **Walltime**
- This can be done as an **interactive batch job** or
- …in a **job script** for submission:

```
#!/bin/bash
#PBS -N LZ_sbahn
#PBS -l select=4:node_type=hsw:node_type_mem=128gb:node_type_core=24c
#PBS -l walltime=00:20:00
…
```

Vulcan / Training Cluster **Compute nodes**

An overview of the available nodes is provided:



type	type_CPU	type_mem	type_core		# of nodes
clx-21	CascadeLake@2.10GHz	384gb	40c	2 x 20 core-CPU per node	8
clx-25	CascadeLake@2.50GHz	384gb	40c	2 x 20 core-CPU per node	84
hsw	Haswell@2.60GHz	128gb	20c	2 x 10 core-CPU per node	76

Current table at:
https://kb.hlrs.de/platforms/index.php/Batch_System_PBSPro_(vulcan)

E.g. selecting **4 nodes** of type hsw, one would have:

- 4 X 20 cores = **80** total core-CPUs
- Executor memory up to **128** GB per node

Vulcan / Training Cluster **Compute nodes**

- The job characteristics should at least match the resources required by the Spark session!
- E.g., our **1** (or 4) **hsw Vulcan nodes** would allow at the most:

```
spark-submit
--name SBahn_script
--executor-memory 128g (128g)
--total-executor-cores 20 (80)
```

Memory/executor = Memory/node (default).

Total number of cores used in the cluster.

Focus on Pre-processing, Feature Engineering and Machine Learning

- Part III: HLRS Systems and Example as a Python script
  - Work on a cluster, parallel Spark
    - « HLRS Systems
    - ~~« Example on script (Ex.)~~
    - « Parallel Spark

~~Work in progress~~

More learning outcomes

Main index

Part I
Part II
Part III

Spark memory management

Spark is a **parallel** application:

https://spark.apache.org/docs/latest/cluster-overview.html

In short, it can run on:

- 1 node: Spark can run **locally** on one compute node with as many worker threads as cores in the node
- Multiple nodes: A **cluster manager** (*master*) is needed, for example:
  - **Standalone** cluster (on Vulcan/TC)

  or third-party managers, such as:
  - Hadoop YARN (on Urika-GX)

Details skipped

Spark memory management

Moreover:

- Options that can be set:
  needed resources (memory, executors).
- The *master* option can be managed through a configuration file. See for an overview of all options:
  https://spark.apache.org/docs/latest/submitting-applications.html#master-urls

Spark memory management

- Data parallelism: Spark revolves around the concept of a **resilient distributed dataset (RDD)**:
  - *"A collection of elements partitioned across the nodes of the cluster that can be operated on in parallel".*
  - In practice, **DataFrames** and Datasets **extend** this abstraction.
- There are two ways to create RDDs:
  - **parallelising an existing collection** in your driver program,
  - referencing a dataset in an external storage system, such as a shared filesystem → **HDFS**

https://spark.apache.org/docs/latest/rdd-programming-guide.html

Spark memory management

Parallel operations on a cluster: *driver-worker* parallelism

- Driver node: Executes the user's **main function** and distributes work to executors;
- Executors on a worker node:
  - *Lazily* execute tasks (local operations on **partitions** of the RDD).
  - Rely on **local disk (when available)** for:
    - storing *shuffle* data (= data exchanged at the end of a *stage*),
    - *spilling* data that are too large.

next slide

Courtesy Cray Urika-XC Training materials

Spark memory management

Spark operations inside a node:

- Transformation APIs, producing a new RDD
- Action APIs, returning some data

… are pipelined into **tasks**.

Details skipped

Spark **stage**: Tasks are executed **on all** RDD partitions (executors), ending with:

- A shuffle, i.e. an **all-to-all communication** (or an output, or data sent back to the driver).
- Then, a global **barrier,** i.e. a synchronisation before the next stage.

Courtesy Cray Urika-XC Training materials

Spark memory management

The communication (*shuffle*) is coordinated **within each node** by a **Block Manager**:

- Locally writes the data needed for reduction.
- Sends the requested data to the receiver.

