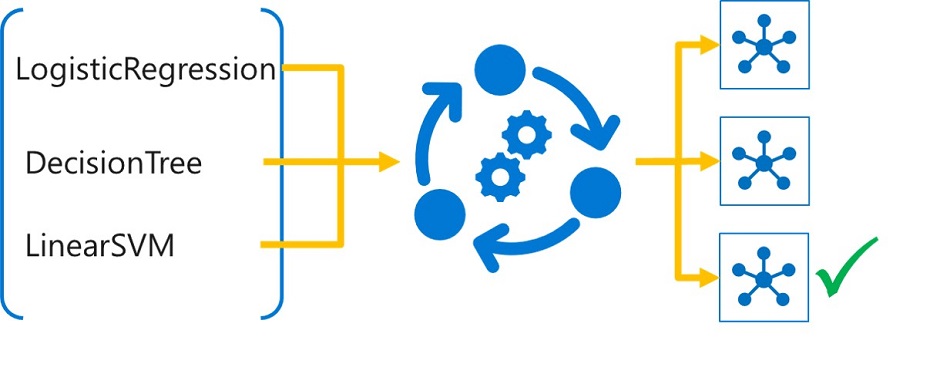
AutoML allows you to try multiple preprocessing transformations and algorithms with your data to find the best machine learning model.



 **Data Preparation for AutoML**:

* Collect and prepare training data.
* Create a data asset in Azure Machine Learning as an MLTable data asset, which includes the data schema.
* Store data in a folder with an MLTable file to create the data asset.

 **Specifying Data Input**:

* Use the following code to specify the data asset as input:

from azure.ai.ml.constants import AssetTypes

from azure.ai.ml import Input

my\_training\_data\_input = Input(type=AssetTypes.MLTABLE, path="azureml:input-data-automl:1")

 **Preprocessing Transformations**:

* AutoML applies automatic scaling and normalization to numeric data to prevent large-scale features from dominating training.
* Multiple scaling or normalization techniques are applied during the experiment.

 **Featurization Options**:

* Default: AutoML performs featurization, including:
  + Missing value imputation.
  + Categorical encoding.
  + Dropping high-cardinality features.
  + Feature engineering (e.g., deriving date parts from DateTime features).
* Customization: You can specify imputation methods or disable featurization if desired.

 **Review Post-Experiment**:

* After the experiment, review applied scaling and normalization methods.
* AutoML will notify about detected data issues, such as missing values or class imbalance.

**Running an Automated Machine Learning (AutoML) Experiment**

1. **Specify Task and Algorithms**:
   * AutoML uses different algorithms based on the specified task (e.g., classification).
   * Common classification algorithms include Logistic Regression, Light GBM, Decision Tree, Random Forest, Naive Bayes, Linear SVM, XGBoost, etc.
   * You can restrict specific algorithms if needed.
2. **Configure an AutoML Experiment**:
   * Use the automl class from the Python SDK (v2) to configure the experiment:

python

Copy code

from azure.ai.ml import automl

classification\_job = automl.classification(

compute="aml-cluster",

experiment\_name="auto-ml-class-dev",

training\_data=my\_training\_data\_input,

target\_column\_name="Diabetic",

primary\_metric="accuracy",

n\_cross\_validations=5,

enable\_model\_explainability=True

)

* + my\_training\_data\_input should be an MLTable data asset in the Azure ML workspace.

1. **Specify the Primary Metric**:
   * Choose the primary metric to optimize the model:

python

Copy code

from azure.ai.ml.automl import ClassificationPrimaryMetrics

list(ClassificationPrimaryMetrics)

* + Select an appropriate primary metric from the available options.

1. **Set Experiment Limits**:
   * Configure limits to control compute costs and training time:

python

Copy code

classification\_job.set\_limits(

timeout\_minutes=60,

trial\_timeout\_minutes=20,

max\_trials=5,

enable\_early\_termination=True,

)

* + These settings help manage the duration and number of trials, as well as enable early termination if performance stagnates.
  + timeout\_minutes: Number of minutes after which the complete AutoML experiment is terminated.
  + trial\_timeout\_minutes: Maximum number of minutes one trial can take.
  + max\_trials: Maximum number of trials, or models that will be trained.
  + enable\_early\_termination: Whether to end the experiment if the score isn't improving in the short term

1. **Run Multiple Trials in Parallel**:
   * Optimize time by running trials in parallel, adjusting based on the compute cluster’s node capacity:

python

Copy code

classification\_job.set\_limits(

max\_concurrent\_trials=4 # Adjust based on cluster size

)

