
AWS
re:Invent
NOV. 28 – DEC. 2, 2022 | LAS VEGAS, NV

BOA402

Using AutoML to develop deep learning solutions automatically

Cyrus Vahid

Principal Specialist ML DA
AWS

Michael Dasseville

Research Scientist – Transportation
Amazon

What is the workshop about?

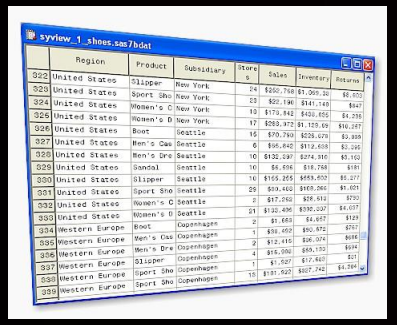
- AutoGluon capabilities
- Development steps
- Tabular data tutorial
- AutoGluon under the hood
- Amazon SageMaker Autopilot
- 5 additional tutorial code examples for self-learning

What is AutoGluon?

AutoGluon

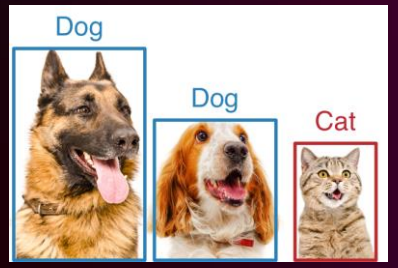
- AutoGluon is an AutoML library capable of:
  - Data preprocessing
  - Hyperparameter tuning
  - Model ensembling
- Using built-in and custom models, which enable users to:
  - Quick ML prototyping
  - Achieving state-of-the-art results with limited ML knowledge
  - Simplifying ML training pipeline through methods such as HPO

Domains



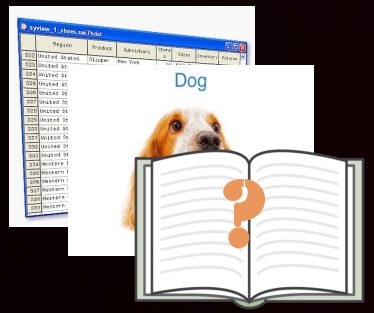
Tabular prediction

Image prediction



Object detection

Text prediction



Bimodal prediction

Timeseries prediction

How does it work?

Methods

- Using pretrained models from a model zoo
- Fine-tuning individual pretrained models based on the provided dataset
- Performing HPO (optional) per model
- Ensemble models for best results

Model zoo

Tabular predictor models

`autogluon.tabular.models`

Model	Description
LGBModel	LightGBM model: https://lightgbm.readthedocs.io/en/latest/
CatBoostModel	CatBoost model: https://catboost.ai/
XGBoostModel	XGBoost model: https://xgboost.readthedocs.io/en/latest/
RFModel	Random forest model (scikit-learn): https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
XTModel	Extra-trees model (scikit-learn): https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html#sklearn.ensemble.ExtraTreesClassifier
KNNModel	K-nearest neighbors model (scikit-learn): https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
LinearModel	Linear model (scikit-learn): https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
TabularNeuralNetTorchModel	PyTorch neural network models for classification/regression with tabular data
TabularNeuralNetMxnetModel	Class for neural network models that operate on tabular data
NNFastAiTabularModel	Class for fastai v1 neural network models that operate on tabular data
VowpalWabbitModel	Vowpal Wabbit model: https://vowpalwabbit.org/

Text predictor models

- Text predictor is an NLP tool based on transformer-based models
- AutoGluon TextPredictor predicts values in a column of a tabular dataset that contains text fields
- TextPredictor focuses on fine-tuning deep learning based models; it supports transfer learning from pretrained NLP models like BERT, ALBERT, and ELECTRA

Image predictor models

Several models are supported that can be listed using `autogluon.vision.ImagePredictor.list_models()`

```
models = ImagePredictor.list_models()
(len(models), models)
```

```
(739,
 ('adv_inception_v3',
  'bat_resnext26ts',
  'beit_base_patch16_224',
  'beit_base_patch16_224_in22k',
  'beit_base_patch16_384',
  'beit_large_patch16_224',
  'beit_large_patch16_224_in22k',
  'beit_large_patch16_384',
  'beit_large_patch16_512',
  'botnet26t_256',
  'botnet50ts_256',
  'cait_m36_384',
  'cait_m48_448',
  'cait_s24_224',
  'cait_s24_384',
  'cait_s36_384',
```

Object detection

- Several object detection models are included in AutoGluon, including Yolo3, FasterRCNN, and SSD
- As of version 0.4.0, autogluon.ObjectDetector is being deprecated and is being reworked with torch backend – Current version: 0.5.2

Time series

`autogluon.tabular.models`

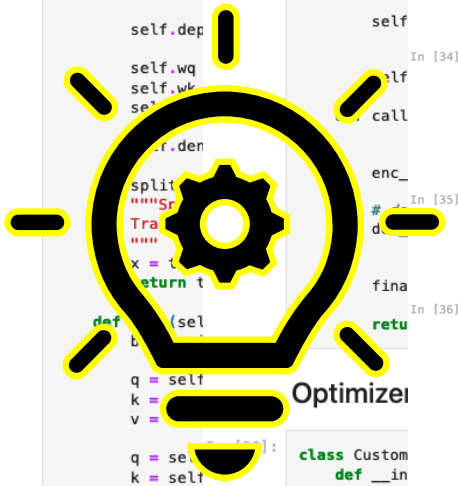
- AutoGluon uses gluonTS for most neural algorithms (some models listed below); for a comprehensive list, refer to https://ts.gluon.ai/stable/getting_started/models.html
- Simpler time series models, such as exponential smoothing or ARIMA, are supported using https://www.sktime.org/en/stable/

Model	L/G	Data layout	Architecture	Reference
DeepAR	Global	Univariate	RNN	Salinas et al. 2020
DeepState	Global	Univariate	RNN, state-space model	Rangapuram et al. 2018
DeepFactor	Global	Univariate	RNN, state-space model, Gaussian process	Wang et al. 2019
Deep Renewal Processes	Global	Univariate	RNN	Türkmen et al. 2021
GPForecaster	Global	Univariate	MLP, Gaussian process	
MQ-CNN	Global	Univariate	CNN encoder, MLP decoder	Wen et al. 2017
MQ-RNN	Global	Univariate	RNN encoder, MLP encoder	Wen et al. 2017
N-BEATS	Global	Univariate	MLP, residual links	Oreshkin et al. 2019

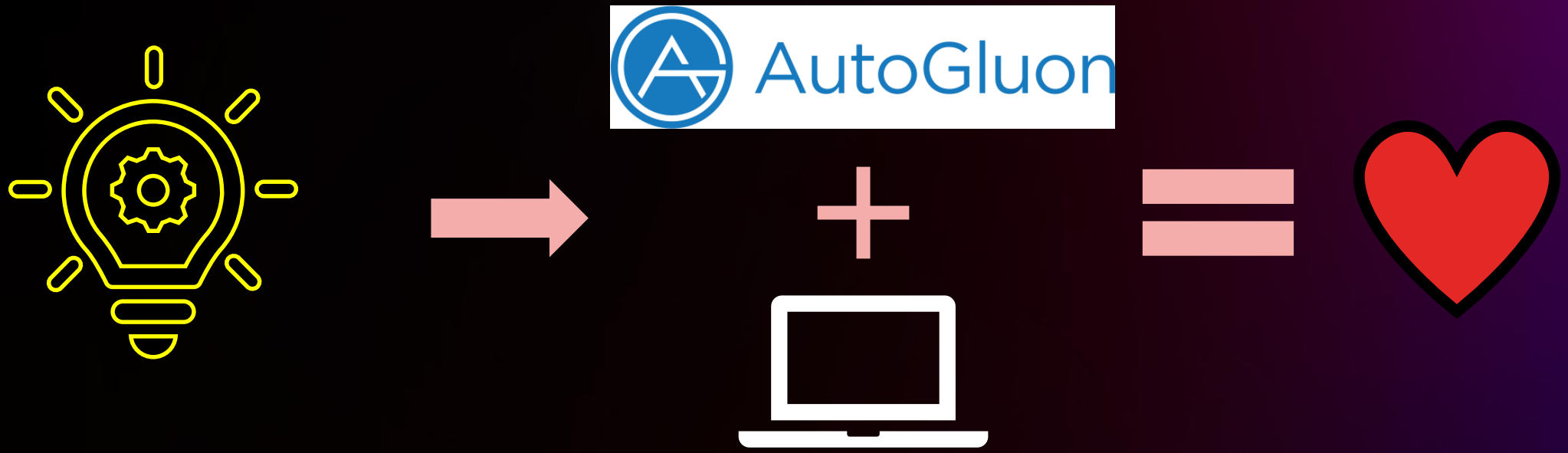
AutoGluon in practice

- Code template
- Tutorial

What is your current ML workflow?



