time series feature extraction with tsfresh

GET RICH OR DIE OVERFITTING

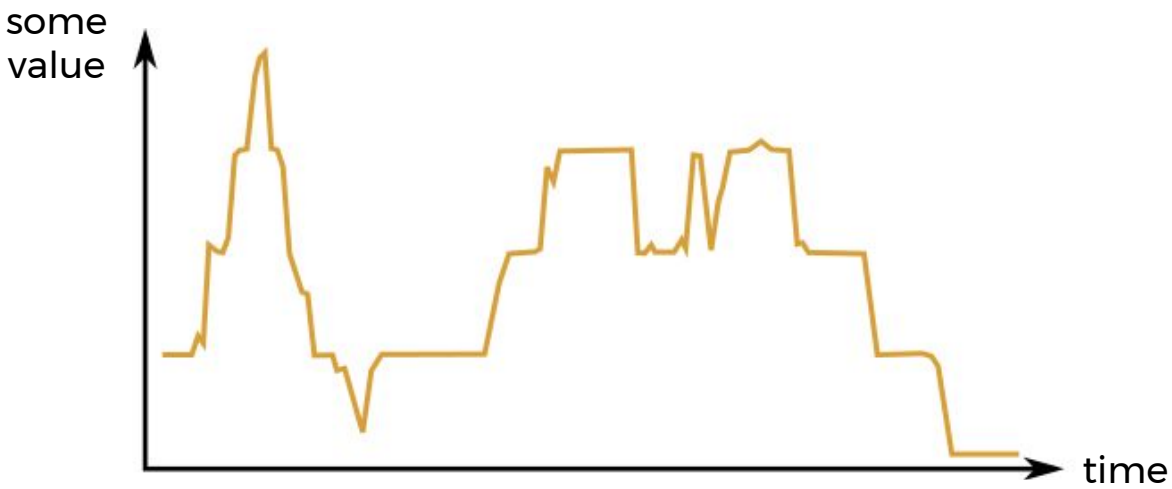
@_nilsbraun

you will not learn how to get rich during this talk

1.

