Using IntegratedML

Version 2020.3
2021-02-04

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

Using IntegratedML
InterSystems IRIS Data Platform Version 2020.3 2021-02-04

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

List of Figures

About This Guide

This guide describes, to end users and to developers, how to use IntegratedML in InterSystems IRIS®. It includes the following sections:

- Introduction to ML
- IntegratedML Basics
- Providers
- ML Configurations
- Model Maintenance
- About AutoML

Appendix:

- IntegratedML SQL Commands
- IntegratedML SQL Functions

For a detailed outline, see the table of contents.

For more information about InterSystems SQL, see the following guides:

- *Using InterSystems SQL* provides in-depth material on SQL components and features, executing SQL queries, error and transaction processing.
- *InterSystems SQL Reference* provides reference material on InterSystems SQL commands, functions, and predicate conditions, and lists of data types and reserved words.

At https://learning.intersystems.com/course/view.php?name=Learn%20IntegratedML you can find additional content related to machine learning and IntegratedML.

1 Introduction to ML

IntegratedML is a feature within InterSystems IRIS® which allows you to use automated machine learning functions directly from SQL to create and use predictive models.

1.1 Purpose

Successful organizations recognize the need to develop applications that effectively harness the massive amounts of data available to them. These organizations want to use machine learning to train predictive models from large datasets, so that they can make critical decisions based on their data. This places organizations without the in-house expertise to build machine learning models at a significant disadvantage. For this reason, InterSystems has created IntegratedML.

IntegratedML enables developers and data analysts to build and deploy machine learning models within a SQL environment, without any expertise required in feature engineering or machine learning algorithms. Using IntegratedML, developers can use SQL queries to create, train, validate, and execute machine learning models.

IntegratedML considerably reduces the barrier to entry into using machine learning, enabling a quick transition from having raw data to having an implemented model. It is not meant to replace data scientists, but rather complement them.

1.2 Background

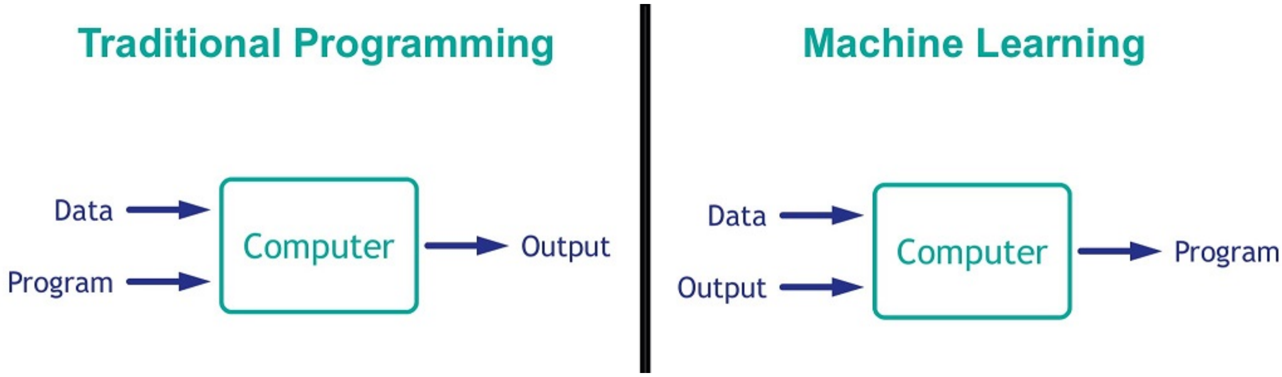
To use IntegratedML, you need an introductory understanding of several commonly used terms:

- Machine learning
- Models
- Training
- Features and labels
- Model validation

What is Machine Learning?

Machine learning is the study of computer algorithms that identify and extract patterns from data in order to build and use predictive *models*.

Figure 1–1: Traditional Programming vs. Machine Learning



In traditional programming, a program is manually developed that, when executed on input data, generates the desired output. In machine learning, the computer takes sample data and its known (or expected) output to develop a program (in this case, a predictive model), which can in turn be executed on further data.

Training a Model

The *training* process is how a machine learning algorithm develops a predictive model. The algorithm uses sample data, or *training data*, to identify patterns that map the inputs to the desired output. These inputs (or *features)* and outputs (or *labels*) are columns in the data set. A trained machine learning model has an algorithmically derived relationship between the features and the resulting label.