Save some time



Solution from AutoGluon

1. Imports
   - Integration with pandas and NumPy

```
import autogluon
import autogluon.core as ag
from autogluon.vision import ImageDataset, ImagePredictor
```

2. Data loading
   - Automatic loading from different sources
   - Train/validation/test split

```
data_loc = 'https://autogluon.s3.amazonaws.com/shopee.zip'
train_data, _, test_data = ImageDataset.from_folders(data_loc)
```

3. Initializing predictor and fitting
   - Default values ➔ just specify the data!
   - Full control via hyperparameters
   - Model's complexity control

```
param={'model': model, 'batch_size': batch_size}
predictor = ImagePredictor()
predictor.fit(train_data, time_limit=60,
              hyperparameters=param)
```

4. Testing
   - By default metrics
   - Setting metrics for your application

```
predictor.evaluate(test_data)
```

5. Prediction

```
predictor.predict(new_data)
```

Examples for different datasets/problems

Tabular data/classification

- ✓ Data: Tabular
- ✓ Problem: Classification
- ✓ Minimum code:
  - ✓ Tabular-specific imports
  - ✓ Ex. Data loaded from separate files
- ✓ **Several models are tested automatically**

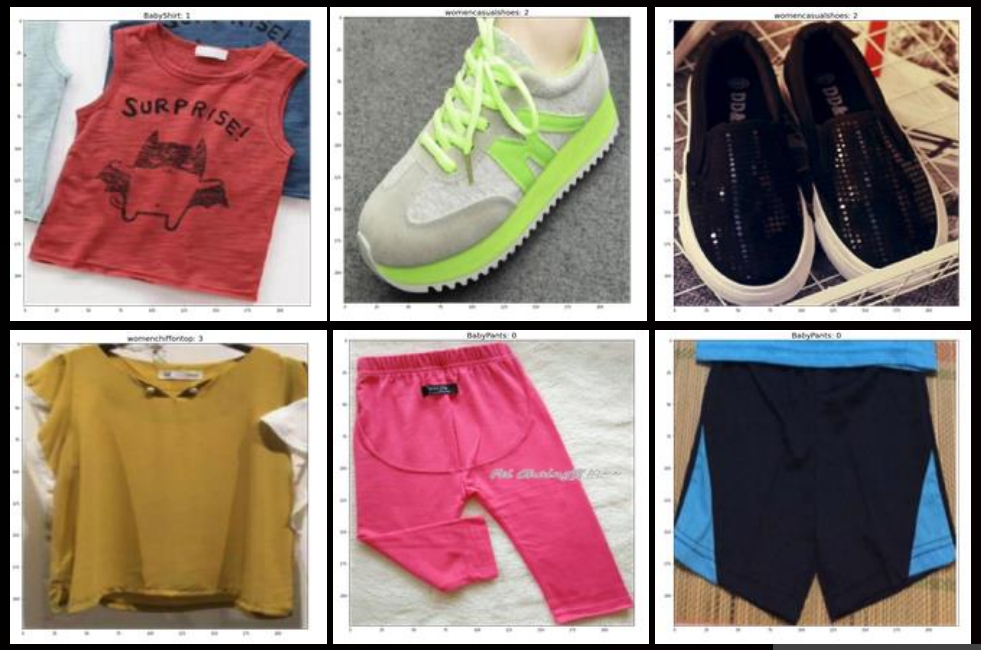
education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	class
Some-college	10	Married-civ-spouse	Exec-managerial	Wife	White	Female	0	0	40	United-States	>50K
10th	6	Married-civ-spouse	Other-service	Wife	White	Female	0	0	8	United-States	<=50K
Some-college	10	Married-civ-spouse	Craft-repair	Husband	White	Male	0	0	44	United-States	<=50K

accuracy: 0.8374

1. Imports
2. Data loading
3. Initializing and fitting
4. Testing
5. Prediction

```
from autogluon.tabular import TabularDataset, TabularPredictor
train_data = TabularDataset(pd.read_csv('tabular_train.csv'))
test_data = TabularDataset(pd.read_csv('tabular_test.csv'))
predictor = TabularPredictor(label='class',sample_weight='class')
predictor.fit(train_data)
predictor.evaluate(test_data, auxiliary_metrics=False)
predictor.predict(new_data)
```

Image data/classification



accuracy: 0.8625

- ✓ Data: Images
- ✓ Problem: Classification
- ✓ Minimum code:
  - ✓ Reading from archive
  - ✓ Automatic train/test split
  - ✓ Default model Resnet50
  - ✓ You can set a list of models
  - ✓ AutoGluon selects the best model

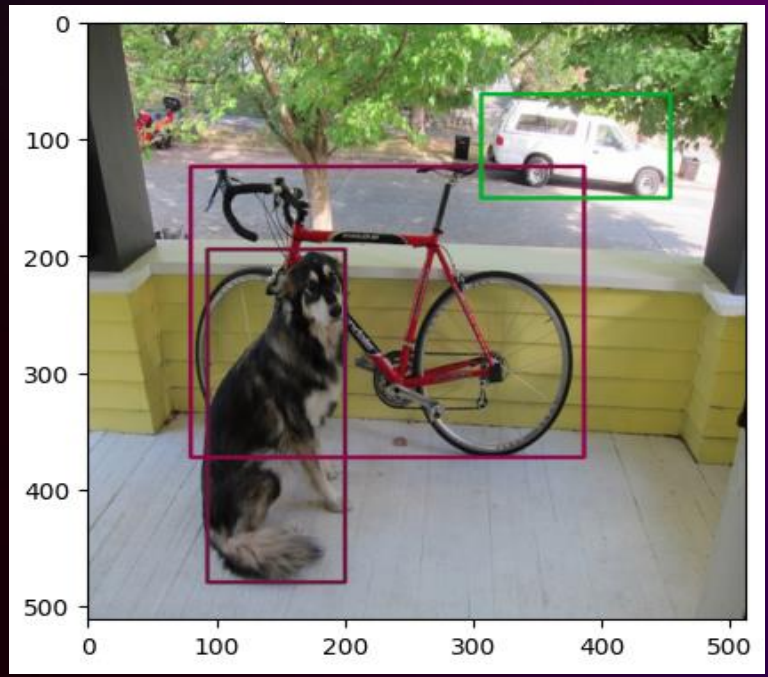
1. Imports
2. Data loading
3. Initializing and fitting
4. Testing
5. Prediction

```
from autogluon.vision import ImageDataset, ImagePredictor
data_location = 'https://autogluon.s3.amazonaws.com/shopee-iet.zip'
train_data, _, test_data = ImageDataset.from_folders(data_location)
models = ag.Categorical('resnet18_v1b', 'mobilenetv3_small', 'vgg19')
hyperparameters = {'model': models}
predictor = ImagePredictor().fit(train_data,
                                 hyperparameters=hyperparameters)
predictor.evaluate(test_data)
predictor.predict(new_data)
```

aws

Image data/object detection

- ✓ Data: Images
- ✓ Problem: Object detection
- ✓ Minimum code:
  - ✓ Physical location of train/test specified



1. Imports
2. Data loading
3. Initializing and fitting
4. Testing
5. Prediction

```
from autogluon.vision import ObjectDetector
data_loc = 'https://autogluon.s3.amazonaws.com/data/tiny_motorbike.zip'
train_data = ObjectDetector.Dataset.from_voc(data_loc, splits='trainval')
test_data = ObjectDetector.Dataset.from_voc(data_loc, splits='test')
detector = ObjectDetector().fit(train_data, hyperparameter={})
detector.evaluate(test_data)
detector.predict(new_data)
```

Text data/sentiment analysis

- ✓ Data: Text
- ✓ Problem: Sentiment analysis
- ✓ Minimum code:
  - ✓ Evaluation metric specified
  - ✓ Default model electra-base
  - ✓ You can set any model from Hugging model zoo
- ✓ **Good accuracy for a difficult problem**

accuracy: 0.9404

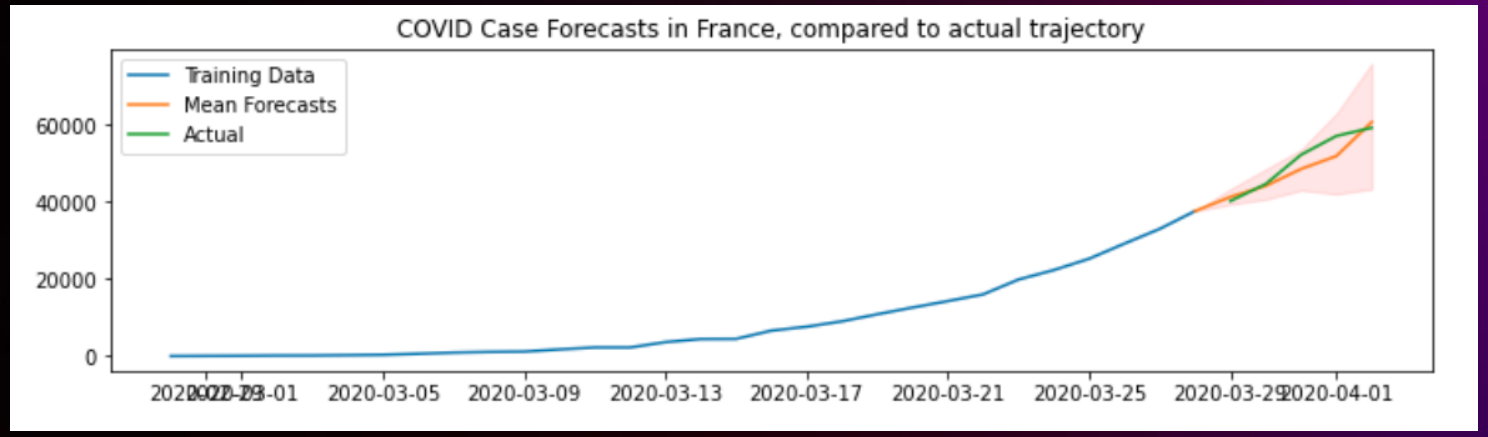
sentence	label
goes to absurd lengths	☹
saw how bad this movie was	☹
the greatest musicians	☺
cold movie	☹
redundant concept	☹
a smile on your face	☺

1. Imports
2. Data loading
3. Initializing and fitting
4. Testing
5. Prediction

```
from autogluon.text import TextPredictor
from autogluon.core.utils.loaders import load_pd
train_data = load_pd.load('text_train.zip')
test_data =load_pd.load('text_test.zip')
predictor = TextPredictor(label='label', eval_metric='acc').fit(train_data)
predictor.evaluate(test_data)
predictor.predict(new_data)
```

Time series/forecasting

- ✓ Data: Time series
- ✓ Problem: Forecasting
- ✓ Minimum code:
  - ✓ Future/past data split
  - ✓ Train different models by default