MOTIVATION

What you will
learn instead


some
value
time

IoT

social networks

home automation

medicine

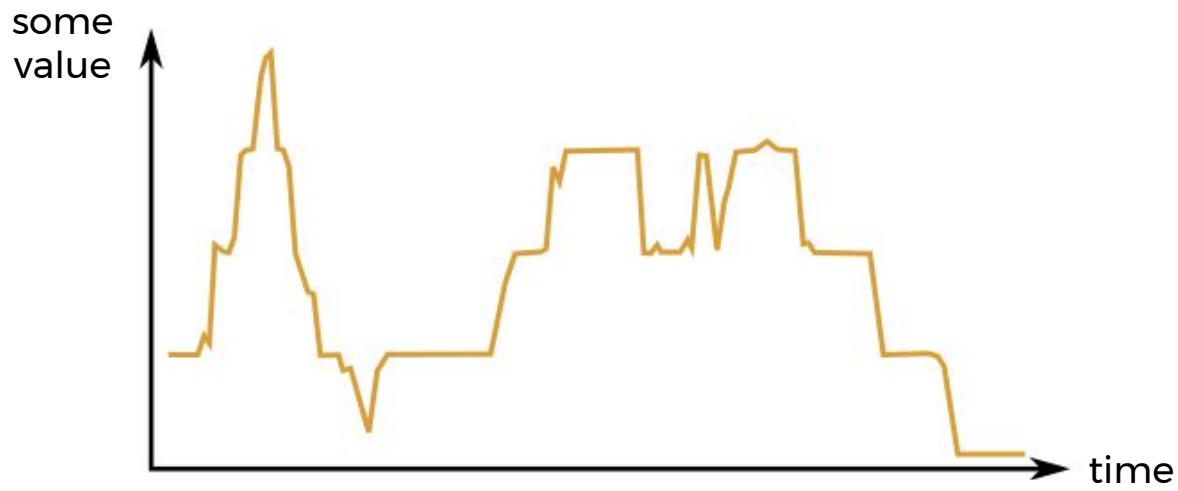
physics

server logs

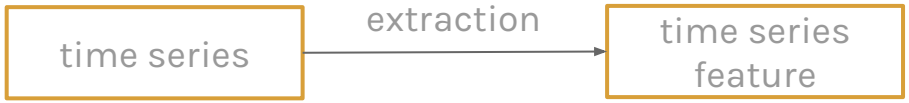
Industrie 4.0

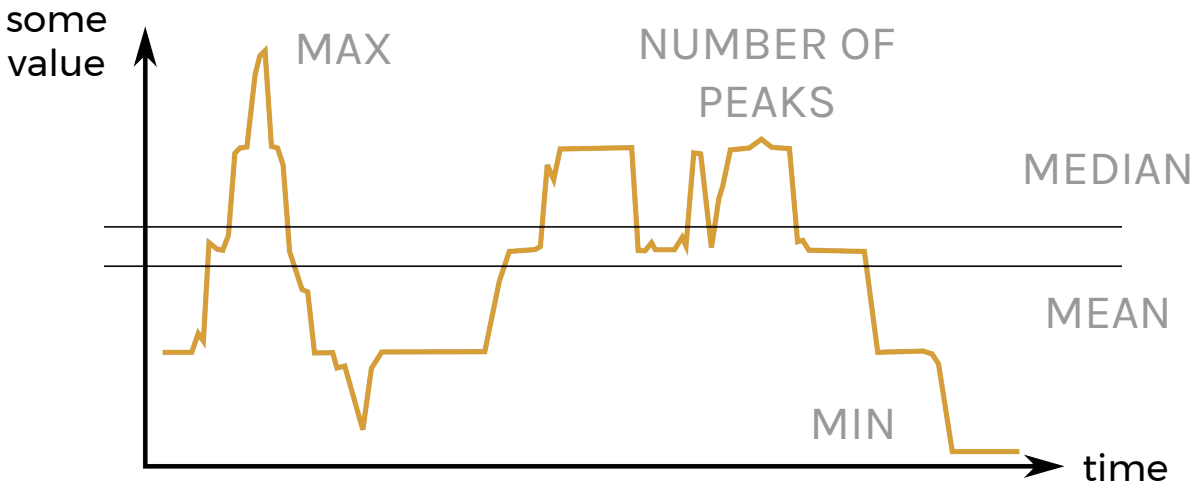
finance data

connected cars



how to apply machine learning?



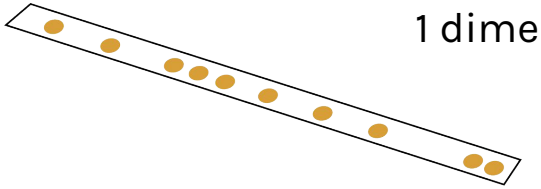


map the structured input
into a lower dimensional space

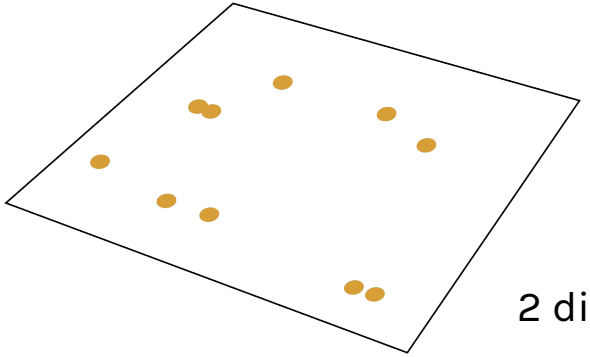
LET'S AUTOMATE FEATURE EXTRACTION!

- automated data analytics plays a crucial role in many future applications of data science
  - live data analysis & streamed data
  - efficient reduction of sample size
  - "data analysis as a service"
- faster and easier feature engineering
- otherwise: knowledge of signal processing and feature extraction in multifarious domains needed

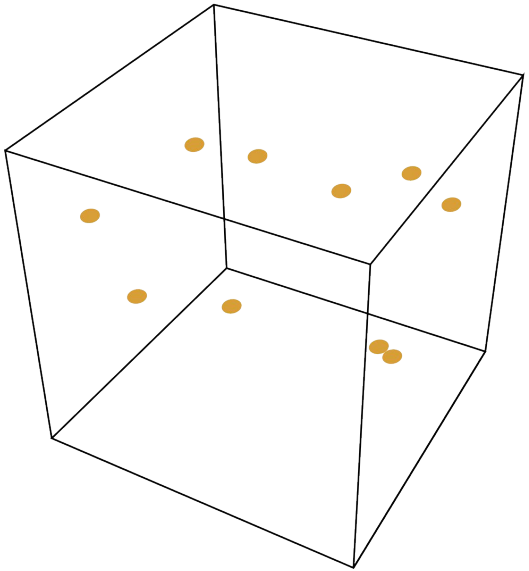
WHY FEATURE SELECTION?



