Mastering Machine Learning Techniques for Time Series

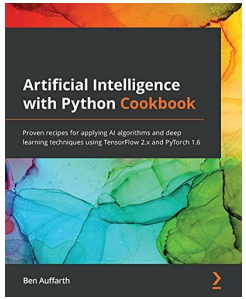
Ben Auffarth

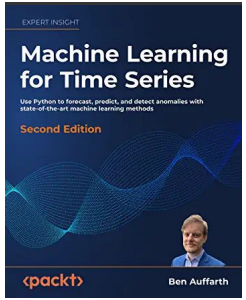
PyData Bristol, April 2023

Who am I?

Ben Auffarth

- Head of data science at loveholidays
- Background in computational and cognitive neuroscience
  - Designed and conducted wet lab experiments on cell cultures
  - Analysed experiments with terabytes of data
  - Run brain models on IBM supercomputers with up to 64k cores
- In a variety of projects and companies:
  - Credit rating, ranking, recommenders, forecasting, NLP, A/B testing, ad bidding, marketing attribution,
  - Built production systems processing hundreds of thousands of transactions per day
  - Trained neural networks on millions of text documents
  - Wrote strategy roadmaps
- Authored two books on machine learning
- Co-founder and former president of Data Science Speakers, London
- Love telling stories to my son





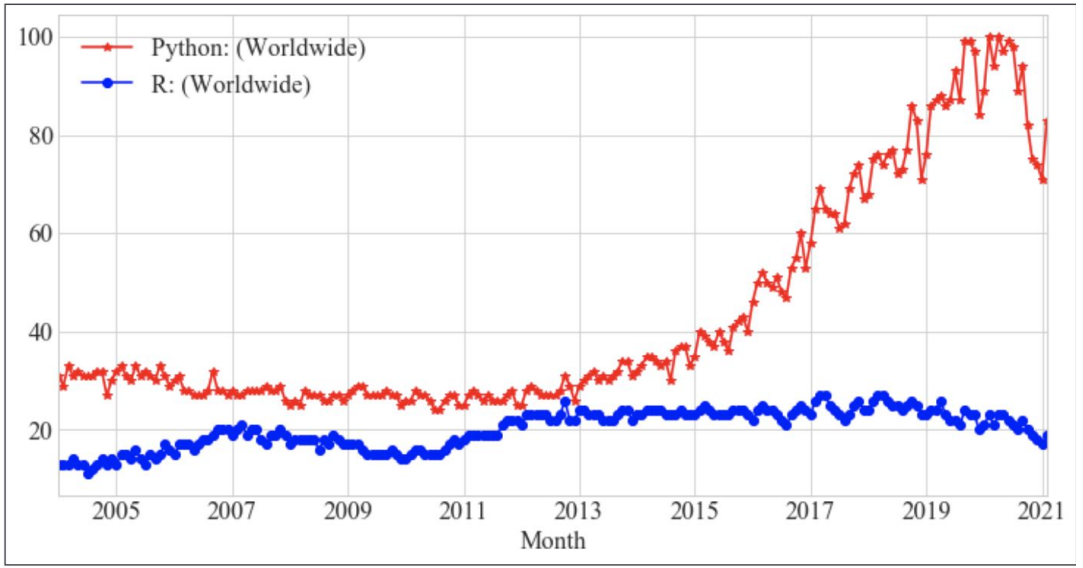
many examples online:
https://github.com/benman1/python-time-series

Machine Learning for Time Series

- Applications
- Time Series concepts
- Preprocessing
- Python ecosystem
  - Forecasting as Regression
- Feature Engineering

What's a Time Series?

- Numerical data obtained at (regular) time-intervals



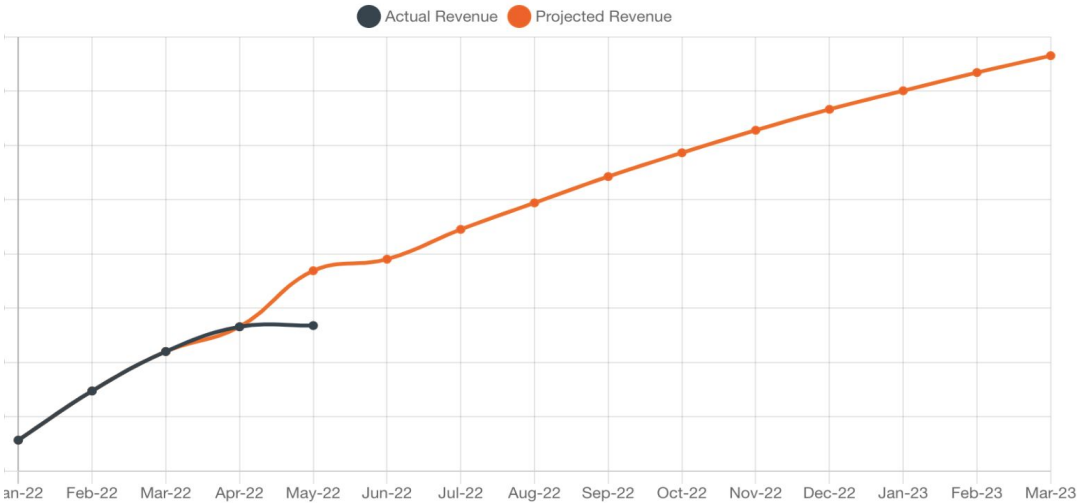
Month	Python: (Worldwide)	R: (Worldwide)	Julia: (Worldwide)
2004-01-01	31	13	1
2004-02-01	29	13	1
2004-03-01	33	13	1
2004-04-01	31	14	1
2004-05-01	32	13	1

Time Series Applications (1)

-

Time Series Applications (1)

- Election forecasts
- Demand/sales forecasting
- Electricity load
- Component failure (predictive maintenance)
- Churn



Time Series Applications (2)

- Website traffic
- Econometrics
- Weather forecast
- Astronomy
- Trading

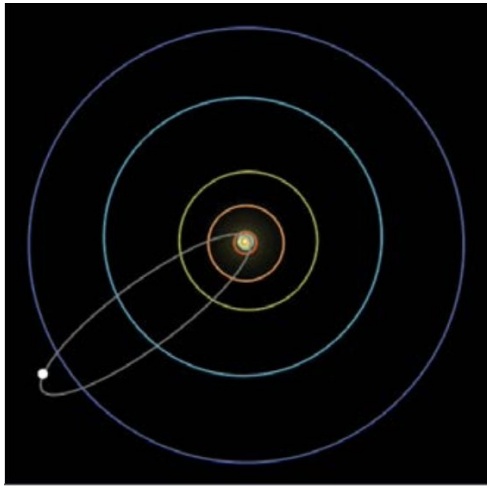


Figure 1.4: Halley's Comet orbit

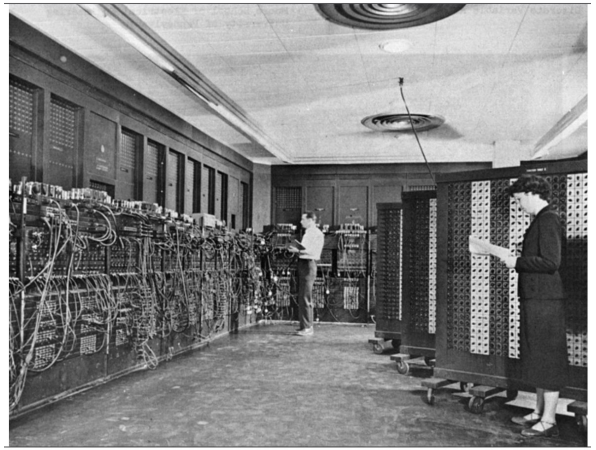


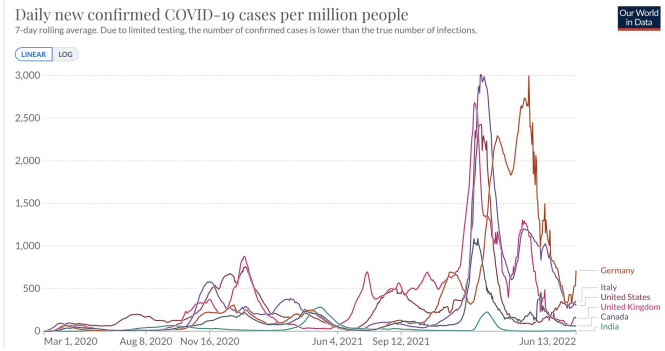
Figure 1.6: Electronic Numerical Integrator and Computer (ENIAC)

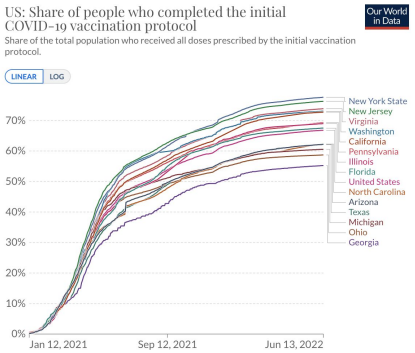
Time Series Approaches

- Time Series analysis
  - describe and summarise time-series data
- Forecasting
  - Statistical and forecasting as regression
- Regression, classification
  - fit models and make predictions
- Unsupervised approaches
  - Clustering
  - Anomaly detection

Terminology (1)

- **Univariate** vs **multivariate**
  - A **univariate** time-series has a single time-dependent variable
  - A **multivariate** time-series has more than one time-dependent dependent variable
- **Endogenous** vs **exogenous** features
- Forecasting, classification, regression