Assumption for efficiency:

Large, fast local block **storage devices** on the executor nodes!
“AI and HPDA workflows can require local storage.
However, HPC nodes usually do not have any local drive except for particular nodes.”
https://kb.hlrs.de/platforms/index.php/Big_Data,_AI_Aplications_and_Frameworks

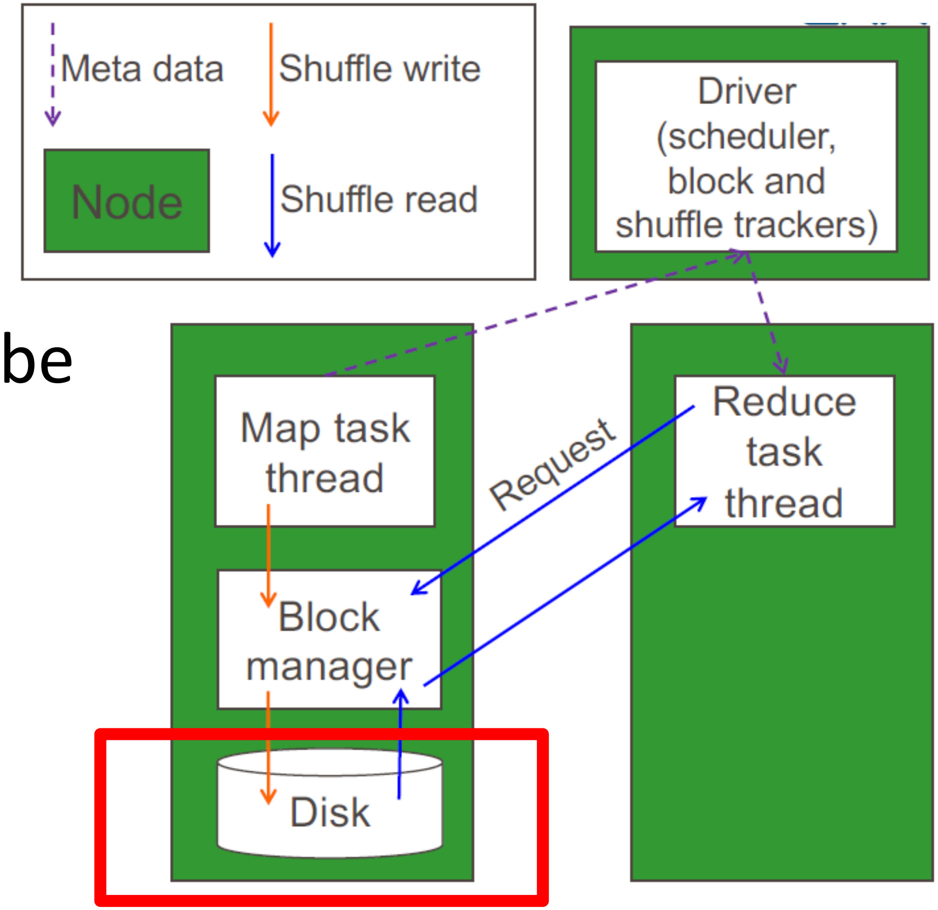
Spark memory management

Spark communication model: Shuffle

Local storage on AI nodes (`clx-ai`, `clx-21`) must be configured for Spark!

See the procedure:

`https://kb.hlrs.de/platforms/index.php/Big_Data,_AI_Aplications_and_Frameworks#Spark`



Courtesy Cray Urika-XC Training materials

Example: Spark memory management (Vulcan 2021)

```
$ grep SPARK_WORKER_DIR /opt/bigdata/spark_cluster/spark-2.4.6-bin-
hadoop2.7/bin/init-spark
SPARK_WORKER_DIR="/tmp/${USER}_spark"
```

/tmp is the storage for shuffle/spilling in the current configuration!

```
clx-ai $ df -h /tmp /localscratch

Filesystem                                           Size   Used Avail Use% Mounted
on
/dev/sda                                             220G    61M  209G   1% /tmp
/dev/md0                                             7.3T    93M  6.9T   1% /localscratch
```

OK

```
clx-21 $ df -h /tmp /localscratch

Filesystem                                           Size   Used Avail Use% Mounted
on

none                                                 512M    49M  464M  10% /var/tmp
/dev/nvme0n1                                         1.8T    77M  1.7T   1% /localscratch
```

Limited!