1 dimension



2 dimensions



3 dimensions

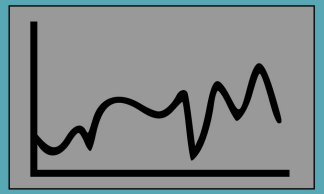
2.

INTRODUCING TSFRESH

Time **S**eries
Featu**R**e
Extraction based
on
Scalable
Hypothesis test

WORKFLOW OF TSFRESH

RAW TIME SERIES



FEATURE EXTRACTION

global maximum	augmented dickey fuller
stddev	cwt coefficients
quantile	fft
autocorrelation	...

P VALUE CALCULATION

p value calculation based on statistical hypothesis tests

Benjamini-Yekutieli procedure respecting a global false discovery rate

FEATURE SELECTION

quantile	autocorrelation
cwt coefficients	...

- feature extraction and selection in parallel
- feature extractors use external binary libraries
- framework optimized for speed

- more than 60 feature extractors with parameters
- selection ready for regression and classification
- utilities for data preprocessing

IT'S **FAST** - IT'S **LARGE** - IT'S TS**FRESH**

- easy to use
- interface to scikit learn

3.

USING TSFRESH

How can you use tsfresh?

FEATURE
EXTRACTION

In [1]:
```
from tsfresh import extract_features
```

In [2]:
```
df.head()
```

Out [2]:

	id	time	temp	pres
0	dev1	0.1	3.14	42
1	dev1	0.2	7.41	47

In [3]:
```
X = extract_features(df,
                     column_id='id',
                     column_sort='time')
```

FEATURE
SELECTION

In [1]:
```
from tsfresh import select_features
```

In [2]:
```
X.head()
```

Out [2]:

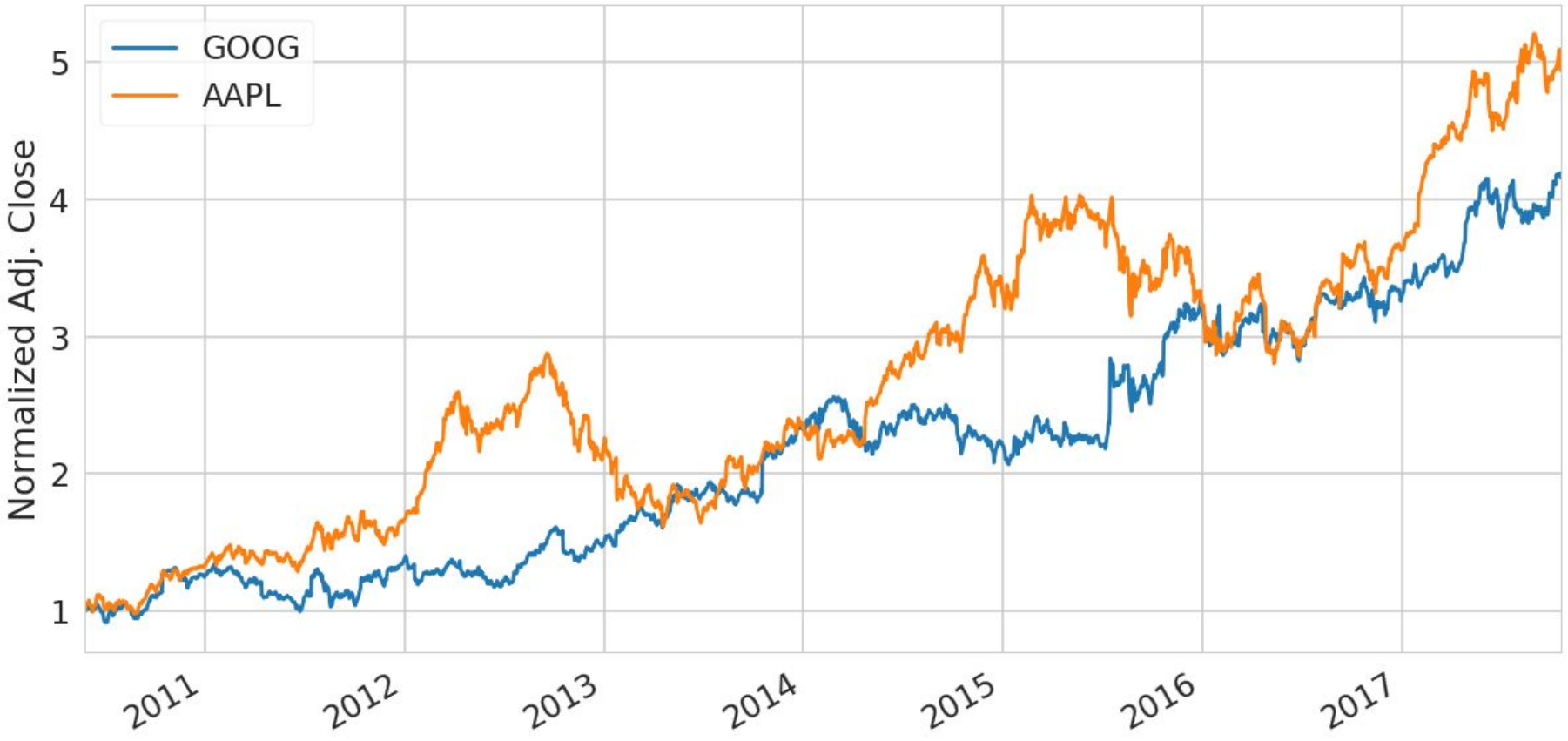
id	temp__mean	temp__max	pres__mean	pres__max
dev1	3.24	4.2	47	66
dev2	4.67	11.34	44	60

