AutoGluon.TimeSeries v1.4

Installation

AutoGluon (GitHub) supports Python 3.9 to 3.12 and is available for Linux, MacOS, and Windows. The fastest way to install AutoGluon is through the uv package manager.

```
# Install uv package installer (faster than pip)
python -m pip install -U uv
# Install AutoGluon
python -m uv pip install autogluon
```

Preparing Data

AutoGluon can generate forecasts for datasets consisting of **multiple univariate** time series. Here we use the M4 Competition Daily dataset to demonstrate how to do forecasting with AutoGluon.

```
import pandas as pd
raw_data = pd.read_csv("m4_daily.csv")
raw_data.head()
```

	item_id	timestamp	target	weekend
0	D1737	1995-05-23	1900.0	0.0
1	D1737	1995-05-24	1877.0	0.0
2	D1737	1995-05-25	1873.0	0.0
3	D1737	1995-05-26	1859.0	0.0
4	D1737	1995-05-27	1876.0	1.0

Each row contains unique ID of each time series, timestamp, value of the time series, and (optional) time-varying **covariates**.

A time series datasets may also optionally include time-independent **static features** (metadata) for each time series.

```
static_features = pd.read_csv("m4_metadata.csv")
static_features.head()
```

	item_id	domain
0	D1737	Industry
1	D1843	Industry
2	D2246	Finance
3	D909	Micro
4	D1345	Micro

We convert raw data into a **TimeSeriesDataFrame** used by AutoGluon.

```
from autogluon.timeseries import TimeSeriesDataFrame
train_data = TimeSeriesDataFrame.from_data_frame(
    raw_data,
    id_column="item_id",
    timestamp_column="timestamp",
    static_features_df=static_features,  # optional
)
```

Training

Train models to forecast the values in the column 'target' 30 steps into the future for each time series.

```
from autogluon.timeseries import TimeSeriesPredictor
predictor = TimeSeriesPredictor(
    target="target",
    prediction_length=30,
).fit(train_data, presets="medium_quality")
```

More options to construct a **TimeSeriesPredictor** instance (docs):

```
# The metric used to tune models
eval_metric="MASE"
# Select quantiles for the probabilistic forecast
quantile_levels = [0.1, 0.5, 0.9]
# If data has irregular timestamps, provide frequency
freq="D"
# Covariates that are known in the future
# (e.g., holidays, promotions, weather forecasts)
known_covariates_names=["weekend"]
```

More options for the **fit** method (docs):

```
# Limit the training time, in seconds
time_limit=600
# More accurate forecasts but longer training time
presets="best_quality"
# Backtest using multiple validation windows
num_val_windows=3
# Manually select what models to use,
# e.g., only use ETS and Chronos-Bolt (Base)
hyperparameters={
    "ETS": {"seasonal_period": 14},
    "Chronos": {"model_path": "bolt_base"},
}
# Ignore some models
excluded_model_types=["AutoARIMA", "PatchTST"]
```

Monitoring

Understand the contribution of each model.

```
predictor.leaderboard()
```

Validation Score | Inference Time | Fitting Time

	model	score_val	pred_time_val	fit_time_marginal
Ensemble Model	0 WeightedEnsemble	-0.032114	20.412614	0.929712
	1 Chronos[bolt_small]	-0.032269	0.577893	0.122052
	2 TemporalFusionTransformer	-0.032757	0.152326	88.964152
	3 RecursiveTabular	-0.033355	0.920905	14.948713
Individual Models	4 Theta	-0.034366	19.682395	0.221477
	5 Naive	-0.034372	2.265100	0.186901
	6 ETS	-0.034412	6.784321	0.236519
	7 SeasonalNaive	-0.037030	0.146423	0.183321
	8 DirectTabular	-0.038809	0.769706	65.110591

Predicting

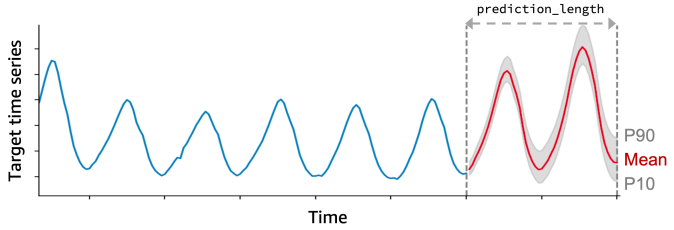
Forecast `prediction_length` steps into the future starting from the end of each time series in `train_data`.

```
predictions = predictor.predict(
    train_data,
    # only necessary if known_covariates_names
    # were provided when creating predictor
    known_covariates=known_covariates,
)
known_covariates.head()
```

	item_id	timestamp	weekend
0	D1737	1997-05-28	0.0
1	D1737	1997-05-29	0.0
2	D1737	1997-05-30	0.0
3	D1737	1997-05-31	1.0
4	D1737	1997-06-01	1.0

AutoGluon generated probabilistic forecasts that include

- mean forecast — expected value of the time series
- quantile forecast — range of possible outcomes




```
predictions.head()
```

		mean	0.1	0.5	0.9
item_id	timestamp				
D1737	1997-05-28	1575.57	1549.26	1576.73	1607.51
	1997-05-29	1575.77	1538.69	1573.41	1612.71
	1997-05-30	1573.44	1524.77	1570.95	1618.38
	1997-05-31	1573.06	1523.11	1562.97	1610.89
	1997-06-01	1573.77	1521.43	1568.05	1625.90

AutoGluon predicts with the final ensemble model. You can also predict using an individual model.

```
models = predictor.model_names()
predictor.predict(test_data, model=models[1])
```

- Detailed time series tutorials.
- For other types of data, check TabularPredictor for tabular data and MultiModalPredictor for multi-modal data such as images and text.
- Check the latest version of this cheat sheet.
- Any questions? Ask here
- Like what you see? Consider starring AutoGluon on GitHub and following us on X (Twitter) to get notified of the latest updates!