1. Imports
2. Data loading
3. Initializing and fitting
4. Testing
5. Prediction

```
from autogluon.timeseries import TimeSeriesPredictor, TimeSeriesDataFrame
data_loc_df = pd.read_csv("Covid/train.csv")
data = TimeSeriesDataFrame.from_data_frame(data_loc_df, timestamp_column="Date")
test_data = train_data.copy()
train_data = train_data.slice_by_timestep(slice(None, -5))
predictor = TimeSeriesPredictor(target="Cases", eval_metric="MAPE",
                                prediction_length=5)
predictor.fit(train_data=train_data, presets="low_quality")
predictor.evaluate(test_data)
predictor.predict(new_data)
```

Multimodal data/classification

- ✓ Data: all
  - ✓ Numerical
  - ✓ Categorical
  - ✓ Text
  - ✓ Images
  - ✓ Time series
- ✓ Problem: Classification
- ✓ Minimum code

accuracy: 0.69

	Name	Age	Breed1	Breed2	Gender	Color1	Description	AdoptionSpeed
0	Yumi Hamasaki	4	292	265	2	1	I rescued Yumi Hamasaki at a food stall far aw...	0
1	Nene/ Kimie	12	285	0	2	5	Has adopted by a friend with new pet name Kimie	0
2	Mattie	12	266	0	2	1	I rescued Mattie with a broken leg. After surg...	0
3	NaN	1	189	307	2	1	She born on 30 September . I really hope the a...	0
4	Coco	6	276	285	2	2	Calico Tame and easy going Diet RC Kitten Supp...	0

1. Imports
2. Data loading
3. Initializing and fitting
4. Testing
5. Prediction

```
from autogluon.multimodal import MultiModalPredictor
data_loc = './data'
train_data = pd.read_csv(f'{dataset_path}/train.csv', index_col=0)
test_data = pd.read_csv(f'{dataset_path}/test.csv', index_col=0)
predictor = MultiModalPredictor(label='class').fit(train_data)
predictor.evaluate(test_data, metrics=["roc_auc",'accuracy','f1'])
predictor.predict(new_data)
```

Choosing the best model

- Performance comparison
  - Mode
    - By default for tabular
    - On request for others
  - AutoGluon automatically maximizes the chosen or default metric
    - Accuracy for classification
    - RMSE for regression
  - Other metrics of interest
    - Training time
    - Execution time
- Saving/loading models

	model	score_test	score_val	pred_time_test	pred_time_val	fit_time	stack_level	can_infer	fit_order
0	RandomForestMSE	-0.031746	-0.023671	0.054240	0.033716	1.244549	1	True	5
1	CatBoost	-0.032491	-0.022593	0.022067	0.005191	1.839485	1	True	6
2	WeightedEnsemble_L2	-0.032845	-0.021535	0.071991	0.035560	1.190444	2	True	12
3	ExtraTreesMSE	-0.032959	-0.022488	0.076029	0.033569	0.386660	1	True	7
4	LightGBMLarge	-0.033156	-0.022549	0.021212	0.007871	0.286389	1	True	11
5	LightGBMXT	-0.033362	-0.022982	0.015830	0.005053	4.372931	1	True	3

```
predictor = TabularPredictor.load(save_path)
y_pred = predictor.predict(new_data)
```

Run feature importance

Feature importance gives you a better understanding of the predictor and the data using permutation-shifting

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

```
Computing feature importance via permutation shuffling for 14 features using 5000 rows with 5 shuffle sets...
        4.23s   = Expected runtime (0.85s per shuffle set)
        4.23s   = Actual runtime (Completed 5 of 5 shuffle sets)
```

	importance	stddev	p_value	n	p99_high	p99_low
marital-status	0.04992	0.002995	1.548031e-06	5	0.056087	0.043753
education-num	0.03192	0.002161	2.507753e-06	5	0.036371	0.027469
capital-gain	0.03052	0.002062	2.485369e-06	5	0.034766	0.026274
age	0.01280	0.003718	7.656432e-04	5	0.020454	0.005146
hours-per-week	0.01152	0.004582	2.460466e-03	5	0.020954	0.002086
occupation	0.00476	0.001711	1.700352e-03	5	0.008283	0.001237
relationship	0.00344	0.000167	6.697221e-07	5	0.003785	0.003095
workclass	0.00184	0.001596	3.074391e-02	5	0.005127	-0.001447

Tutorial

Tutorial structure

1. Tabular (40 min)
   - Demo: classification
   - Exercise: regression

2. Image classification (20 min)
   - Demo
   - Exercise

3. Time series forecasting (20 min)
   - Demo
   - Exercise

- Timing is approximative
- Goal is to finish the tabular exercise
- Solutions will be shown at the end of each section

Tutorial

- Open a notebook in your AWS SageMaker
- Git clone the tutorial
- URL: **bit.ly/BOA402**
- git clone https://github.com/MichaelDasseville/Autogluon-Workshop.git

Event hash to fill:
bcf2-152b2cc1b4-85

AutoGluon tabular data under the hood

Architecture

- AutoGluon has a modular architecture; you install and include only the features your need
  - *autogluon.tabular*: `from autogluon.tabular import TabularDataset, TabularPredictor`
  - *autogluon.multimodal*: `from autogluon.multimodal import MultiModalPredictor`
  - *autogluon.vision:* `from autogluon.vision import ImagePredictor, ImageDataset`
  - *autogluon.core*: `import autogluon.core as ag`
  - *autogluon.features:* `from autogluon.features.generators import AutoMLPipelineFeatureGenerator`

Tabular model features

- Advanced data processing
- Deep learning
- Multi-layer model ensembling
- Automatic recognition the data type in each column

AutoGluon tabular principles

- Simplicity
- Robustness
- Fault tolerance
- Predictable timing

The `fit` API basic process

```
from autogluon import TabularPrediction as task
predictor = task.fit("train.csv", label="class")
predictions = predictor.predict("test.csv")
```

The `fit` API optional features

- `hyperparameter tune = True optimizes` the hyperparameters of the individual models
- `auto stack = True` adaptively chooses a model ensembling strategy based on bootstrap aggregation (bagging) and (multi-layer) stacking
- `time limits` controls the runtime of `fit()`
- `eval metric` specifies the metric used to evaluate
- predictive performance
- `auto stack = True` allows resumption of training from the last checkpoint

Data preprocessing

Model agnostic
(Transforms input to all models)

- Categorizing each feature
  - numeric
  - categorical
  - Text → *n*-grams
  - Date/time → numeric
- Features like userID with little prediction are discarded

Model specific (Applied to data to be used to train each model)

- Further processing
  - Feed forward ANN
  - LightGBM boosted trees
  - CatBoost boosted trees
  - Random forest
  - Extremely randomized trees
  - *k*-nearest neighbours

Data preprocessing

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	class
6118	51	Private	39264	Some-college	10	Married-civ-spouse	Exec-managerial	Wife	White	Female	0	0	40	United-States	>50K
23204	58	Private	51662	10th	6	Married-civ-spouse	Other-service	Wife	White	Female	0	0	8	United-States	<=50K
29590	40	Private	326310	Some-college	10	Married-civ-spouse	Craft-repair	Husband	White	Male	0	0	44	United-States	<=50K
18116	37	Private	222450	HS-grad	9	Never-married	Sales	Not-in-family	White	Male	0	2339	40	El-Salvador	<=50K
33964	62	Private	109190	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	15024	0	40	United-States	>50K

```
print(train_data['workclass'].unique(), traindata_post['workclass'].unique())
```

```
[' Private' ' Local-gov' ' Self-emp-not-inc' ' ?' ' Self-emp-inc'
 ' Federal-gov' ' State-gov'] [3]
Categories (7, int64): [0, 1, 2, 3, 4, 5, 6]
```

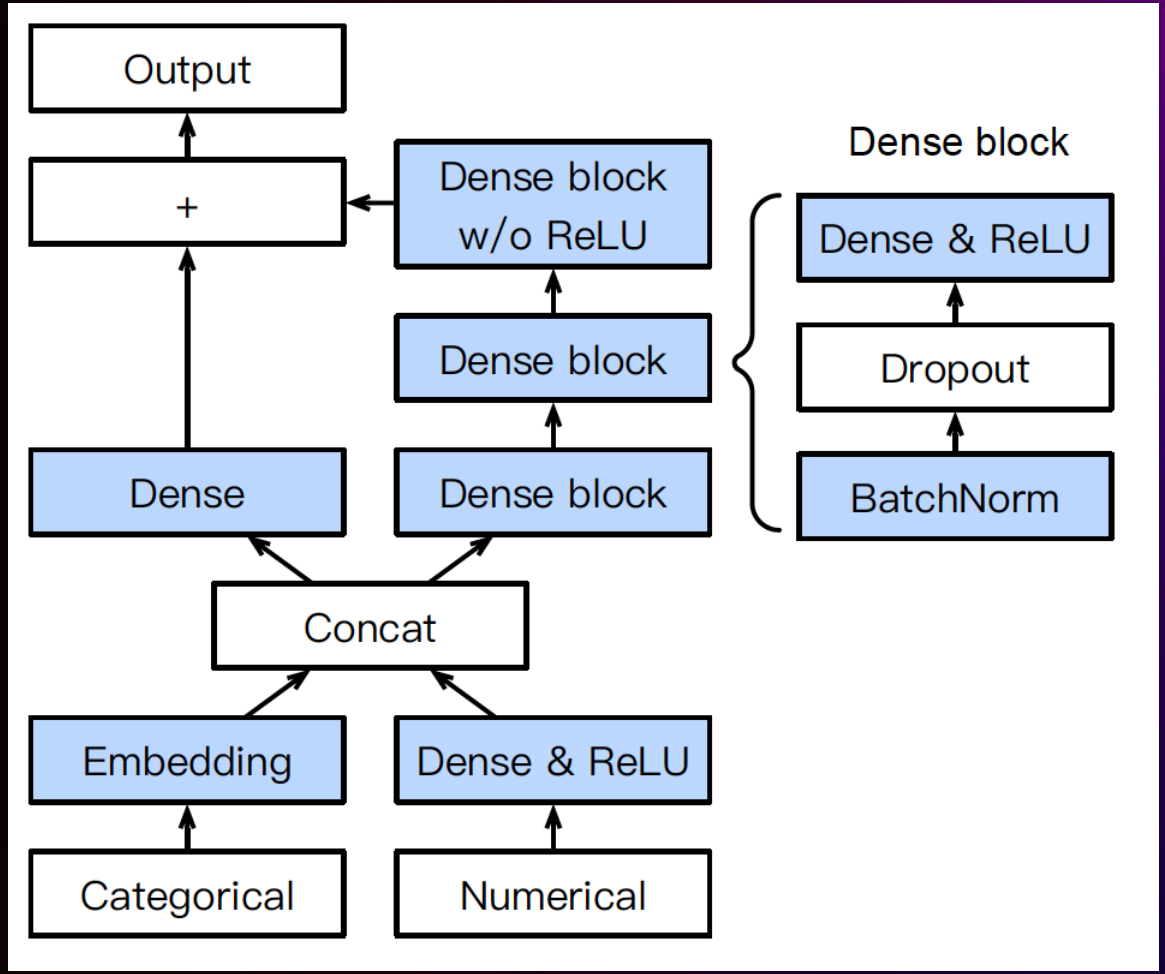
```
train_data['class'].unique()
```

```
array([' >50K', ' <=50K'], dtype=object)
```