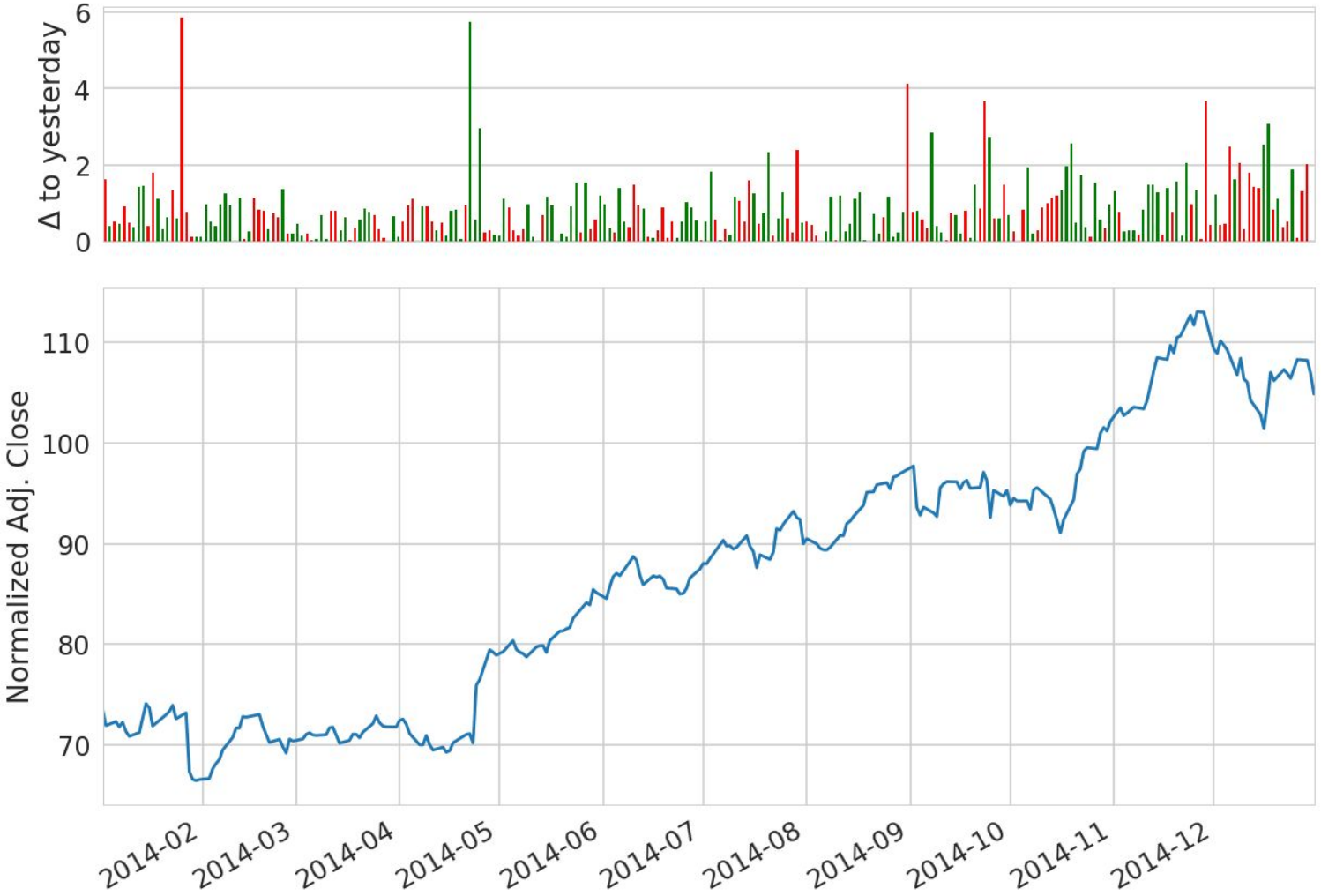
In [3]:
```
sel_X = select_features(X, y,
                        fdr_level=0.05)
```