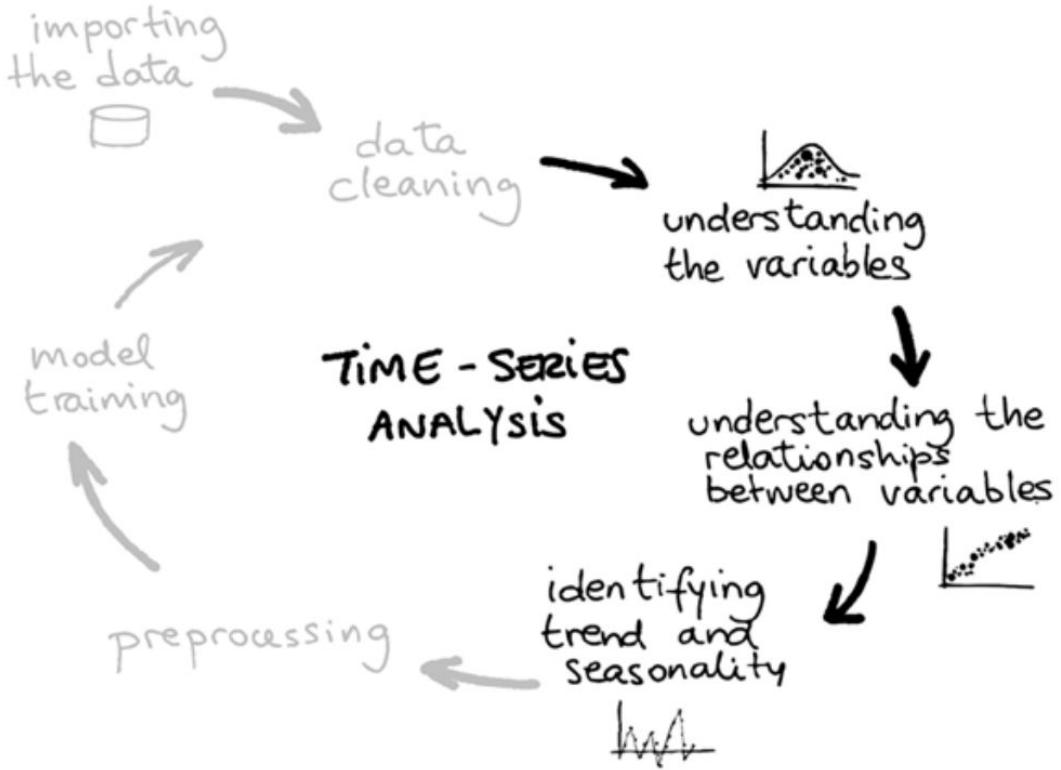




Terminology (2)

- **Stationarity**
- **Linearity**
- **Trend**
- **Seasonality**

Time Series Analysis



Error measures for time-series

Error measures for time-series

MAE, MSE, RMSE, MAPE, SMAPE, MASE, EMALE

Mean Absolute Percentage Error $$MAPE = \frac{1}{n}\sum_{t=1}^{n}|\frac{Y_t - F_t}{Y_t}|$$

Symmetric Mean Absolute Percentage Error $$SMAPE = \frac{1}{n}\sum_{t=1}^{n}\frac{|F_t - Y_t|}{(Y_t + F_t)/2}$$

Mean Absolute Scaled Error $$MASE = \frac{1}{n}\sum_{t=1}^{n}\frac{|Y_t - F_t|}{\frac{1}{T-1}\sum_{t=2}^{n}|Y_t - Y_{t-1}|}$$

Exponentiated Mean Absolute Log Error

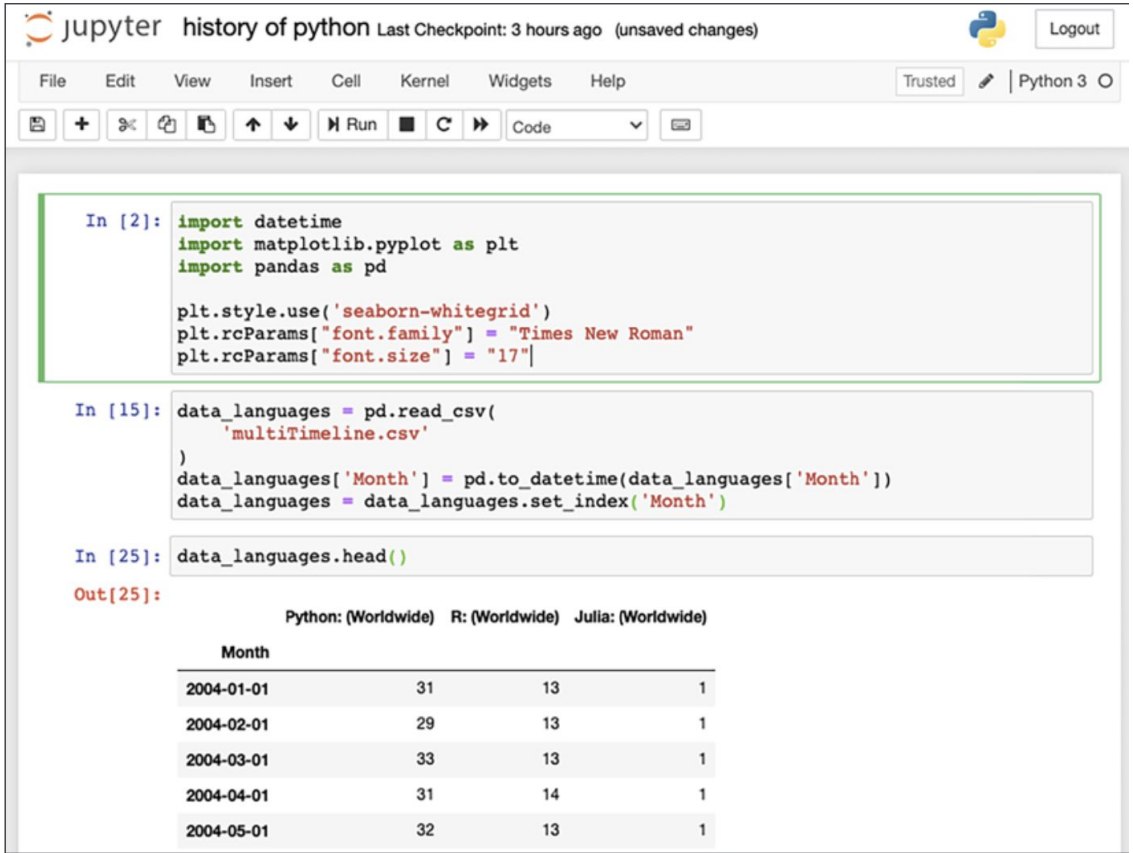
$$EMALE = \exp(\frac{1}{n}\sum_{t=1}^{n}|\log(Y_t) - \log(F_t)|)$$

Preprocessing

Preprocessing

- Decomposition
- Feature engineering

Loading a Time-Series



Decomposition (1)

```
from numpy import polyfit

def fit(X, y, degree=3):
    coef = polyfit(X, y, degree)
    trendpoly = np.poly1d(coef)
    return trendpoly(X)

def get_season(s, yearly_periods=4, degree=3):
    X = [i%(365/4) for i in range(0, len(s))]
    seasonal = fit(X, s.values, degree)
    return pd.Series(data=seasonal, index=s.index)

def get_trend(s, degree=3):
    X = list(range(len(s)))
    trend = fit(X, s.values, degree)
    return pd.Series(data=trend, index=s.index)
```

Decomposition (2)