1. **Exclude/Include Algorithms and Use Ensemble Models**:
   * Customize the algorithms AutoML will use:

python

Copy code

classification\_job.set\_training(

blocked\_algorithms=["RandomForest"], # Exclude specific algorithms

allowed\_algorithms=["LogisticRegression", "LightGBM"], # Include specific algorithms

enable\_ensembling=True # Enable or disable ensemble models

)

1. **Submit the AutoML Experiment**:
   * Submit the job and monitor it in the Azure ML studio:

python

Copy code

# Submit the AutoML job

returned\_job = ml\_client.jobs.create\_or\_update(classification\_job)

# Get the job monitoring link

aml\_url = returned\_job.studio\_url

print("Monitor your job at", aml\_url)

**Monitoring and Results**

* Use the provided aml\_url to monitor the job in the Azure Machine Learning studio, reviewing progress and results.

**Evaluating and Comparing Models After an AutoML Experiment**

1. **Reviewing Models in Azure ML Studio**:
   * Navigate to the AutoML experiment in the Azure Machine Learning studio.
   * On the **Overview** page, review the input data asset and summary of the best model.
   * Select the **Models** tab to explore all trained models.
2. **Exploring Preprocessing Steps**:
   * **Data Guardrails**: These include class balancing detection, missing feature values imputation, and high cardinality feature detection.
     + **Passed**: No issues detected; no action required.
     + **Done**: Changes applied to data; review the changes.
     + **Alerted**: Issues detected but not fixed; review and address these issues manually.
   * Scaling and normalization techniques applied to models are listed under the Algorithm name, e.g., MaxAbsScaler, LightGBM.
3. **Retrieving the Best Run and Its Model**:
   * Models are sorted by the primary metric, with the best performing model at the top.
   * You can edit the columns in the **Models** tab to include other metrics for a comprehensive comparison.
   * To generate explanations for a model, select the model and click **Explain model**.
4. **Generating Model Explanations**:
   * When configuring the AutoML experiment, specify that explanations should be generated for the best performing model.
   * If interested in another model's interpretability, select that model and choose **Explain model**.

As a data scientist, you'll want to develop a model in a notebook as it allows you to quickly test and run code.

Anytime you train a model, you want the results to be reproducible. By tracking and logging your work, you can review your work at any time and decide what the best approach is to train a model.

**MLflow** is an open-source library for tracking and managing your machine learning experiments. In particular, **MLflow Tracking** is a component of MLflow that logs everything about the model you're training, such as **parameters**, **metrics**, and **artifacts**.

**Use Azure Machine Learning notebooks**

Within the Azure Machine Learning workspace, you can create notebooks and connect the notebooks to an Azure Machine Learning managed **compute instance**.

When you're running a notebook on a compute instance, MLflow is already configured, and ready to be used.

To verify that the necessary packages are installed, you can run the following code:

pip show mlflow

pip show azureml-mlflow

**Training and Tracking Models in Notebooks with MLflow**

**Creating an MLflow Experiment**

* Group model training results by creating an MLflow experiment:

import mlflow

mlflow.set\_experiment(experiment\_name="heart-condition-classifier")

**Logging Results with MLflow**

**Enable Autologging**

* **Autologging** allows MLflow to automatically log metrics, parameters, artifacts, and models relevant to the machine learning framework you're using.
* Enable autologging for XGBoost:

from xgboost import XGBClassifier

with mlflow.start\_run():

mlflow.xgboost.autolog()

model = XGBClassifier(use\_label\_encoder=False, eval\_metric="logloss")

model.fit(X\_train, y\_train, eval\_set=[(X\_test, y\_test)], verbose=False)

* MLflow will automatically start a run and track the experiment's run in Azure Machine Learning.

**Use Custom Logging**

* **Custom logging** allows you to log additional or specific information not covered by autologging.

**Common MLflow Custom Logging Functions:**

* mlflow.log\_param(key, value): Logs a single key-value parameter.
* mlflow.log\_metric(key, value): Logs a single key-value metric (value must be a number).
* mlflow.log\_artifact(local\_path, artifact\_path): Logs a file (e.g., a plot saved as an image).
* mlflow.log\_model(model, artifact\_path, \*\*kwargs): Logs a model with additional details such as custom signature and environment.

**Example of Custom Logging:**

from xgboost import XGBClassifier

from sklearn.metrics import accuracy\_score

with mlflow.start\_run():

model = XGBClassifier(use\_label\_encoder=False, eval\_metric="logloss")

model.fit(X\_train, y\_train, eval\_set=[(X\_test, y\_test)], verbose=False)

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

mlflow.log\_metric("accuracy", accuracy)