4.

TSFRESH AND FINANCE DATA

How to (still) get rich with tsfresh

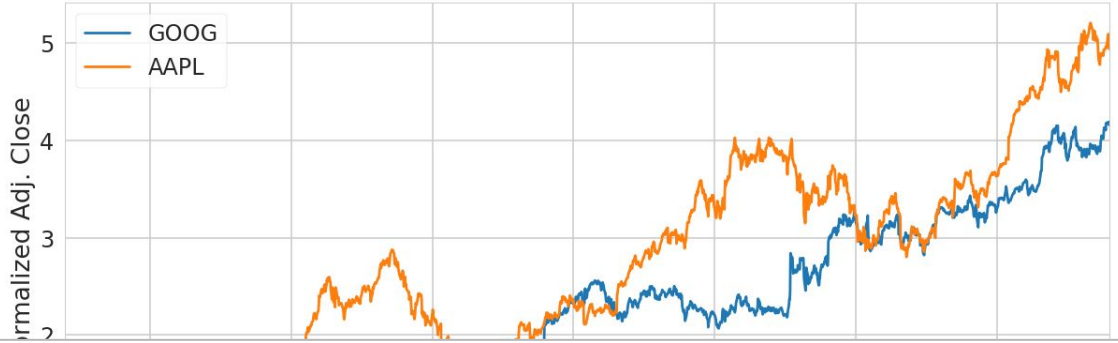

GOOG
AAPL
Normalized Adj. Close
1
2
3
4
5
2011
2012
2013
2014
2015
2016
2017


Δ to yesterday
6
4
2
0
Normalized Adj. Close
110
100
90
80
70
2014-02
2014-03
2014-04
2014-05
2014-06
2014-07
2014-08
2014-09
2014-10
2014-11
2014-12

In [1]:
```
from pandas_datareader import data
```

In [2]:
```
symbols = ["AAPL", "GOOG", "MSFT", "FB", "TSM", "ORCL",
           "INTC", "IBM", "SAP", "NVDA", "AVGO", "TXN",
           "DCM", "QCOM", "ADBE", "BIDU", "CRM",
           "AABA", "AMAT"]

stock_data = {symbol: data.get_data_yahoo(symbol) for symbol in symbols}
```



In [1]:
```
from pandas as pd
```

In [2]:
```
melted = pd.melt(df)
melted.rename(columns={"variable": "kind", "value": "price"},
              inplace=True)
melted["time"] = list(df.index)*len(df.columns)
melted["id"] = 1
```

In [3]:
```
from tsfresh.utilities.dataframe_functions import roll_time_series
use_data_of = 100

rolled = roll_time_series(melted.copy(), column_id="id", column_sort="time",
                          column_kind="kind",
                          rolling_direction=1,
                          max_timeshift=use_data_of - 1).reset_index(drop=True)
rolled = rolled.groupby("id")\
               .filter(lambda x: len(set(x["time"])) == use_data_of)\
               .reset_index(drop=True)
```

In [1]:
```
from tsfresh import extract_features
from tsfresh.feature_extraction.settings import EfficientFCParameters
```