- Here, Spark is not using the local scratch for shuffling/spilling!
- In most nodes, `/tmp` is RAM-disk and quite small.
- Total RAM given by `free -h` : Look for `available` in the output.

Details skipped

Spark memory management

A job is split among executors:

The quick red | fox looks at | the red robin.

Each task is executed + **writing** of shuffle data:

```
the(1) quick(1) red (1)          the(1) red(1) robin(1)

               fox(1) looks(1) at(1)
```

Communication among executors:

```
quick(1) red(1+1)                the(1+1) robin(1)

               fox(1) looks(1) at(1)
```

Execution again (count):

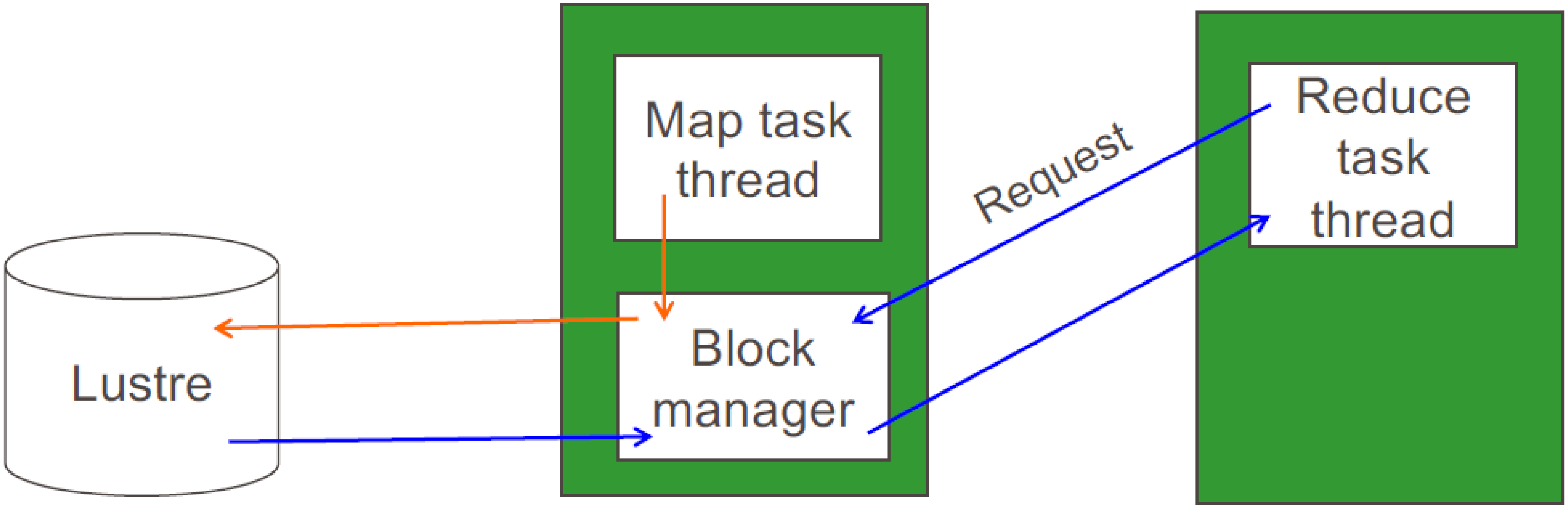
```
quick(1) red(2)                  the(2) robin(1)

               fox(1) looks(1) at(1)
```

Spark memory management

Data traffic through Lustre instead of local storage

→ Would create a major bottleneck!