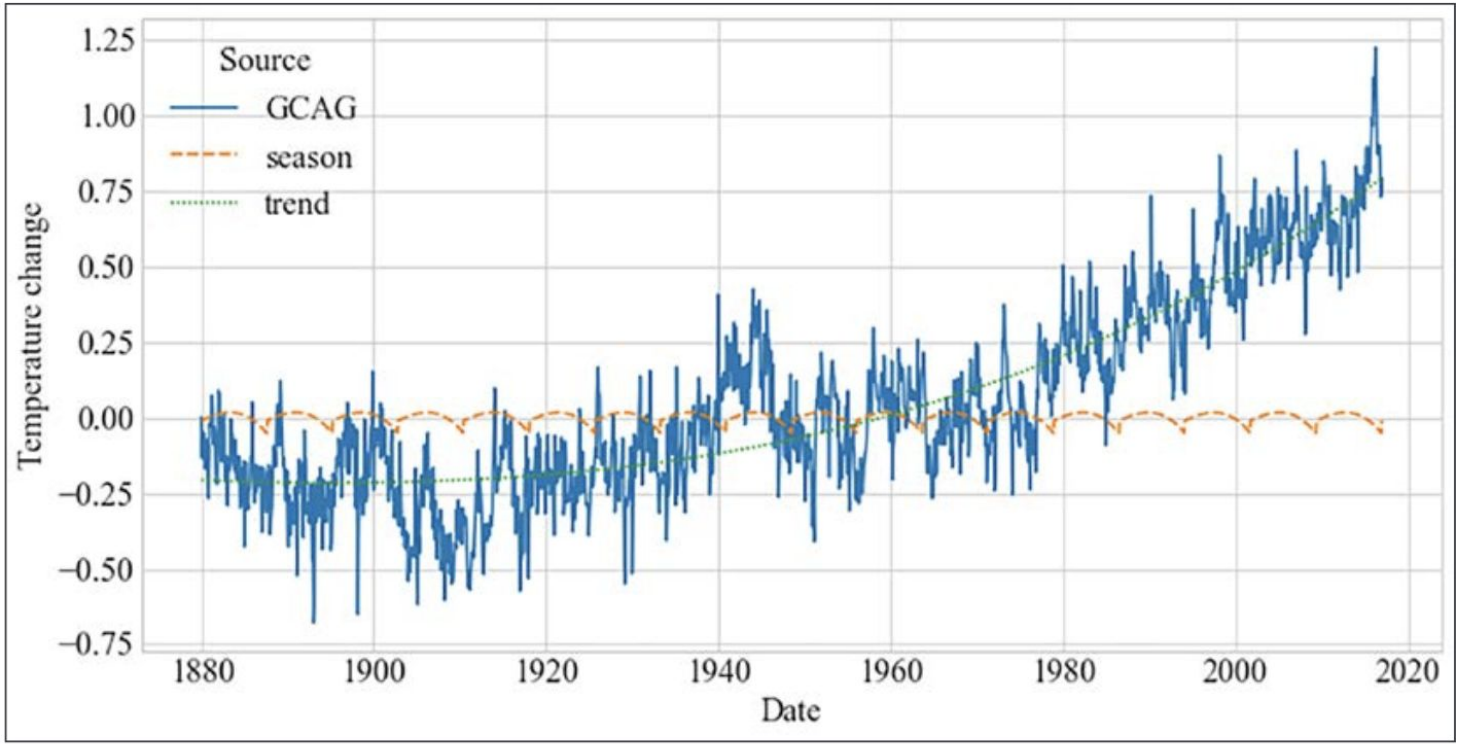
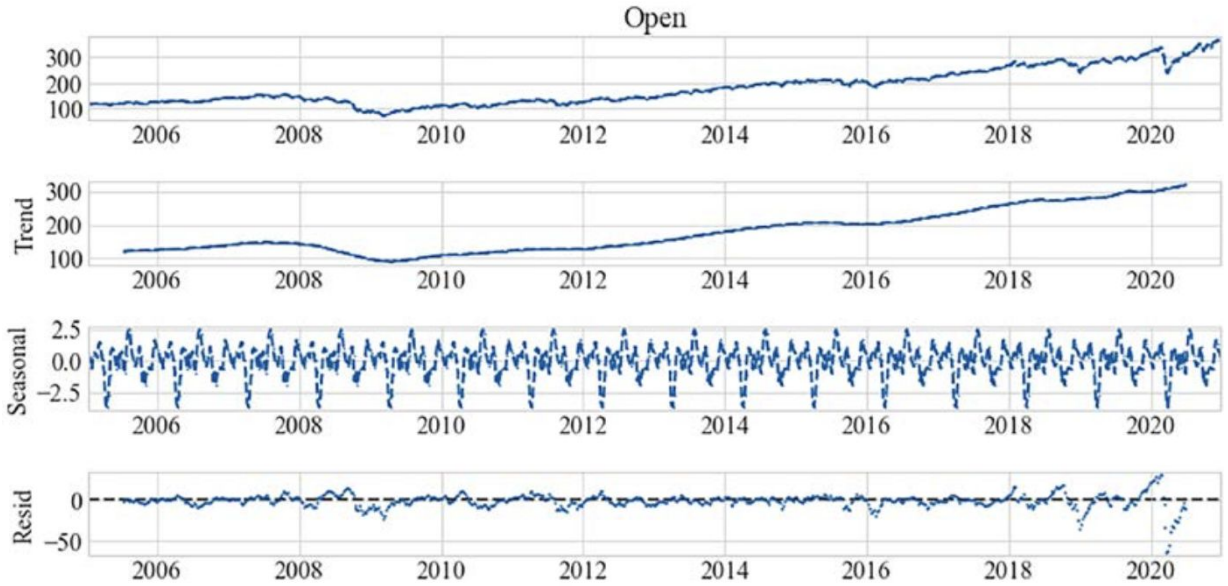


Figure 2.9: Temperature change from the late 19th century to today

Decomposition (3)

```
from statsmodels.tsa.seasonal import seasonal_decompose
result = seasonal_decompose(df, model='additive', period=52)
result.plot()
```

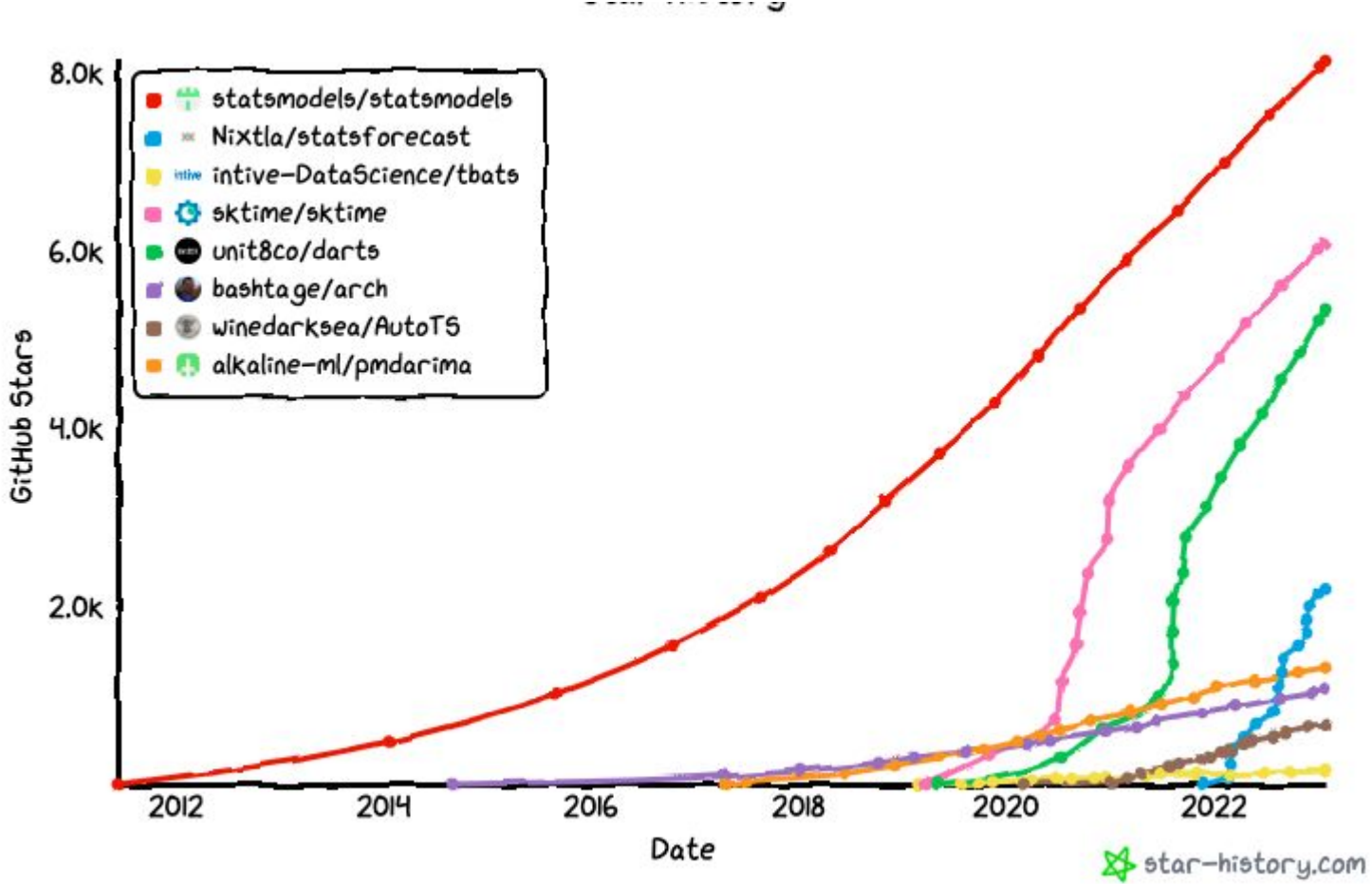


Python libraries for time-series

- statsmodels
- statsforecast
- sktime
- darts
- gluon-ts
- tslearn
- pyts
- seglearn
- cesium
- pmdarima
- prophet

Language	Repositories for time-series
Jupyter Notebook	11,297
Python	4,891
R	3,656
Julia	104

Python libraries with classical models



Models

	Exogenous variables	Multivariate	Training Speed	Complexity	Probabilistic
Generalized Linear Model	Yes	No	Fast	Low	No
Exponential Smoothing	No	No	Fast	Low	No
UnobservedComponents	Yes	No	Fast	Low	Yes
ARIMA	Yes	No	Slow	Low	No
VARMAX	Yes	Yes	Fast	Low	Yes
Theta	No	No	Medium	Low	Yes
lightgbm	Yes	Yes	Slow	Medium	No
LSTM	Yes	Yes	Slow	High	No
Transformer	Yes	Yes	Slow	High	No
CNN	Yes	Yes	Slow	High	No

Stationarity can be enforced

Statsmodels

Class	Description
`ar_model.AutoReg`	Univariate Autoregression Model
`arima.model.ARIMA`	Autoregressive Integrated Moving Average (ARIMA) model
`ExponentialSmoothing`	Holt Winter's Exponential Smoothing
`SimpleExpSmoothing`	Simple Exponential Smoothing

Figure 5.4: A few models implemented in statsmodels

Function	Description
`stattools.kpss`	Kwiatkowski-Phillips-Schmidt-Shin test for stationarity
`stattools.adfuller`	Augmented Dickey-Fuller unit root test
`stattools.ccf`	The cross-correlation function
`stattools.pacf`	Partial autocorrelation estimate
`stats.diagnostic.het_arch`	Engle's Test for Autoregressive Conditional Heteroscedasticity (ARCH), also referred to as the ARCH-LM test
`stattools.q_stat`	Ljung-Box Q Statistic
`tsa.seasonal.seasonal_decompose`	Seasonal decomposition using moving averages
`tsa.tsatools.detrend`	Detrend a vector

Figure 5.5: Useful functions in statsmodels

Statsforecast

Model	Point Forecast	Probabilistic Forecast	Insample fitted values	Probabilistic fitted values
Theta	✅	✅	✅	✅
OptimizedTheta	✅	✅	✅	✅
DynamicTheta	✅	✅	✅	✅
DynamicOptimizedTheta	✅	✅	✅	✅
SimpleExponentialSmoothing	✅			
SimpleExponentialSmoothingOptimized	✅			
SeasonalExponentialSmoothing	✅			
SeasonalExponentialSmoothingOptimized	✅			
Holt	✅	✅	✅	✅
HoltWinters	✅	✅	✅	✅

Automatic Forecasting

Automatic forecasting tools search for the best parameters and select the best possible model for a group of time series. These tools are useful for large collections of univariate time series.