	age	fnlwgt	education-num	sex	capital-gain	capital-loss	hours-per-week	workclass	education	marital-status	occupation	relationship	race	native-country
38771	25	106889	9	0	0	0	30	3	10	1	1	5	4	14
1721	67	172756	2	0	2062	0	34	3	NaN	5	6	1	4	NaN
33964	62	109190	13	1	15024	0	40	3	8	1	3	0	4	14
38864	43	149102	13	1	0	0	40	3	8	1	1	0	4	14
36616	47	368561	10	1	0	0	40	3	13	1	2	0	4	14

Feed forward ANN

- Separate embedding layer for each categorical feature → *Learning each variable separately before blending*
- Numerical features are embedded concatenated with results of embedding → *Similar to DSSN networks enriching the data*
- The combined vector passes thorough a dense layer as well as a 3L network→ *Speeding up and stabilizing the learning + reducing overfitting*

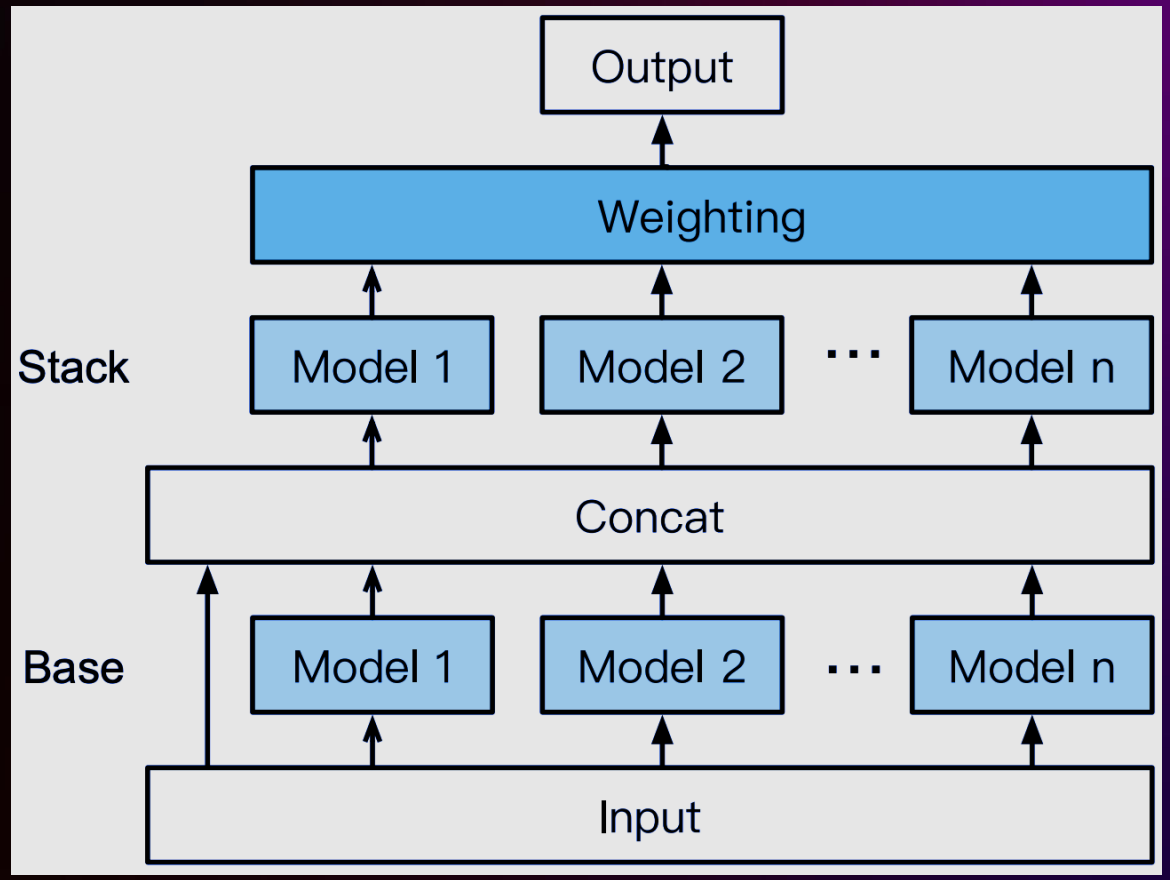


Multi-layer stack ensembling

- Stacking
- Bagging or bootstrap aggregation

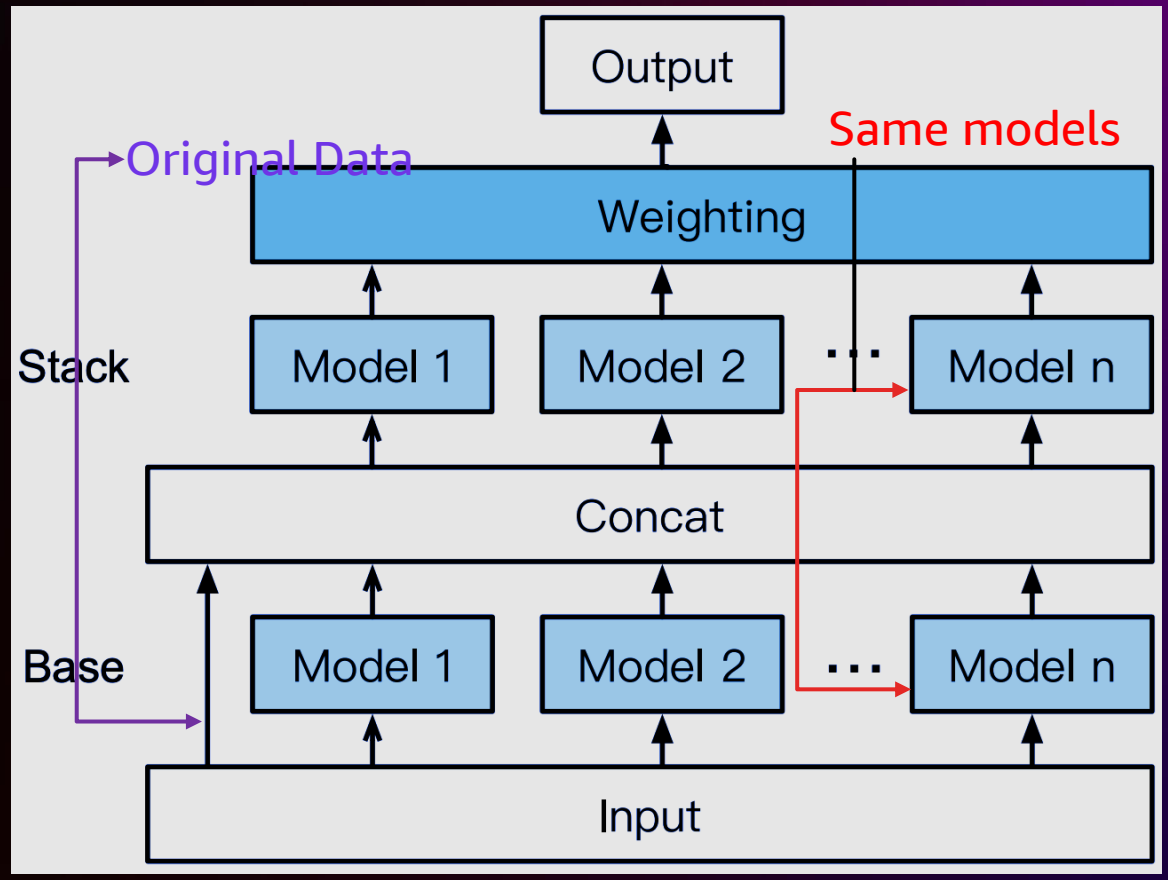
Multi-layer stacking

- A “stacker” model is trained using the aggregated predictions of the base models as its features → *Improve upon base models*
- Multi-layer stacking sends the output of stacker models to another stacker layer
- AutoGluon implements a novel multi-stacking mechanism depicted in the opposite diagram