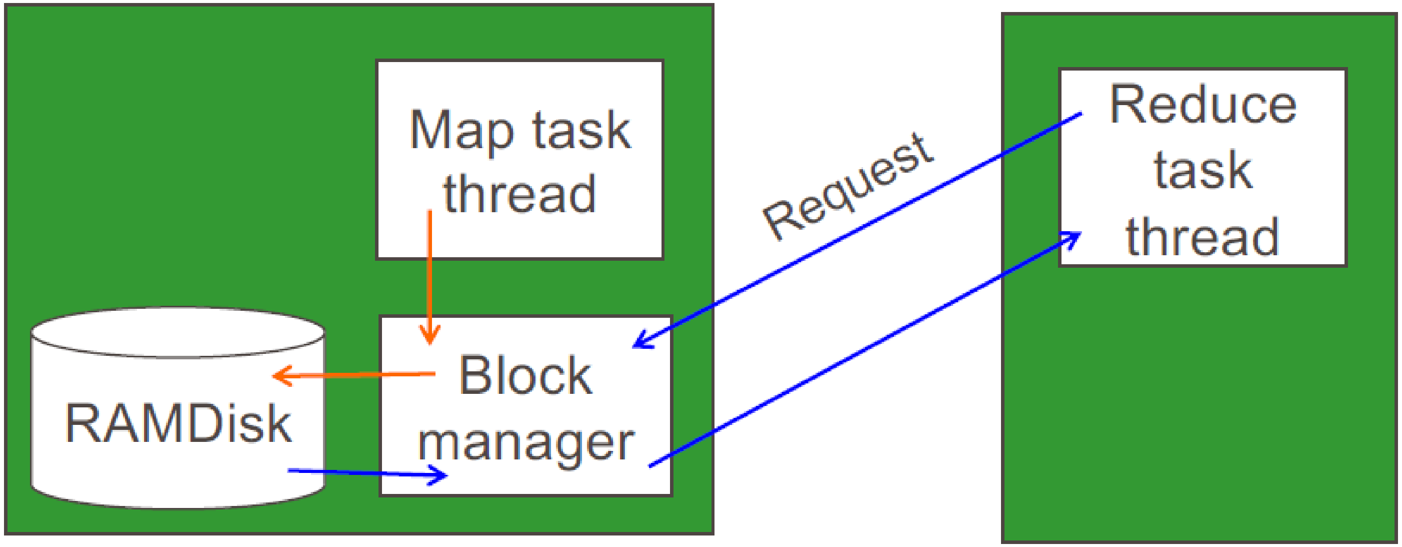
→ Could be configured.



Courtesy Cray Urika-XC Training materials

Spark memory management

Provided as RAMDisk (**all non-AI nodes**)



- This space could be small,
- … **taking away memory that could otherwise be allocated to Spark execution**.
- **Communication will be slow**, no scaling.

Courtesy Cray Urika-XC Training materials

Spark memory management

More on Spark memory management…

1. https://0x0fff.com/spark-memory-management/

2. https://stackoverflow.com/questions/30797724/how-to-optimize-shuffle-spill-in-apache-spark-application

Summary

What we have done:

- We went through **Machine Learning workflow** from data manipulation to visualisation,
- … using the ML framework **Spark**.
- The same example was executed on a Linux cluster within an **interactive batch job**:
  - on a **Jupyter Notebook**,
  - ~~as a **script**~~

Summary

We have gone through:

- Machine and Deep Learning basic **concepts**
  - features, overfitting, learning curve, hyperparameters, …
- Some Machine Learning **methods and algorithms**
  - linear regression, decision trees, random forest.

https://developers.google.com/machine-learning/glossary

Outlook

- Use Spark, Pandas, other ML frameworks for your own applications.
- **Parallelisation** and performance depend on the **architecture available**, the **software**, the **expertise**.
- Neural Networks / Deep Learning → Day 2
- CFD Applications / Data compression / Reproducibility? → Day 3

[Sayash Kapoor, Arvind Narayanan: Leakage and the reproducibility crisis in machine learning-based science, 2023, Patterns 4.]

Thank you!

https://www.hlrs.de/training

https://www.hlrs.de/training/2024/SCA-DA-MGNT

https://www.hlrs.de/training/2024/IKILEUS-NLP

https://www.hlrs.de/training/2024/BC-AI-NV