Model	Point Forecast	Probabilistic Forecast	Insample fitted values	Probabilistic fitted values
AutoARIMA	✅	✅	✅	✅
AutoETS	✅	✅	✅	✅
AutoCES	✅	✅	✅	✅
AutoTheta	✅	✅	✅	✅

Nixtla ecosystem

Table available on [Darts github page]

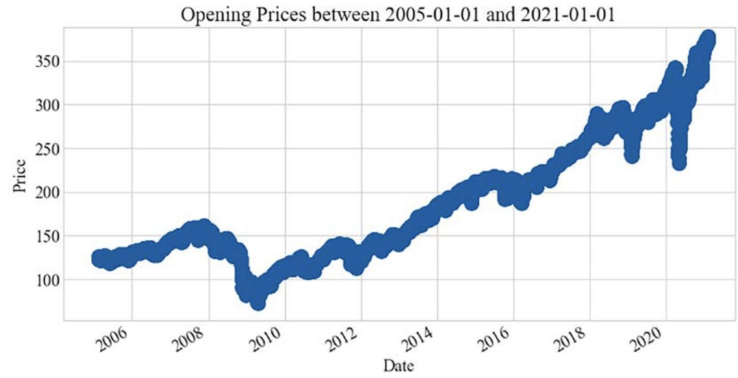
Darts

Model	Univariate	Multivariate	Probabilistic	Multiple-series training	Past-observed covariates support	Future-known covariates support
`ARIMA`	✅		✅			✅
`VARIMA`	✅	✅				✅
`AutoARIMA`	✅					✅
`ExponentialSmoothing`	✅		✅			
`Theta` and `FourTheta`	✅					
`Prophet`	✅		✅			✅
`FFT` (Fast Fourier Transform)	✅					
`RegressionModel` (incl `RandomForest`, `LinearRegressionModel` and `LightGBMModel`)	✅	✅		✅	✅	✅
`RNNModel` (incl. LSTM and GRU); equivalent to DeepAR in its probabilistic version	✅	✅	✅	✅		✅
`BlockRNNModel` (incl. LSTM and GRU)	✅	✅	✅	✅	✅	
`NBEATSModel`	✅	✅	✅	✅	✅	
`TCNModel`	✅	✅	✅	✅	✅	
`TransformerModel`	✅	✅	✅	✅	✅	
`TFTModel` (Temporal Fusion Transformer)	✅	✅	✅	✅	✅	✅
Naive Baselines	✅					

Some models support Being trained on multiple Series. We call them **global models**.

Table available on [Darts github page]

Example with classical models (1)




```
from datetime import datetime
import yfinance as yf

start_date = datetime(2005, 1, 1)
end_date = datetime(2021, 1, 1)

df = yf.download(
    'SPY',
    start=start_date,
    end = end_date
)
```

```
from statsmodels.tsa.forecasting.stl import STLForecast
mod = STLForecast(
  df1, sm.tsa.arima.ARIMA,
  model_kwargs=dict(order=(1, 1, 0), trend="t")
)
res = mod.fit().model_result
```

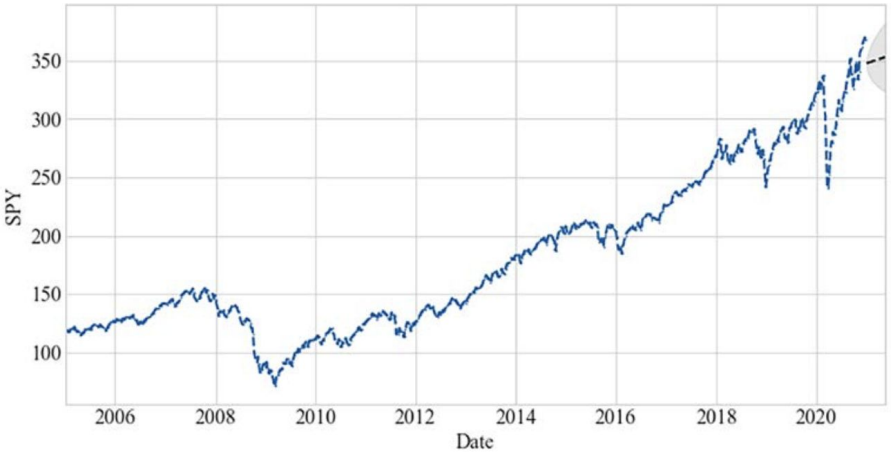
Example with classical models (2)

```
STEPS = 20
forecasts_df = res.get_forecast(steps=STEPS).summary_frame()
```

This gives us a forecast 20 steps into the future.

Let's visualize this!

```
ax = df1.plot(figsize=(12, 6))
plt.ylabel('SPY')
forecasts_df['mean'].plot(style='k--')
ax.fill_between(
    forecasts_df.index,
    forecasts_df['mean_ci_lower'],
    forecasts_df['mean_ci_upper'],
    color='k',
    alpha=0.1
)
```



Example with classical models (3)

```
from statsforecast.models import (
    AutoARIMA, ETS, AutoTheta, AutoCES,
    SimpleExponentialSmoothing, MSTL,
    SeasonalExponentialSmoothing, HoltWinters,

)

models = [
    ETS(season_length=season_length),
    Naive(),
    AutoTheta(season_length=season_length),
    AutoARIMA(season_length=season_length),
    AutoCES(season_length=season_length),
    SimpleExponentialSmoothing(alpha=0.2),
    SeasonalExponentialSmoothing(season_length=season_length, alpha=0.2),
    MSTL(season_length=season_length),
]
sf = StatsForecast(
    df=Y_train_df,
    models=models,
    freq='M',
    n_jobs=-1
)
Y_hat_df = sf.forecast(horizon)
```

Regression as Forecasting

- use established ml techniques to capture relationships

Window Regression vs Date Part Regression

f(exogenous, lagged endogenous) = forecast of endogenous variables

```
>>> from statsmodels.tsa.tsatools import lagmat
>>> import numpy as np
>>> X = np.arange(1,7).reshape(-1,2)
>>> X
array([[1, 2],
       [3, 4],
       [5, 6]])
>>> lagmat(X, maxlag=2, trim="forward", original='in')
array([[1., 2., 0., 0., 0., 0.],
       [3., 4., 1., 2., 0., 0.],
       [5., 6., 3., 4., 1., 2.]])
```

```
In [13]: np.roll(X, shift=-1, axis=0)
Out[13]:
array([[3, 4],
       [5, 6],
       [1, 2]])
```

Regression as Forecasting

SkTime

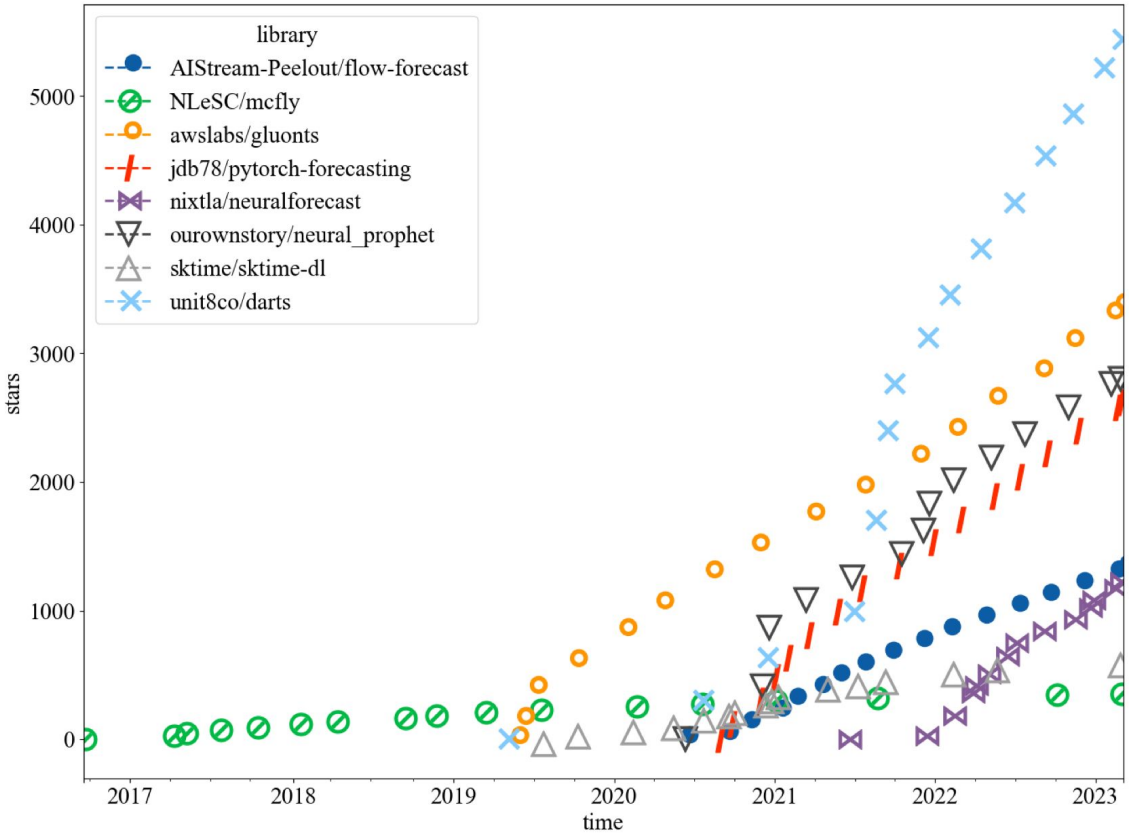
```
from sklearn.neighbors import KNeighborsRegressor
from sktime.forecasting.compose import make_reduction
regressor = KNeighborsRegressor(n_neighbors=1)
forecaster = make_reduction(regressor, window_length=15, strategy="recursive")
forecaster.fit(y_train)
y_pred = forecaster.predict(MAX_HORIZON)
```