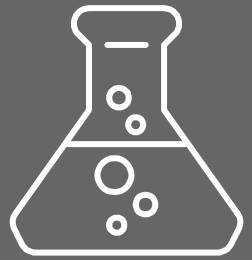
In [2]:
```
X = extract_features(rolled, column_id="id", column_sort="time",
                     column_kind="kind", column_value="price",
                     default_fc_parameters=EfficientFCParameters(), n_jobs=6)
```

variable	abs_energy	absolute_sum_of_changes	agg_autocorrelation__f_agg_"mean"	...	value_count__value_nan	variance	symbol
2010-05-25 00:00:00AABA	26766.161079	24.269979	-0.015184	...	0.0	0.764041	AABA
2010-05-26 00:00:00AABA	26723.295145	24.089980	-0.019028	...	0.0	0.769769	AABA
2010-05-27 00:00:00AABA	26677.061245	24.199980	-0.016888	...	0.0	0.767560	AABA
2010-05-28 00:00:00AABA	26615.503945	24.489980	-0.015479	...	0.0	0.768132	AABA
2010-06-01 00:00:00AABA	26546.295445	24.339981	-0.016363	...	0.0	0.776085	AABA


tsfresh
S&P 500
return
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1.0
0.9
2013-07
2014-01
2014-07
2015-01
2015-07
2016-01
2016-07

5.

SOME DESIGN DECISIONS

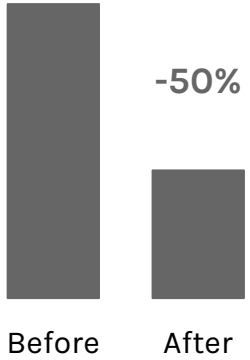


What we have learned during the development

PANDAS IS GREAT, BUT **SLOW**

- especially the creation of `Series` and `Dataframes` should not happen often
  - for example: in the `groupby.apply` function: never return a `Series`! Go with a list of `tuples` instead
- we all know, but sometimes forget: do not append to `Dataframes`

SPEEDING UP TSFRESH



- start with a good profiling of the software (as always!), e.g. using `line_profiler`
- reduce the amount of pandas objects
  - especially when using random access, masking, basic arithmetics
  - try not to copy them around or recreate them every time
- always have a consistent speed test ready

HOW TO HANDLE **SETTINGS**?

- our top level functions (`extract_features`, `select_features`) have a large amount of parameters
- especially the information, which feature should be extracted, must be easily accessible by the user

A SETTINGS OBJECT, A **GOOD** IDEA!?

```
from tsfresh import Settings
settings = Settings()
extract_features(data,
                 settings)
```

- many questions on gitter/issues on github about this
- object and function development needed to be synced
- we started to invent our own dictionary class

HAVING TOO **MANY** TIME SERIES?

NO PROBLEM.

6.

BRINGING TSFRESH TO THE CLOUD

https://what-if.xkcd.com/13/

USING TSFRESH WITH **DASK**

```
from tsfresh import extract_features

extract_features(df)
```

```
from tsfresh.utilities.distribution import ClusterDaskDistributor
from tsfresh import extract_features

distributor = ClusterDaskDistributor("192.168.178.111:8000")
extract_features(df, distributor=distributor)
```

HOW FAR ARE WE?

- dask (distributed) support built into tsfresh
  - can run on Amazon EC2
- first tested test instance running on AWS Lambda
  - with the help of `zappa` and `flask`
- more information:
  - https://content.pivotal.io/blog/automated-machine-learning-deploying-automl-to-the-cloud
  - https://nils-braun.github.io/amazon-lambda-and-map-2/

TSFRESH ON ONE SLIDE

INHOMOGENEOUS SOURCES

several time series and meta-information simultaneously

DECENTRAL PROCESSING

process close to source, faster feedback (no latency), no memory problems

BIG DATA

scale with number of time series, number of devices and length of time series

ROBUSTNESS

labeled samples are expensive, overfitting bad

FIELD TESTED

tsfresh was developed for a consulting project on Industrie 4.0, more than 2k stars on github

Christ, Kempa-Liehr, Feindt: Workshop on Learning on Big Data, ACML 2016

ADDITIONAL RESOURCES

tsfresh.readthedocs.io/en/latest/

github.com/blue-yonder/tsfresh/

@_nilsbraun

nils-braun.github.io

Michael Feindt, Max Christ,
Julius Neuffer, Andreas Kempa-Liehr


tsfresh is powered with Python

THANK YOU!