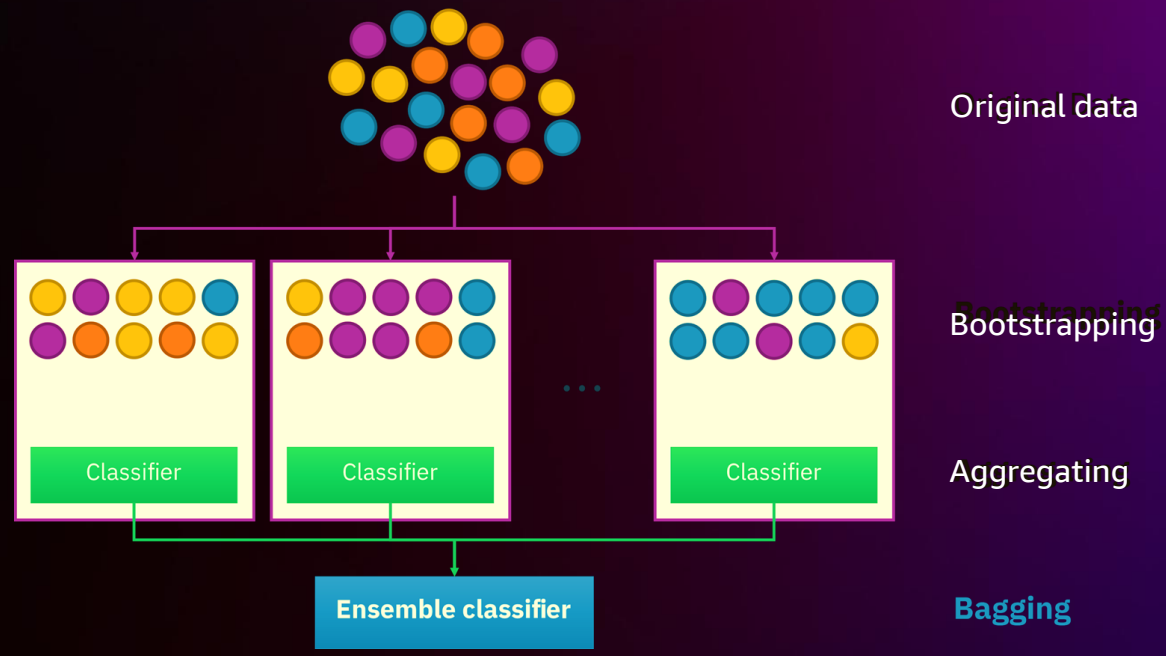
Multi-layer stacking – AutoGluon contributions

- Unlike traditional models, AutoGluon reuses all of its base-layer model types rather than adopting to use simpler models; this is similar to layer-wise training → *Efficiency*
- Using original input along with output of stackers as higher layer input; this is similar to skip layer → *Compactness*
- Using final layer for ensemble selection in a weighted manner → *Reduces overfitting*



Bagging

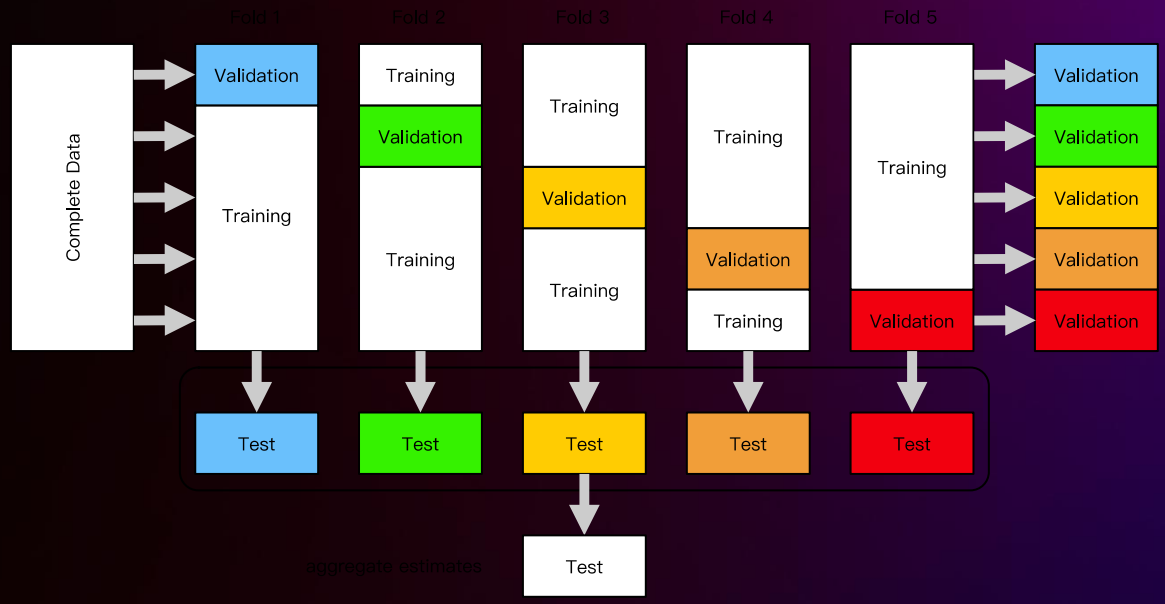
Fit several independent models and average their predictions in order to obtain a model with a lower variance
→ *Robustness, so the resulting model works better on larger collection of datasets*



https://en.wikipedia.org/wiki/Bootstrap_aggregating

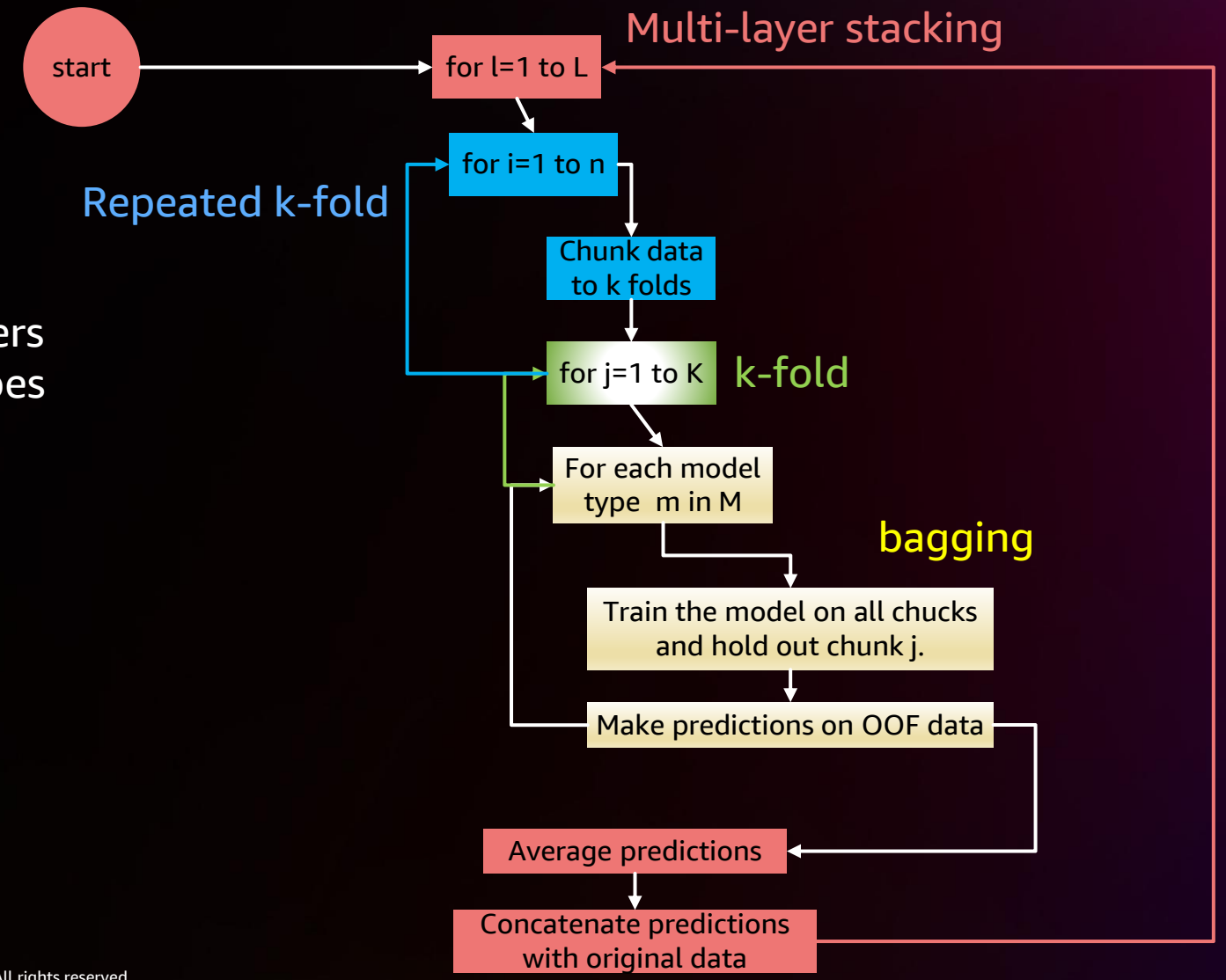
Repeated *k-fold bagging*

- Randomly partitioning data into *k* disjoint chunks
- Training *k* copies of the model with different data chunks
- AutoGluon bags all the models and makes prediction with on a held-out chunk called an out of fold (OOF)
- This results in reducing variance in resulting predictions, thus helping with ***bias correction***



Putting it all together

L: number of stacker layers
M: Set of base model types



Distillation

- Problem
  - The final model is too large or two slow and using several submodels such as tree-based and ANN-based models
- Distillation
  - A teacher (large base model) describes a function
  - A student model (smaller model) approximates the same function, meaning that we train a smaller model to produce the same result as the complex model within an acceptable error
  - Training on lots of data helps closing the gap between student and teacher
- Trade-off
  - Loss of accuracy