XGBoost with Darts

```
import pandas as pd
from darts.timeseries import TimeSeries
#from darts.models.forecasting.lgbm import LightGBMModel
from darts.models.forecasting.xgboost import XGBModel
from darts.datasets import AirPassengersDataset

series = AirPassengersDataset().load()
# alternatively, something like this:
# df = pd.read_csv('AirPassengers.csv', delimiter=",")
# series = TimeSeries.from_dataframe(df, 'Month', ['#Passengers'])
train, val = series.split_before(pd.Timestamp('19540101'))
model = XGBModel(lags=15)
model.fit(train)
forecast = model.predict(HORIZON)
```

Deep Learning

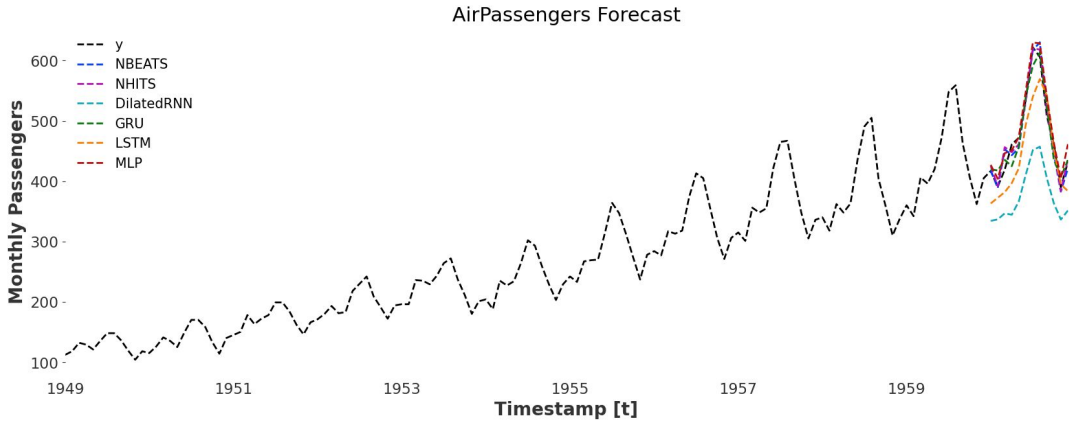


Deep Learning (1)

```python
import matplotlib.pyplot as plt
from neuralforecast import NeuralForecast
from neuralforecast.models import (
    NBEATS, NHITS, DilatedRNN, GRU, LSTM, MLP
)
from neuralforecast.utils import AirPassengersDF

Y_df = AirPassengersDF
Y_train_df = Y_df[Y_df.ds<='1959-12-31'] # 132 train
Y_test_df = Y_df[Y_df.ds>'1959-12-31'] # 12 test

horizon = len(Y_test_df)
params = dict(
    input_size=2 * horizon, h=horizon, max_epochs=500
)
models = [
    NBEATS(**params),
    NHITS(**params),
    DilatedRNN(**params),
    GRU(**params),
    LSTM(**params),
    MLP(**params)
]
nforecast = NeuralForecast(models=models, freq='M')
nforecast.fit(df=Y_train_df)
Y_hat_df = nforecast.predict().reset_index()
```



Deep Learning (2)

```
from time_series.dataset.utils import get_energy_demand
from time_series.dataset.time_series import TrainingDataSet

train_df = get_energy_demand()
tds = TrainingDataSet(train_df)
```

```
from time_series.models.deepar import DeepAR
ar_model = DeepAR(tds)
ar_model.instantiate_and_fit(verbose=1, epochs=N_EPOCHS)
```

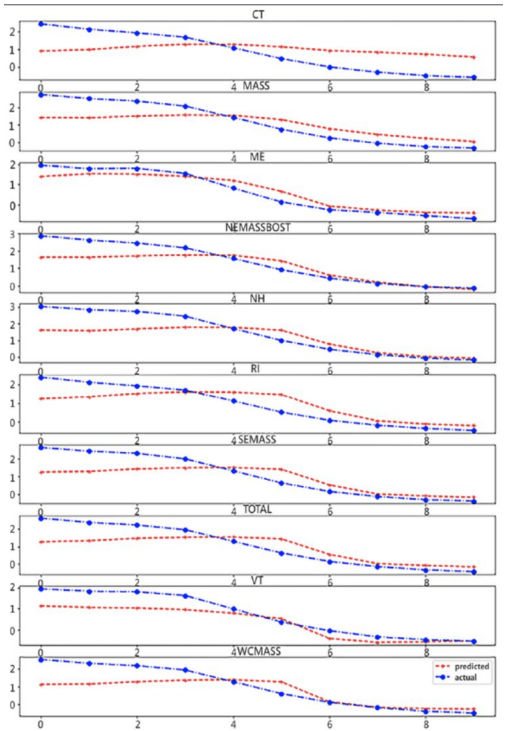
https://github.com/benman1/time-series

Deep Learning (3)

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 10, 10)]	0
lstm_1 (LSTM)	(None, 10, 4)	240
dense_1 (Dense)	(None, 10, 4)	20
main_output (GaussianLayer)	[(None, 10, 10), (None, 1	100

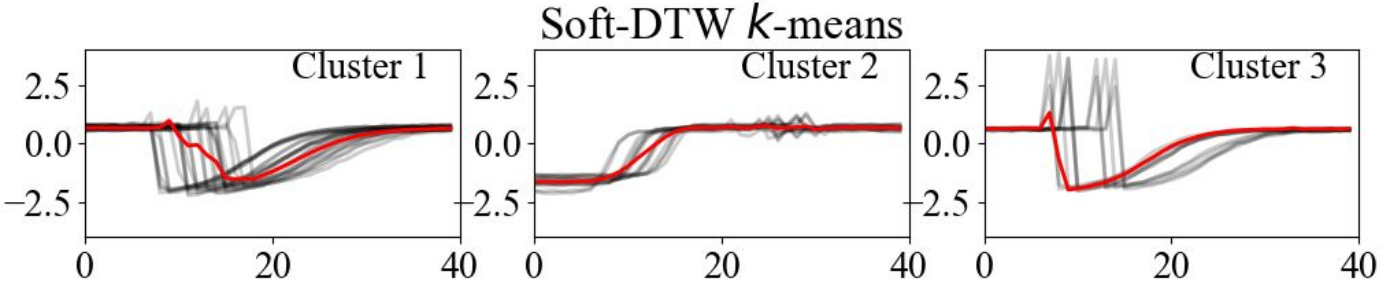
Total params: 360
Trainable params: 360
Non-trainable params: 0

```
y_predicted = ar_model.model.predict(tds.X_test)
evaluate_model(tds=tds, y_predicted=y_predicted,
    columns=train_df.columns, first_n=10)
```