Validating a Model

After training a model, but before deployment, you can validate your model to confirm that is useful on data aside from the data that was used to train it. *Model validation* is the process of evaluating a model's predictive performance by comparing the model's output to the results of real data. While training data was used to train the model, *testing* data is used to validate it. In the simplest case, the testing dataset is data from an original dataset that is set aside from the training data.

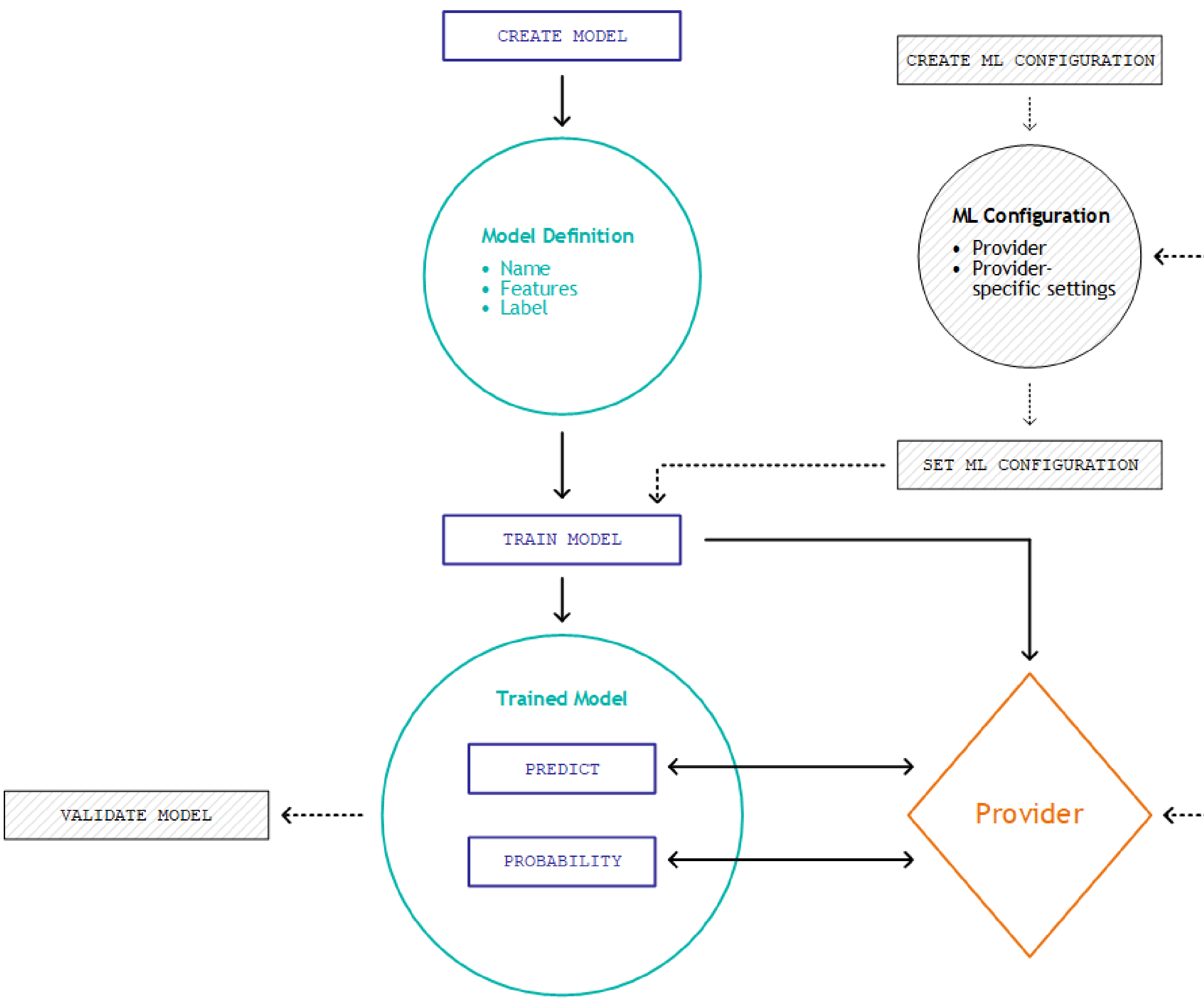
Using a Model

A trained machine learning model is used to make *predictions* on new data. This data must contain the same features as the training and testing data, but without the label column as this is the output of the model.