Distillation for text and multimodal

- `fit` method of `TextPredictor` and `MultimodalPredictor` accepts a `teacher_predictor` parameter
- `TextPredictor.fit(train_data=data, teacher_predictor=teacher_model, ...)`
- `TextPredictor` focuses on fine-tuning of transformer-based models through transfer learning rather than ensembling
- The pre-trained teacher predictor or its saved path are acceptable values; if provided, `fit` can distill its knowledge to a student predictor, i.e., the current predictor

Distillation for tabular data

The `Distill()` method of `TabularPredictor` can be explicitly called to:

- Distill AutoGluon's most accurate ensemble-predictor into single models that are simpler/faster and require less memory/compute; distillation can produce a model that is more accurate than the same model fit directly on the original training data
- After calling `Distill()`, there will be more models available in the Predictor, which can be evaluated using `predictor.leaderboard(test_data)` and are deployed with: `predictor.predict(test_data, model=MODEL_NAME)`

AutoGluon in comparison

Handling of raw data

AutoML Framework	Open	Raw	Neural Network	CASH Strategy	Model Ensembling
Auto-WEKA	√	×	sigmoid MLP	Bayesian Optimization	bag, boost, stack, vote
auto-sklearn	√	×	none	BayesOpt + Meta-Learn	ensemble selection
TPOT	√	×	none	genetic programming	stacking
H2O	√	√	MLP + Adadelta	random search	stacking + bagging
GCP-Tables	×	√	AdaNet (??)	AdaNet (??)	boosting (??)
AutoGluon	√	√	embed categorical + skip-connection	fixed defaults (set adaptively)	multi-layer stacking + repeated bagging

Competition

Framework	Wins	Losses	Failures	Champion	Avg. Rank	Avg. Rescaled Loss	Avg. Time (min)
AutoGluon	-	-	**1**	**23**	**1.8438**	**0.1385**	201
H2O AutoML	4	26	8	2	3.1250	0.2447	220
TPOT	6	27	5	5	3.3750	0.2034	235
GCP-Tables	5	20	14	4	3.7500	0.3336	**195**
auto-sklearn	6	27	6	3	3.8125	0.3197	240
Auto-WEKA	4	28	6	1	5.0938	0.8001	244

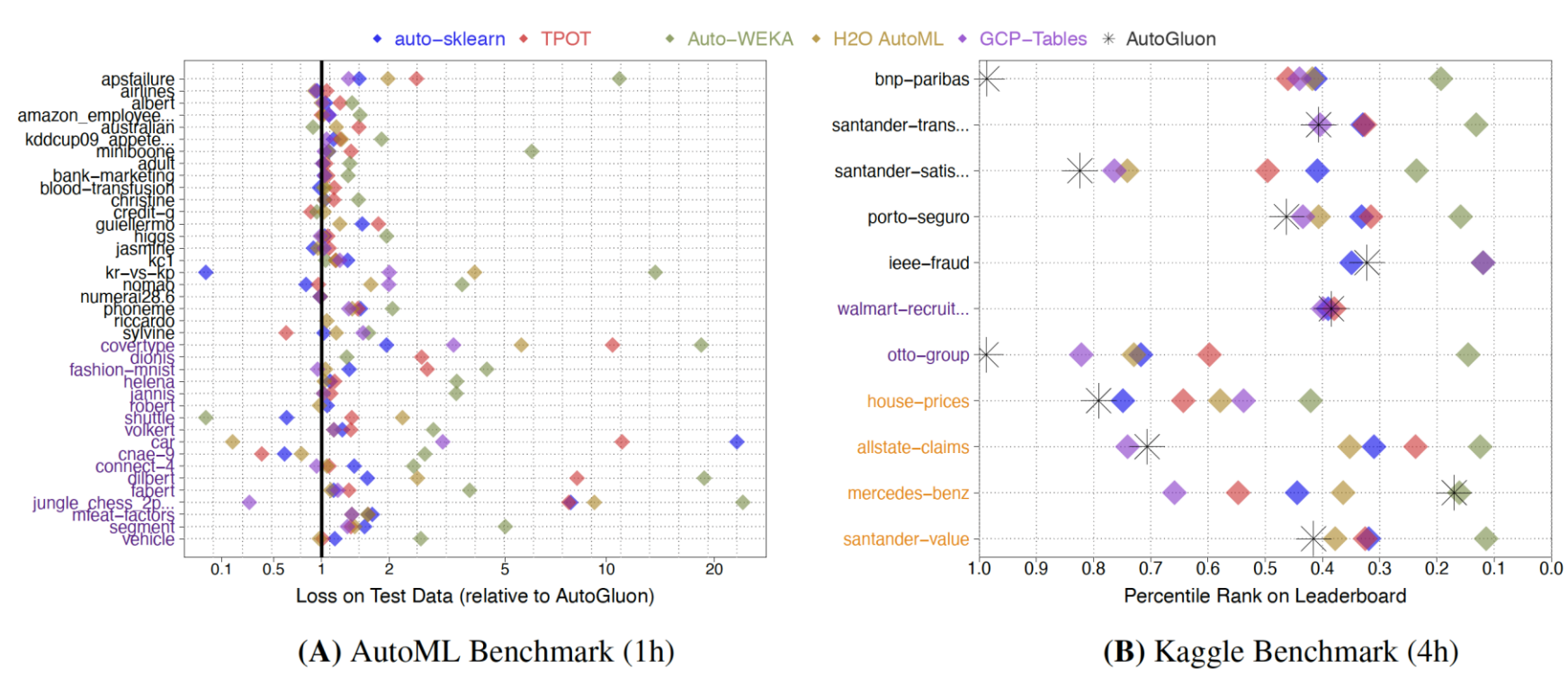
Comparing each AutoML framework against AutoGluon on the 39 AutoML Benchmark datasets (with 4h training time). Listed are the number of datasets where each framework produced better predictions than AutoGluon (Wins), worse predictions (Losses), a system failure during training (Failures), or more accurate predictions than all of the other 5 frameworks (Champion). The latter 3 columns show the average: rank of the framework (among the 6 AutoML frameworks applied to each dataset), (rescaled) loss on the test data, and actual training time.

Competition

Framework	Wins	Losses	Failures	Champion	Avg. Rank	Avg. Percentile	Avg. Time (min)
AutoGluon	-	-	**0**	**7**	**1.7143**	**0.7041**	**202**
GCP-Tables	3	7	1	3	2.2857	0.6281	222
H2O AutoML	1	7	3	0	3.4286	0.5129	227
TPOT	1	9	1	0	3.7143	0.4711	380
auto-sklearn	3	8	**0**	1	3.8571	0.4819	240
Auto-WEKA	0	10	1	0	6.0000	0.2056	221

Comparing each AutoML framework against AutoGluon on the 11 Kaggle competitions (under 4h time limit)

Benchmarking

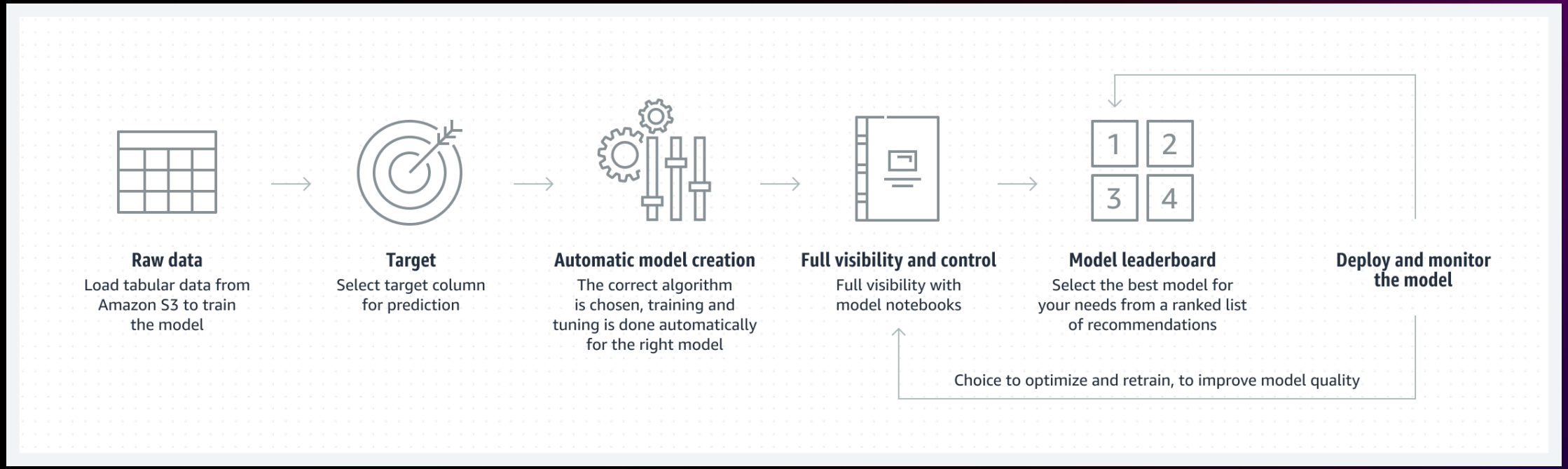


(A) Performance of AutoML frameworks relative to AutoGluon on the AutoML Benchmark (with 1h training time)

(B) Proportion of teams in each Kaggle competition whose scores were beat by each AutoML framework (with 4h training time)

AutoML and SageMaker

SageMaker Autopilot



Key features

- Automatic data preprocessing and feature engineering
- Automatic ML model selection
- Model leaderboard
- Feature importance
- Customizable AutoML journey
- Automatic notebook creation

Automatic data preprocessing

Amazon SageMaker Autopilot Data Exploration

This report provides insights about the dataset you provided as input to the AutoML job. It was automatically generated by the AutoML training job: **airline-sampled-mc-2**.