https://github.com/benman1/time-series

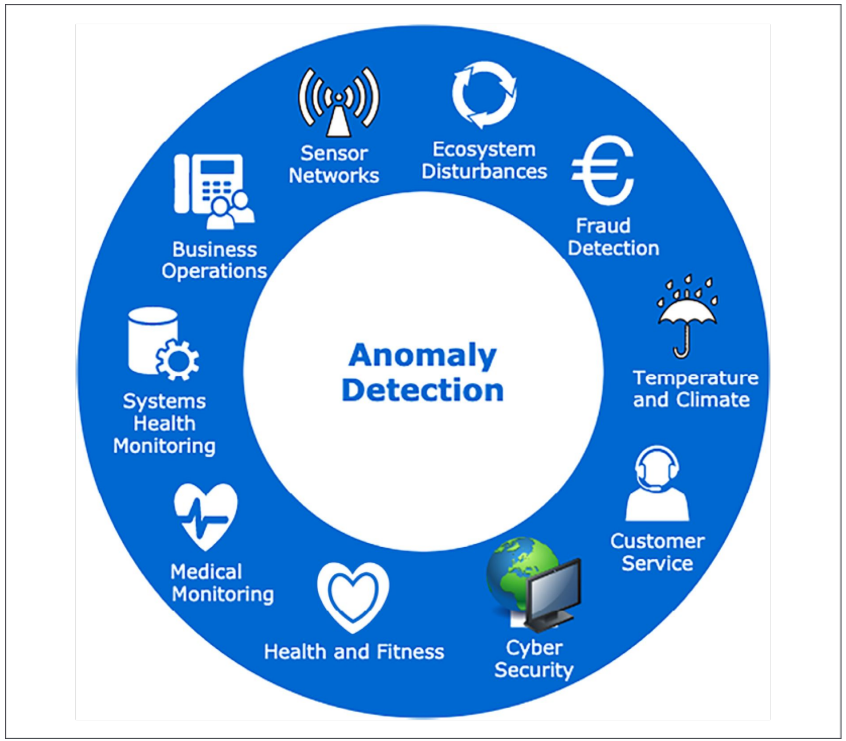
Clustering

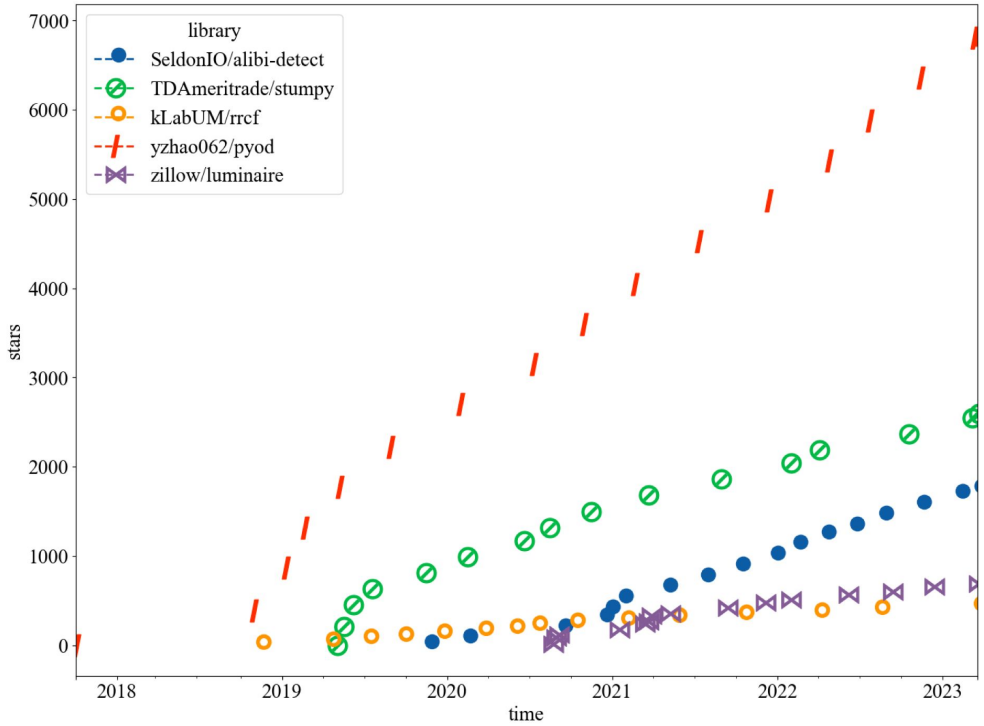


tslearn

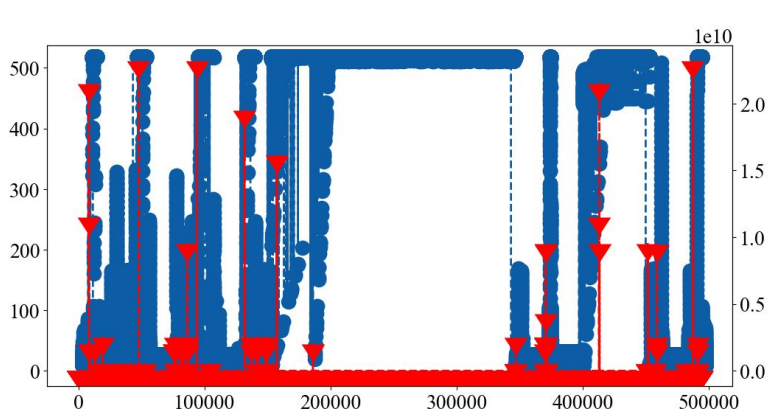
```
sdtw_km = TimeSeriesKMeans(
    n_clusters=3,
    metric="softdtw",
    metric_params={"gamma": .01},
    verbose=True,
    random_state=seed
)
y_pred = sdtw_km.fit_predict(X_train)
plot_clusters(offset=1, title="Soft-DTW $k$-means")
```

Anomaly detection





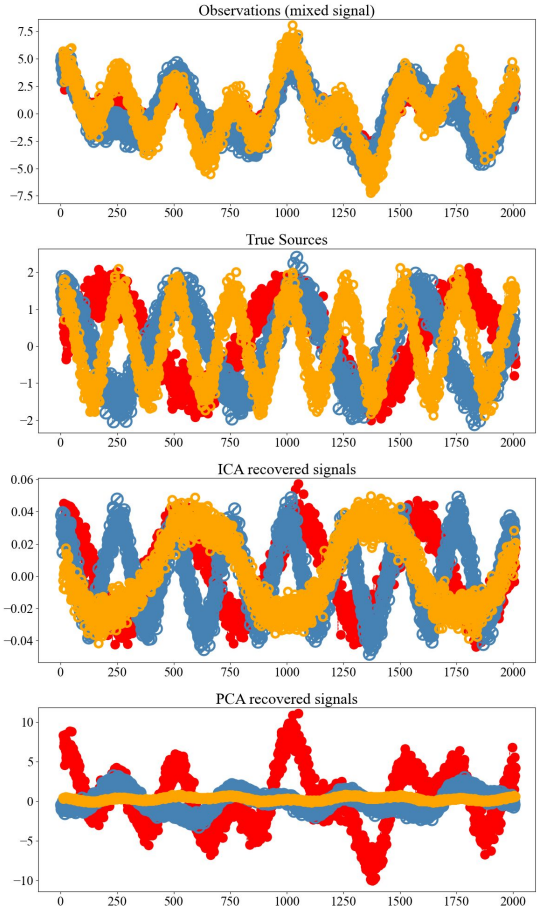
Anomaly detection



pyOD

```
from pyod.models.lof import LOF
clf = LOF()
clf.fit(intrusions['data'][:, 0].reshape(-1, 1))
```

Dimensionality reduction



Feature Engineering

Feature engineering (1) - hand-crafted features

- Machine learning algorithms can use different representations of the input features (preprocessing/feature engineering)
- Hand-crafted vs automated feature extraction
- Interpretable: mean, max, min, etc
  - Pooled within windows
- Catch22
  - features and simple summary statistics extracted from phase dependant intervals
  - Reduced set from Highly Comparative Time Series Analysis (hctsa) toolbox
- Calendar features

- trade-off between accuracy and prediction times
  - Huge differences in time complexity and model performance

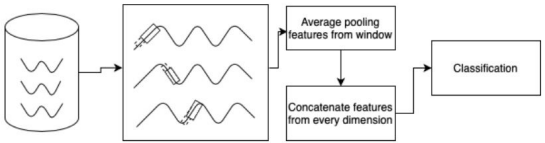


Figure 2: Feature extraction and classification using a rolling window.

- **4-stat:** 4 features (mean, min, max, std dev)
- **9-stat:** To above, add median, kurtosis, number of peaks, 25 and 75 percentile.

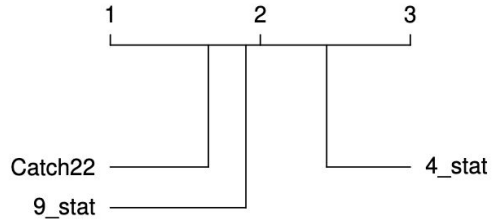
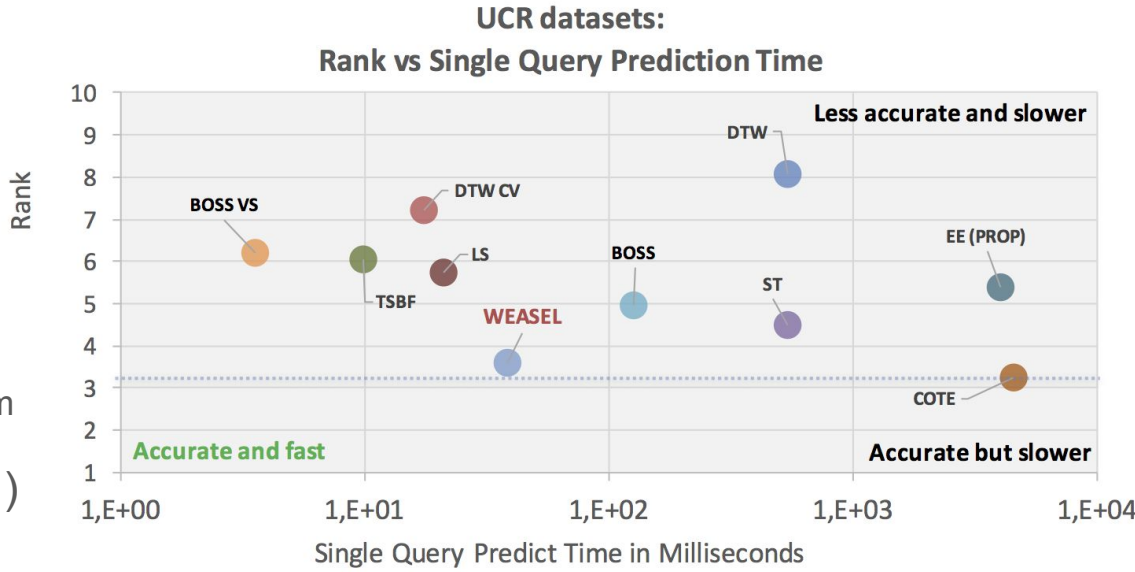


Figure 4: Critical difference diagram for methods based only on statistical features on 26 datasets.

