

AutoGluon v0.8 Cheat Sheet

Installation

[AutoGluon](#) ([GitHub](#)) requires `pip > 1.4` (upgrade by `pip install -U pip`). [More installation options](#). AutoGluon v0.7 supports Python 3.8 to 3.10. Installation is available for Linux, MacOS, and Windows.

```
pip install autogluon
```

Preparing Data

AutoGluon accepts DataFrames as inputs, where each row stores an example, while a column presents a feature. Here we use the [Kaggle Titanic](#) dataset to demonstrate how to use AutoGluon.

```
import pandas as pd
train_data = pd.read_csv('titanic/train.csv')

from autogluon.tabular import TabularDataset
train_data = TabularDataset('titanic/train.csv')
# it's also a Pandas DataFrame but with additional methods
```

Little data preprocessing, such as removing obvious non-predictive columns, is needed for AutoGluon.

```
train_data = train_data.drop(columns=['PassengerId'])
```

Training

Train models to predict the values in the column 'Survived'. The training log will tell you how AutoGluon extracts features, selects, trains and ensembles models.

```
from autogluon.tabular import TabularPredictor

predictor =
TabularPredictor(label='Survived').fit(train_data)
```

More options to construct a `TabularPredictor` instance ([docs](#)):

```
verbosity=3 # More training log.
# The metric used to tune models. All available metrics.
eval_metric='roc_auc'
```

More options for the `fit` method ([docs](#), [presets](#)):

```
# Limit the training time, in second
time_limit=600
# Better model ensemble for a better accuracy, but
longer training time. All available options.
presets='best_quality'
# Use a separate dataset to tune models.
tuning_data=val_data
# Explore less models. You can fully control the model
search space. All available options.
hyperparameters='very_light'
# Ignore some models.
excluded_model_types=['KNN', 'NN_TORCH']
```

Monitoring

Understand the contribution of each model ([docs](#)).

```
predictor.leaderboard()
```

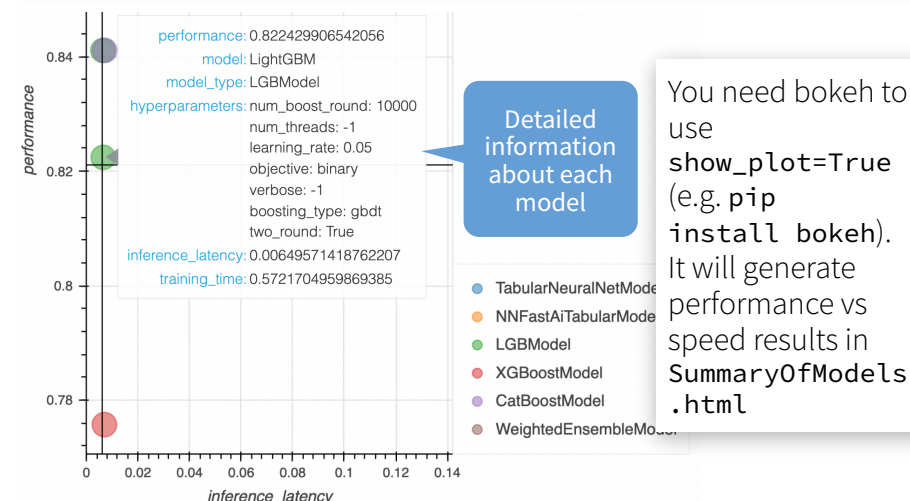
		model	score_val	pred_time_val	fit_time
The model ensembles all	0	WeightedEnsemble_L2	0.869159	0.171661	13.350093
	1	LightGBMXt	0.841121	0.006379	0.439488
Individual model	2	CatBoost	0.841121	0.007126	0.308449
	3	NeuralNet	0.841121	0.007126	0.308449

More options for `leaderboard`:

```
silent=True # Recommend when using Jupyter.
# Report metrics on a separate test dataset.
data=test_data
# Evaluate more metrics.
extra_metrics=['accuracy', 'log_loss']
```

Understand more about the trained models ([docs](#)).

```
predictor.fit_summary(show_plot=True)
```



Understand the importance of each feature ([docs](#)).

```
predictor.feature_importance(test_data)
```

	importance	stddev	p_value
Name	0.204744	0.020024	0.001587
Pclass	0.041199	0.011695	0.012912
Sex	0.039950	0.010314	0.010751
Age	0.024969	0.004325	0.004926
SurvSp	0.015854	0.001873	0.002745

Predicting

```
test_data = TabularDataset('test.csv')
# Predict for each row
predictor.predict(test_data)
# Return the class probabilities for classification
predictor.predict_proba(test_data)
# Evaluate various metrics, it needs test_data to have
the label column
predictor.evaluate(test_data)
```

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

```
# Get a list of string names
models = predictor.get_model_names()
# Predict with the 2nd model. Both predict_proba and
evaluate also accept the model argument
predictor.predict(test_data, model=models[1])
```

Deploying

AutoGluon models are saved to disk automatically. You can check log to find where it saves, or get the path by `predictor.path`.

```
# Load saved model from disk.
predictor = TabularPredictor.load('AutogluonModels/
ag-20220129_004130/')
```

If the inference speed matters, [there are multiple ways to accelerate the speed](#). First, you can force all models in memory.

```
predictor.persist_models()
```

During training, you can use presets for the `fit` method optimized for fast inference (though may hurt model performance).

```
presets=['good_quality', 'optimize_for_deployment']
```

Alternatively, you can distill the ensemble into a single model.

```
# Get the list of names of the distilled models.
students = predictor.distill()
# Evaluate the 3rd distilled model.
predictor.evaluate(test_data, model=students[2])
```

Results on Titanic. Accuracy 83.8% → 84.5%, evaluation time 82ms → 49ms. Here the distilled model even has a better accuracy.

- Click [here](#) for detailed Tabular tutorials.
- For data involving text and images, try out [MultiModalPredictor](#).
- Check the latest version of this cheat sheet at <https://auto.gluon.ai/stable/cheatsheet.html>
- 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!