As part of the AutoML job, the input dataset was randomly split into two pieces, one for **training** and one for **validation**. The training dataset was randomly sampled, and metrics were computed for each of the columns. This notebook provides these metrics so that you can:

1. Understand how the job analyzed features to select the candidate pipelines.
2. Modify and improve the generated AutoML pipelines using knowledge that you have about the dataset.

We read `5846` rows from the training dataset. The dataset has `25` columns and the column named `DelayType` is used as the target column. This is identified as a `MulticlassClassification` problem. Here are **5** examples of labels: `['SecurityDelay', 'WeatherDelay', 'NASDelay', 'CarrierDelay', 'LateAircraftDelay']`.

Suggested Action Items

- Look for sections like this for recommended actions that you can take.

Contents

Dataset Sample

The following table is a random sample of **10** rows from the training dataset. For ease of presentation, we are only showing **20 of the 25** columns of the dataset.

Suggested Action Items

- Verify the input headers correctly align with the columns of the dataset sample. If they are incorrect, update the header names of your input dataset in Amazon Simple Storage Service (Amazon S3).

	Year	Month	DayofMonth	DayOfWeek	DepTime	CRSDepTime	ArrTime	CRSArrTime	UniqueCarrier	FlightNum	...	DepDelay	Origin	Dest	Distance	TaxiIn	TaxiOut	Cancelled	CancellationCode	Diverted	DelayType
0	2008	2	19	2	1219.0	1203	1446.0	1421	FL	1695	...	16.0	EWR	ATL	745	5.0	21.0	0		0	CarrierDelay
1	2008	2	19	2	1545.0	1430	1650.0	1534	YV	2826	...	75.0	ONT	LAS	197	11.0	12.0	0		0	LateAircraftDelay
2	2008	1	11	5	948.0	915	1026.0	948	FL	188	...	33.0	ATL	STL	483	4.0	14.0	0		0	CarrierDelay
3	2008	3	27	4	1943.0	1720	2211.0	1930	MQ	4007	...	143.0	ORD	CMH	296	8.0	29.0	0		0	CarrierDelay
4	2008	2	23	6	1742.0	1729	2012.0	1957	FL	625	...	13.0	PHL	MCO	861	8.0	13.0	0		0	CarrierDelay
5	2008	2	18	1	1201.0	1130	1313.0	1234	DL	462	...	31.0	SAV	ATL	215	9.0	21.0	0		0	CarrierDelay
6	2008	1	2	3	1027.0	945	1132.0	1055	MQ	3403	...	42.0	DFW	HOU	247	3.0	15.0	0		0	CarrierDelay
7	2008	1	13	7	1527.0	1510	1839.0	1820	MQ	4153	...	17.0	ORD	PVD	849	3.0	19.0	0		0	LateAircraftDelay
8	2008	2	23	6	1812.0	1640	1855.0	1721	OH	5146	...	92.0	ABE	LGA	82	5.0	8.0	0		0	NASDelay
9	2008	1	17	4	644.0	640	751.0	734	UA	1235	...	4.0	DSM	DEN	589	7.0	34.0	0		0	NASDelay

Automatic ML selection

TRIAL COMPONENTS

1 row selected 0/20 filters

Search column name to start

Add chart | Deploy model

Trial name	ObjectiveMetric	validation:accuracy	validation:merror	validation:macro_recall	validation:objective_l...	validation:mult
tuning-job-1-afb5da873b2e4c16a4-249-c971f83e-aws-trial	0.70051	0.70051	0.29949			
tuning-job-1-afb5da873b2e4c16a4-248-8b75985a-aws-trial	0.66297	0.66297	0.33703			
tuning-job-1-afb5da873b2e4c16a4-251-1f5e410c-aws-trial	0.20051194727420807			0.15630197525024414	2.1411008834838867	0.2005119472
tuning-job-1-afb5da873b2e4c16a4-247-dfeee99f-aws-trial	0.69625	0.69625	0.30375			
tuning-job-1-afb5da873b2e4c16a4-245-8a9e4ff4-aws-trial	0.69454	0.69454	0.30546			
tuning-job-1-afb5da873b2e4c16a4-250-5393ab3b-aws-trial	0.64505	0.64505	0.35495			
tuning-job-1-afb5da873b2e4c16a4-246-d9a06691-aws-trial	0.67236	0.67236	0.32765			
tuning-job-1-afb5da873b2e4c16a4-243-1944b28b-aws-trial	0.69198	0.69198	0.30802			
tuning-job-1-afb5da873b2e4c16a4-244-99ee4a6a-aws-trial	0.69198	0.69198	0.30802			
tuning-job-1-afb5da873b2e4c16a4-242-20856684-aws-trial	0.67065	0.67065	0.32935			
tuning-job-1-afb5da873b2e4c16a4-235-6beceaae-aws-trial	0.70734	0.70734	0.29266			
tuning-job-1-afb5da873b2e4c16a4-241-684c9a2b-aws-trial	0.69795	0.69795	0.30205			
tuning-job-1-afb5da873b2e4c16a4-240-a1323164-aws-trial	0.6220099925994873	0.6220099925994873	0.3779900074005127			
tuning-job-1-afb5da873b2e4c16a4-239-24678556-aws-trial	0.69966	0.69966	0.30034			
tuning-job-1-afb5da873b2e4c16a4-238-9dd93711-aws-trial	0.67918	0.67918	0.32082			
tuning-job-1-afb5da873b2e4c16a4-237-4ee36a8a-aws-trial	0.6851500272750854	0.6851500272750854	0.31485000252723694			
tuning-job-1-afb5da873b2e4c16a4-236-f118ec26-aws-trial	0.69625	0.69625	0.30375			
tuning-job-1-afb5da873b2e4c16a4-234-b0000dcd-aws-trial	0.7133100032806396	0.7133100032806396	0.28668999671936035			
tuning-job-1-afb5da873b2e4c16a4-233-451c99bd-aws-trial	0.7090399861335754	0.7090399861335754	0.29096001386642456			
tuning-job-1-afb5da873b2e4c16a4-232-16f3d8e7-aws-trial	0.70563	0.70563	0.29437			
tuning-job-1-afb5da873b2e4c16a4-230-8513450a-aws-trial	0.7167199850082397	0.7167199850082397	0.28327998518943787			
tuning-job-1-afb5da873b2e4c16a4-231-512d9de5-aws-trial	0.69113	0.69113	0.30887			
tuning-job-1-afb5da873b2e4c16a4-229-38a1bd73-aws-trial	0.69113	0.69113	0.30887			
tuning-job-1-afb5da873b2e4c16a4-227-b630bbe3-aws-trial	0.7099000215530396	0.7099000215530396	0.29010000824928284			
tuning-job-1-afb5da873b2e4c16a4-226-ae7b9917-aws-trial	0.69198	0.69198	0.30802			
tuning-job-1-afb5da873b2e4c16a4-228-6532284c-aws-trial	0.6006799936294556	0.6006799936294556	0.39932000637054443			
tuning-job-1-afb5da873b2e4c16a4-224-d76ecc6a-aws-trial	0.69539	0.69539	0.30461			
tuning-job-1-afb5da873b2e4c16a4-225-70971c62-aws-trial	0.7107499837875366	0.7107499837875366	0.289249986410141			