Feature engineering (2)

- Non-Interpretable
- SIFT features
  - scale-invariant features
  - shapes surrounding the extremum (Bailly and others, 2015)
- Shapelets (Ye and Keogh 2011)
- ROCKET
  - MINIROCKET
- BOSS
  - histograms of n-grams to form bag-of-patterns (BoP)
  - BOSS in Vector Space (BOSS VS)
- WEASEL+MUSE
  - Word Extraction for Time Series Classification
  - Multivariate Unsupervised Symbols and Derivatives



Calendar features (1)

```
YEAR = 2021
seasons = [
    ('winter', (date(YEAR,  1,  1),  date(YEAR,  3, 20))),
    ('spring', (date(YEAR,  3, 21),  date(YEAR,  6, 20))),
    ('summer', (date(YEAR,  6, 21),  date(YEAR,  9, 22))),
    ('autumn', (date(YEAR,  9, 23),  date(YEAR, 12, 20))),
    ('winter', (date(YEAR, 12, 21),  date(YEAR, 12, 31)))
]

def is_in_interval(current_date: datetime.date, seasons):
      return next(season for season, (start, end) in seasons
                  if start <= current_date.replace(year=YEAR) <= end)

is_in_interval(today, seasons)
```

Calendar (2)

```
from astral.sun import sun
from astral import LocationInfo
CITY = LocationInfo("London", "England", "Europe/London", 51.5, -0.116)
def get_sunrise_dusk(current_date: datetime.date, city_name='London'):
      s = sun(CITY.observer, date=current_date)
      sunrise = s['sunrise']
      dusk = s['dusk']
      return (sunrise - dusk).seconds / 3600

get_sunrise_dusk(today)
```

Calendar (3)

```
def get_business_days(current_date: datetime.date):
      last_day = calendar.monthrange(current_date.year, current_date.
month)[1]
      rng = pd.date_range(current_date.replace(day=1), periods=last_
day, freq='D')
      business_days = pd.bdate_range(rng[0], rng[-1])
      return len(business_days), last_day - len(business_days)

get_business_days(date.today())
```

Automated Feature Extraction

```
import featuretools as ft
from featuretools.primitives import Minute, Hour, Day, Month, Year,
Weekday

data = pd.DataFrame(
    {'Time': ['2014-01-01 01:41:50',
              '2014-01-01 02:06:50',
              '2014-01-01 02:31:50',
              '2014-01-01 02:56:50',
              '2014-01-01 03:21:50'],
     'Target': [0, 0, 0, 0, 1]}
)
data['index'] = data.index
es = ft.EntitySet('My EntitySet')
es.entity_from_dataframe(
    entity_id='main_data_table',
    index='index',
    dataframe=data,
    time_index='Time'
)
fm, features = ft.dfs(
    entityset=es,
    target_entity='main_data_table',
    trans_primitives=[Minute, Hour, Day, Month, Year, Weekday]
)
```

Deep Learning

Deep generative model for the probabilistic nowcasting of precipitation from radar
Architecture blocks:

- feed-forward convolutional neural network (conditioning stack)
- ConvGRU (sampler)
- Normal, Convolutions (latent conditioning stack)

Article | Open Access | Published: 29 September 2021

Skilful precipitation nowcasting using deep generative models of radar

Suman Ravuri, Karel Lenc, Matthew Willson, Dmitry Kangin, Remi Lam, Piotr Mirowski, Megan Fitzsimons, Maria Athanassiadou, Sheleem Kashem, Sam Madge, Rachel Prudden, Amol Mandhane, Aidan Clark, Andrew Brock, Karen Simonyan, Raia Hadsell, Niall Robinson, Ellen Clancy, Alberto Arribas & Shakir Mohamed

Nature **597**, 672–677 (2021) | Cite this article

Some algorithms

- COTE (Collective Of Transformation-based Ensembles
  - Ensemble of 35 classifiers
  - Univariate
- HIVE-COTE
  - Extension of COTE with hierarchical structure and probabilistic voting
  - hugely computationally intensive $O(N^2 \cdot T^4)$
- ROCKET

Type	UTSC	MTSC
Distance-based	DTW	DTW-D
Shapelets	Shapelet Transform	MrSEQL
Symbolic	SAX-VSM, BOSS, WEASEL	WEASEL-MUSE
Ensemble	EE, COTE, HIVE-COTE, TS-CHIEF	ROCKET
Deep Learning	ResNet, FCN	TapNet

HIVE-COTE

1.0:

- Shapelet Transform Classifier (STC)
- Time-Series Forest
- Contract Bag of Symbolic-Fourier Approximation Symbols (cBOSS)
- Random Interval Spectral Ensemble (c-RISE)

More improvements coming:

- Temporal Dictionary Ensemble (TDE, 2021)
- Canonical Interval Forest (CIF)

Implementations

- Sktime
- Pyts
- Other
  - https://github.com/hfawaz
  - https://github.com/angus924/minirocket

Table 2 Classifier availability in the two toolkits tsml and sktime

Algorithm	tsml	sktime	sktime-dl	sktime-shapelets
DTW_D	X	X		
DTW_I	X	X		
DTW_A	X			
MUSE	X			
gRFS				X
MrSEQL		X		
ROCKET		X		
CIF	X	X		
TapNet				
ResNet			X	
InceptionTime			X	
CBOSS	X	X		
STC	X	X		X
RISE	X	X		
TSF	X	X		
HIVE-COTE	X	X		

Multivariate Time-Series

Data Mining and Knowledge Discovery
https://doi.org/10.1007/s10618-020-00727-3

Check for updates

The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances

Alejandro Pasos Ruiz[1] · Michael Flynn[1] · James Large[1] · Matthew Middlehurst[1] · Anthony Bagnall[1]

Received: 24 August 2020 / Accepted: 25 November 2020

Deep learning for time series classification: a review

Hassan Ismail Fawaz[1] · Germain Forestier[1,2] · Jonathan Weber[1] · Lhassane Idoumghar[1] · Pierre-Alain Muller[1]

Benchmarks for Time-Series

- Classification vs regression
  - Unsupervised vs supervised
- Univariate
  - Each case has a single series and a class label
  - Emphasis of earlier models (classical modelling)
  - UCR archive: about 120 datasets
- Multivariate
  - time series for a single case has multiple dimensions
  - Formally k>1
  - More frequent in practice
  - UAE archive: 30 datasets
  - MTS archive: 13 datasets.

Dataset	Train Cases	Test Cases	Dimensions	Length	Classes
ArticularyWordRecognition	275	300	9	144	25
AtrialFibrillation	15	15	2	640	3
BasicMotions	40	40	6	100	4
CharacterTrajectories	1422	1436	3	182	20
Cricket	108	72	6	1197	12
DuckDuckGeese	60	40	1345	270	5
EigenWorms	128	131	6	17984	5
Epilepsy	137	138	3	206	4
EthanolConcentration	261	263	3	1751	4
ERing	30	30	4	65	6
FaceDetection	5890	3524	144	62	2
FingerMovements	316	100	28	50	2
HandMovementDirection	320	147	10	400	4
Handwriting	150	850	3	152	26
Heartbeat	204	205	61	405	2
JapaneseVowels	270	370	12	29	9
Libras	180	180	2	45	15
LSST	2459	2466	6	36	14
InsectWingbeat	30000	20000	200	78	10
MotorImagery	278	100	64	3000	2
NATOPS	180	180	24	51	6
PenDigits	7494	3498	2	8	10
PEMS-SF	267	173	963	144	7
Phoneme	3315	3353	11	217	39
RacketSports	151	152	6	30	4
SelfRegulationSCP1	268	293	6	896	2
SelfRegulationSCP2	200	180	7	1152	2
SpokenArabicDigits	6599	2199	13	93	10
StandWalkJump	12	15	4	2500	3
UWaveGestureLibrary	120	320	3	315	8

Table 2: A summary of the 30 datasets in the UEA Multivariate Time Series Classification archive, 2018

The great multivariate time series classification bake off

- Review/Survey paper: Pasos Ruiz et al 2020
- multivariate TSC
- 26 datasets from UAE archive
- Distance-based approaches
  - Dynamic time warping, independent warping, adaptive warping
- MUSE, ...

Results (1)

- Not all finished
  - Comparison of sixteen classifiers on twenty datasets
- Critical difference diagrams
- ROCKET achieves considerable improvement in at least an order of magnitude less time

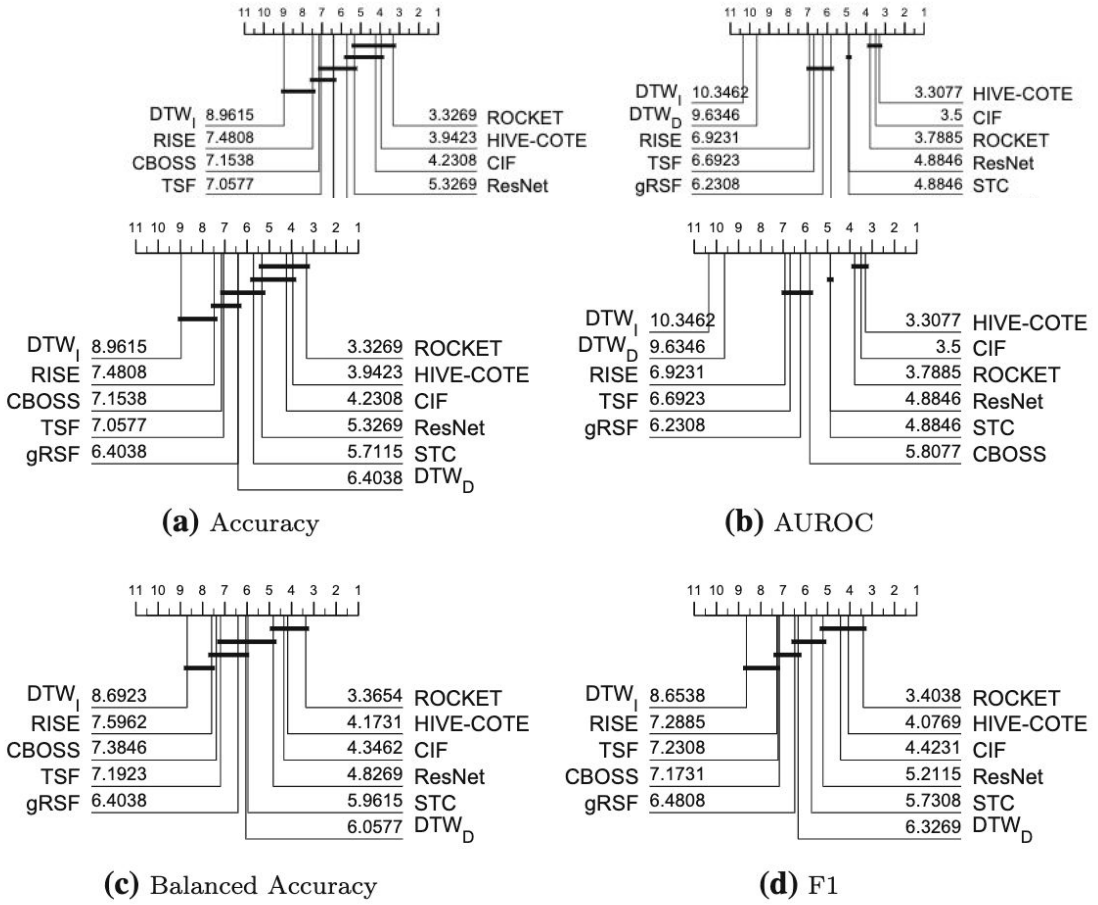


Fig. 7 Critical difference diagrams for 11 classifiers on the 26 equal length UEA datasets using pairwise Wilcoxon test to form cliques

