AutoGluon.TimeSeries v1.0

Installation

AutoGluon (GitHub) requires pip > 1.4 (upgrade by pip install -U pip). More installation options. AutoGluon supports Python 3.8 to 3.11. Installation is available for Linux, MacOS, and Windows.

```
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 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)
```

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
# Ignore some models
excluded_model_types=["AutoARIMA", "PatchTST"]
# Manually select what models to train.
# E.g., only train ETS with seasonal_period=14
# and DeepAR with default hyperparameters
hyperparameters={
    "ETS": {"seasonal_period": 14},
    "DeepAR": {},
}
```

Monitoring

Understand the contribution of each model.

```
predictor.leaderboard()
```

Validation score | Inference time | Training time

Combined ensemble model

Individual model

	model	score_val	pred_time_val	fit_time_marginal
0	WeightedEnsemble	-0.845	2.101	0.623
1	RecursiveTabular	-0.875	0.356	8.768
2	SeasonalNaive	-1.107	0.100	0.066
3	DirectTabular	-1.671	0.223	3.754
4	Theta	-2.267	1.422	0.070
5	ETS	-2.304	29.722	0.066

Predicting

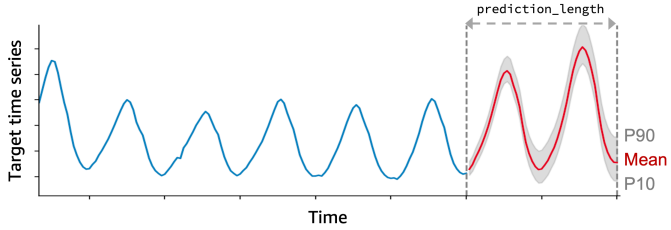
Forecast `prediction_length` steps into the future starting from the end of each time series in `train_data`.

```
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 Twitter to get notified of the latest updates!