Model leaderboard

EXPERIMENT: AIRLINE-SAMPLED-MC-2

Problem type MulticlassClassification

Open candidate generation notebook | Open data exploration notebook

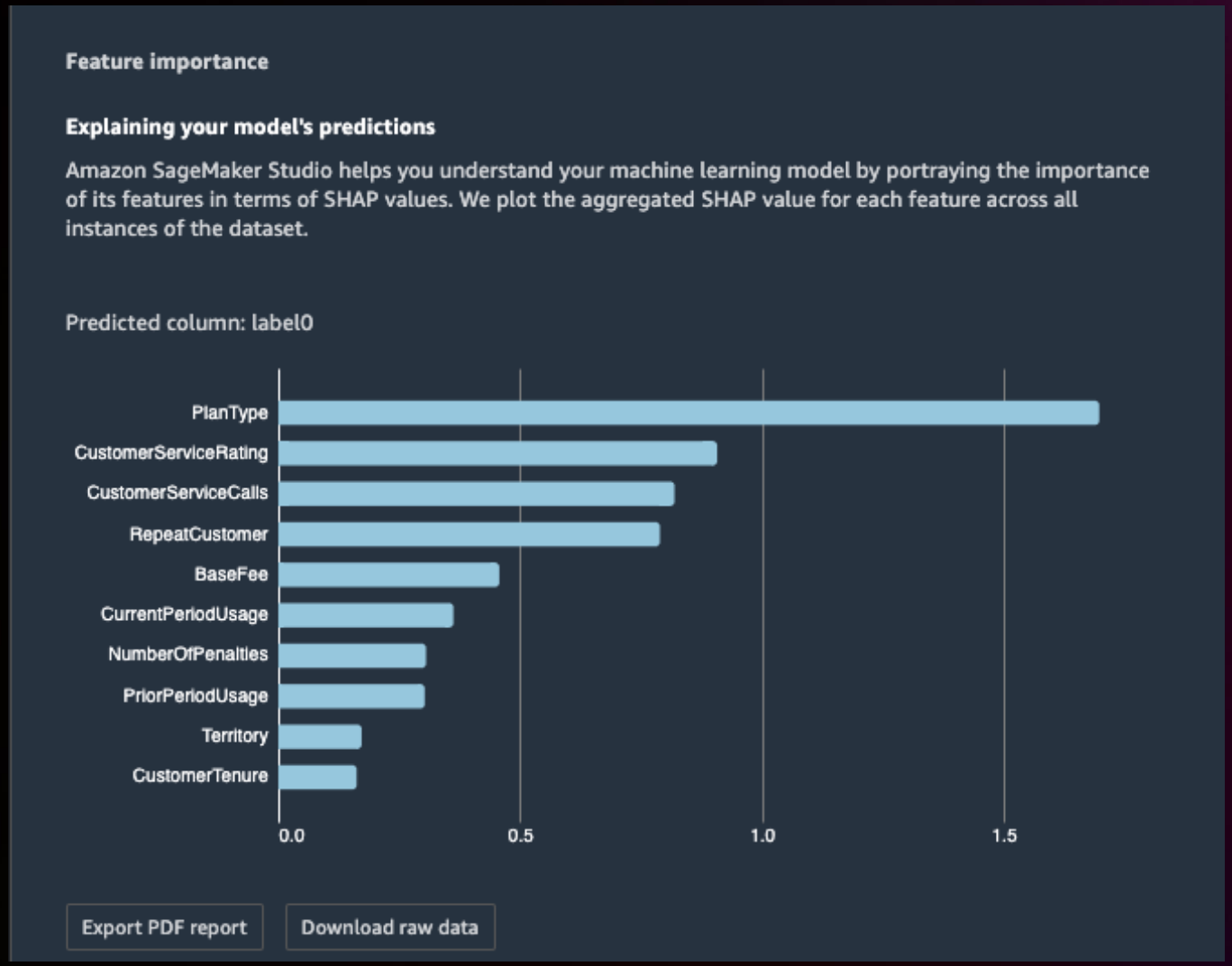
Trials | Job profile

TRIALS 0 rows selected

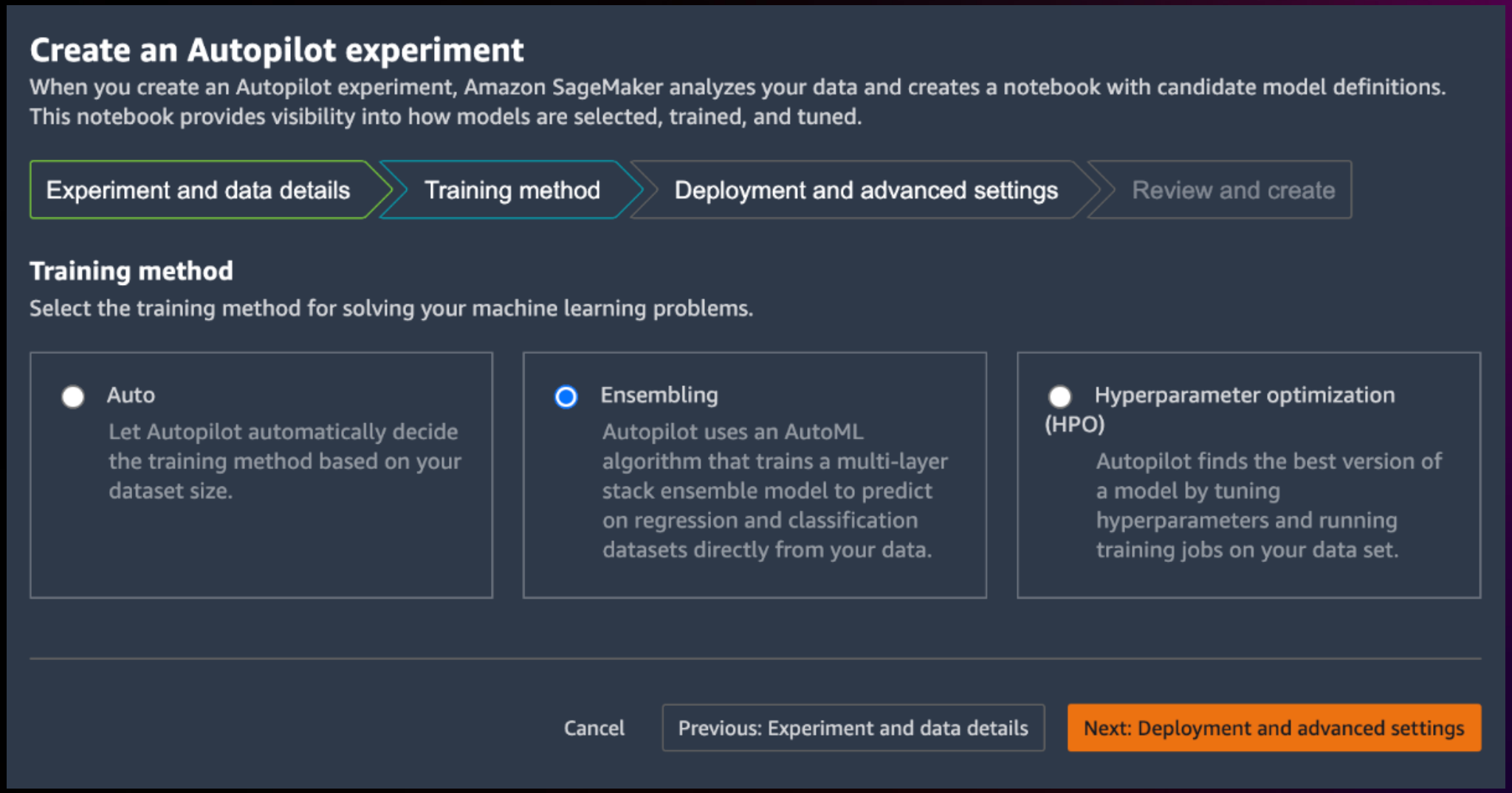
Deploy model

Trial name	Status	Start time	Objective: Accuracy
tuning-job-1-afb5da873b2e4c16a4-181-26bee647	Completed	1 month ago	0.7226999998092651
★ *Best:* tuning-job-1-afb5da873b2e4c16a4-163-7...	Completed	1 month ago	0.7226999998092651
tuning-job-1-afb5da873b2e4c16a4-155-7a3d564e	Completed	1 month ago	0.7218400239944458
tuning-job-1-afb5da873b2e4c16a4-151-a74a1321	Completed	1 month ago	0.7218400239944458
tuning-job-1-afb5da873b2e4c16a4-087-2d8bcb7e	Completed	1 month ago	0.720990002155304
tuning-job-1-afb5da873b2e4c16a4-143-90c8282f	Completed	1 month ago	0.720990002155304
tuning-job-1-afb5da873b2e4c16a4-110-99577f5e	Completed	1 month ago	0.720990002155304
tuning-job-1-afb5da873b2e4c16a4-109-c776de67	Completed	1 month ago	0.720990002155304
tuning-job-1-afb5da873b2e4c16a4-166-04265c01	Completed	1 month ago	0.7201399803161621
tuning-job-1-afb5da873b2e4c16a4-249-c971f83e	Completed	1 month ago	0.7192800045013428
tuning-job-1-afb5da873b2e4c16a4-203-3e177457	Completed	1 month ago	0.7184299826622009
tuning-job-1-afb5da873b2e4c16a4-245-8a9e4ff4	Completed	1 month ago	0.7184299826622009
tuning-job-1-afb5da873b2e4c16a4-211-3bda6dc4	Completed	1 month ago	0.7175800204277039
tuning-job-1-afb5da873b2e4c16a4-132-a34915b9	Completed	1 month ago	0.7175800204277039
tuning-job-1-afb5da873b2e4c16a4-178-2fbbec23	Completed	1 month ago	0.7175800204277039
tuning-job-1-afb5da873b2e4c16a4-113-ad2f5aa6	Completed	1 month ago	0.7175800204277039
tuning-job-1-afb5da873b2e4c16a4-130-07225d3c	Completed	1 month ago	0.7175800204277039
tuning-job-1-afb5da873b2e4c16a4-235-6beceaae	Completed	1 month ago	0.7175800204277039
tuning-job-1-afb5da873b2e4c16a4-230-8513450a	Completed	1 month ago	0.7167199850082397
tuning-job-1-afb5da873b2e4c16a4-212-55b029a4	Completed	1 month ago	0.7167199850082397
tuning-job-1-afb5da873b2e4c16a4-194-f56fcc18	Completed	1 month ago	0.7167199850082397
tuning-job-1-afb5da873b2e4c16a4-192-11f7443b	Completed	1 month ago	0.7167199850082397
tuning-job-1-afb5da873b2e4c16a4-165-86bf295d	Completed	1 month ago	0.7167199850082397
tuning-job-1-afb5da873b2e4c16a4-244-99ee4a6a	Completed	1 month ago	0.7167199850082397
tuning-job-1-afb5da873b2e4c16a4-221-09826f70	Completed	1 month ago	0.7167199850082397
tuning-job-1-afb5da873b2e4c16a4-213-a39ef206	Completed	1 month ago	0.7167199850082397

Feature importance



Customize AutoML journey



Automatic notebook creation

Amazon SageMaker Autopilot Candidate Definition Notebook

This notebook was automatically generated by the AutoML job **airline-sampled-mc-2**. This notebook allows you to customize the candidate definitions and execute the SageMaker Autopilot workflow.

The dataset has **25** columns and the column named **DelayType** is used as the target column. This is being treated as a **MulticlassClassification** problem. The dataset also has **5** classes. This notebook will build a MulticlassClassification model that **maximizes** the **"ACCURACY"** quality metric of the trained models. The **"ACCURACY"** metric provides the percentage of times the model predicted the correct class.

As part of the AutoML job, the input dataset has been randomly split into two pieces, one for **training** and one for **validation**. This notebook helps you inspect and modify the data transformation approaches proposed by Amazon SageMaker Autopilot. You can interactively train the data transformation models and use them to transform the data. Finally, you can execute a multiple algorithm hyperparameter optimization (multi-algo HPO) job that helps you find the best model for your dataset by jointly optimizing the data transformations and machine learning algorithms.

Available Knobs Look for sections like this for recommended settings that you can change.

Contents

Thank you!

Cyrus Vahid
cyrusmv@amazon.com

Michael Dasseville
dssvm@amzon.com

Please complete the session survey in the **mobile app**