ROCKET

- MINIROCKET (Dempster et al 2020)
  - up to 75x faster on big datasets while maintaining the same accuracy

MINIROCKET: A Very Fast (Almost) Deterministic Transform for Time Series Classification

Angus Dempster, Daniel F. Schmidt, Geoffrey I. Webb

Until recently, the most accurate methods for time series classification were limited by high computational complexity. ROCKET achieves state-of-the-art accuracy with a fraction of the computational expense of most existing methods by transforming input time series using random convolutional kernels, and using the transformed features to train a linear classifier. We reformulate ROCKET into a new method, MINIROCKET, making it up to 75 times faster on larger datasets, and making it almost deterministic (and optionally, with additional computational expense, fully deterministic), while maintaining essentially the same accuracy. Using this method, it is possible to train and test a classifier on all of 109 datasets from the UCR archive to state-of-the-art accuracy in less than 10 minutes. MINIROCKET is significantly faster than any other method of comparable accuracy (including ROCKET), and significantly more accurate than any other method of even roughly-similar computational expense. As such, we suggest that MINIROCKET should now be considered and used as the default variant of ROCKET.

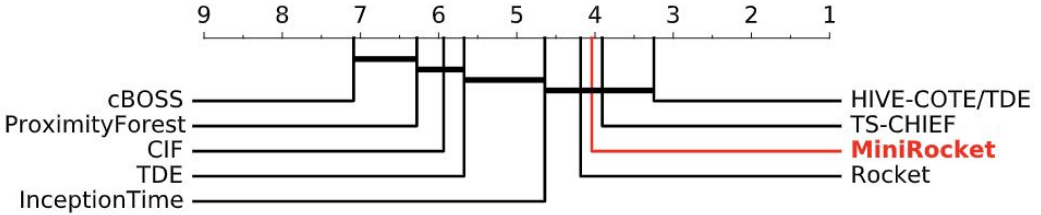


Figure 1: Mean rank of MINIROCKET in terms of accuracy versus other SOTA methods over 30 resamples of 109 datasets from the UCR archive.

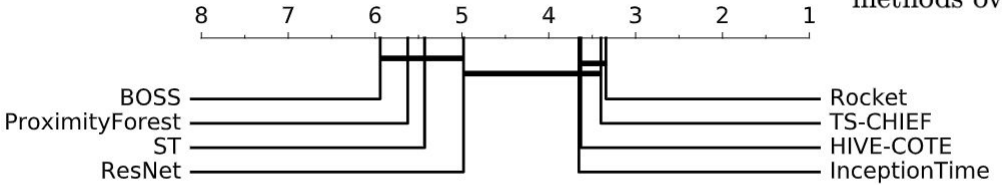


Fig. 1 Mean rank of ROCKET versus state-of-the-art classifiers on the 85 'bake off' datasets.

Deep Learning for Time-Series (1)

- Review/Survey paper: Fawaz and others 2019
- Companion repo
  - tensorflow/keras
- On the univariate UCR/UEA time series classification archive (85 problems)
- 11 classifiers:
  - LeNet, FCN, Time-CNN, ResNet,
- cluster of 60 GPUs
- results in about a month

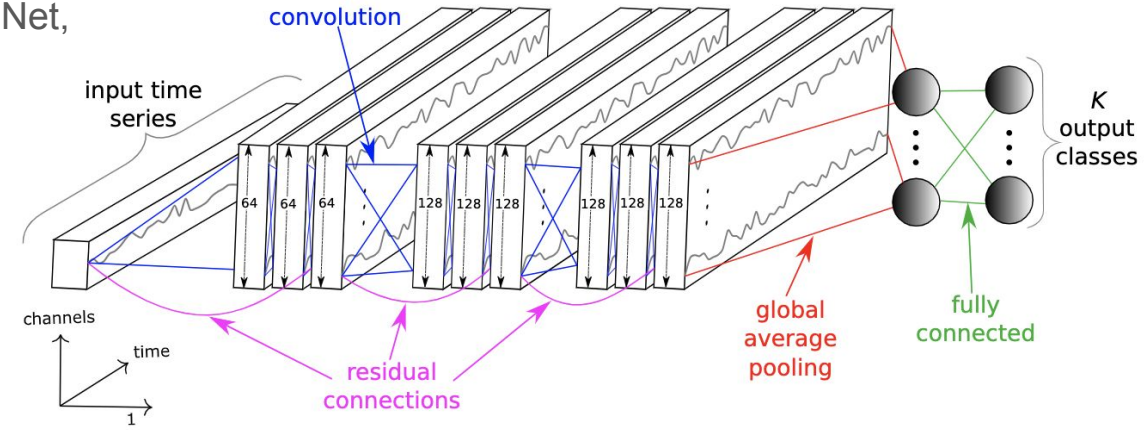


Fig. 6: The Residual Network's architecture for time series classification.

Results (1)

- Comparison
- 9 completed all tests
- ResNet wins on 50 problems out of 85

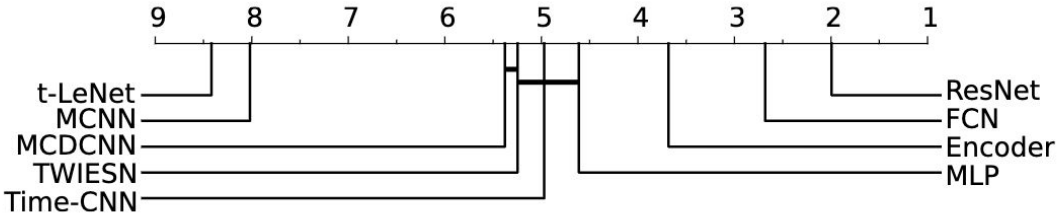


Fig. 7: Critical difference diagram showing pairwise statistical difference comparison of nine deep learning classifiers on the univariate UCR/UEA time series classification archive.

Results (2) - Comparison to SOTA approaches

- Most perform worse than SOTA
- ResNet not significantly worse

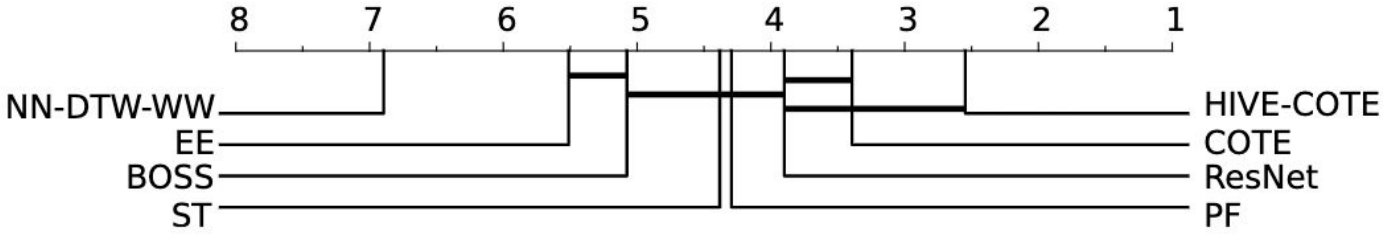
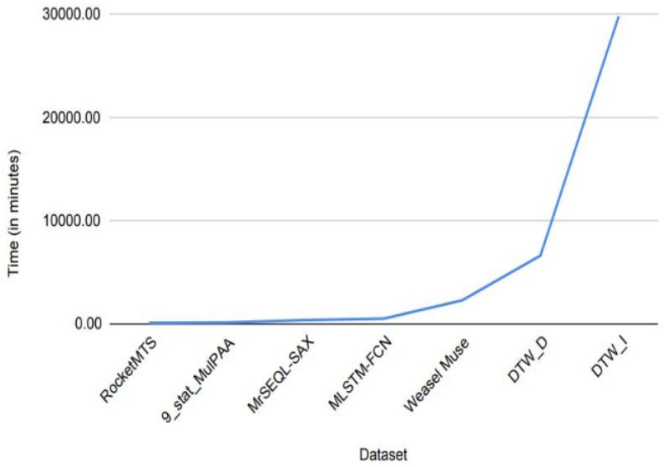


Fig. 8: Critical difference diagram showing pairwise statistical difference comparison of state-of-the-art classifiers on the univariate UCR/UEA time series classification archive.

Deep Learning

- [Dhariyal et al 2020]
- 20 datasets
- Accuracy: ROCKET wins, deep learning not that good
- Runtime: ROCKET: 34 minutes, DTW runs for days



Challenges	ROCKET-MTS	MrSEQL-SAX	DTW_I	DTW_D	WEASEL-MUSE
Variable-length	C		C	C	
Scalability		C	C	C	C
Memory Consumption					C
Overfitting					

TABLE 3: Challenges faced by evaluated state-of MTSC methods. 'C' marks the challenge this metl not yet address well.