2
IntegratedML Basics

IntegratedML is a feature within InterSystems IRIS® which allows you to use automated machine learning functions directly from SQL to create and use predictive models.

Figure 2–1: IntegratedML Workflow



1. To use IntegratedML, you begin by specifying a model definition, which contains metadata about the input fields (features), predicted field (label), and the data types of these fields. The data itself is not stored within the model definition; just the structure of the data.
   - *Optional* — You can select the ML configuration, which specifies a provider to perform training. You can customize this configuration before training, or use the system default configuration without any action needed.
2. You train the model on data, using the provider specified by your ML configuration. The provider uses a structured process to compare the performance of different machine learning model types (linear regression, random forest, etc.) with the dataset and return the appropriate model.
   - *Optional* — After training, you can validate the model using test data to evaluate the predictive performance of the model.
3. Your trained model can now be invoked by SQL functions to make predictions on data.

Definitions

See below for definitions of IntegratedML-specific terms:

Models

Models are the primary objects used in IntegratedML. There are two types of model entities:

- *Model Definitions* — With IntegratedML, models are part of the database schema, like tables or indexes. The **CREATE MODEL** statement introduces a new model definition into the schema. This model definition specifies the features, labels, and data types, along with the ML configuration to be used for training.
- *Trained Models*— The **TRAIN MODEL** command uses a model definition to train a model with a provider specified by your configuration. This trained model is used to make predictions on data.

Providers

Several organizations offer ML-as-a-Service, supplying the tools and computation power to develop machine learning models based on datasets supplied by customers. These automated solutions often come in standalone applications, with no framework that connects directly to your datasets. You are then burdened with exporting your data to other workflows, subject to conditions that vary based on the machine learning framework.

IntegratedML addresses these issues by bringing automated machine learning capabilities directly inside the InterSystems IRIS® data platform, facilitating the connection between your data in InterSystems IRIS and these automated workflows. *Providers* are powerful machine learning frameworks that are accessible in a common interface in IntegratedML. The following providers are available:

- AutoML — a machine learning engine developed by InterSystems, housed in InterSystems IRIS
- H2O — an open-source automated machine learning platform
- DataRobot — an advanced enterprise automated machine learning platform

ML Configurations

An *ML configuration* is a collection of settings that IntegratedML uses to train a model. Primarily, a configuration specifies a machine learning provider that will perform training. Depending on the provider, the configuration may also specify requisite information for connection such as a URL and/or an API token. A default ML configuration is immediately active upon installation, requiring no adjustment in a simplest case. Optionally, you can create and select additional configurations to suit individual needs.

2.1 Creating Model Definitions

Before you can train a model, you must use the **CREATE MODEL** statement to specify a model definition. A model definition is a template that IntegratedML uses to train models; it contains metadata about the input fields (features), predicted field (label), and the data types of these fields. Only the structure of the data is stored within the model definition, not the data itself.

Syntax

The **CREATE MODEL** statement has the following syntax:

```
CREATE MODEL model-name PREDICTING (label-column) [ WITH feature-column-clause ] FROM model-source [
USING json-object-string ]
```

Or

```
CREATE MODEL model-name PREDICTING (label-column) WITH feature-column-clause [ FROM model-source ] [
USING json-object-string ]
```

Or

```
CREATE MODEL model-name PREDICTING (label-column) WITH feature-column-clause FROM model-source [ USING
 json-object-string ]
```

Examples

The following examples highlight use of different clauses for your **CREATE MODEL** statements:

Selecting Feature Columns with FROM

The following command creates a model definition `HousePriceModel`. The label column, or the column to be predicted, is `Price`. The columns of the `HouseData` table are implicitly sourced as the feature columns of the model definition by using a **FROM** clause:

```
CREATE MODEL HousePriceModel PREDICTING (Price) FROM HouseData
```

Important: If you do not use **FROM** in your **CREATE MODEL** statement, **FROM** is required in your **TRAIN MODEL** statement.

Selecting Feature Columns with WITH

The following command creates the same model definition as above, `HousePriceModel`, but uses a **WITH** clause to explicitly name the feature columns and their data types:

```
CREATE MODEL HousePriceModel PREDICTING (Price) WITH (TotSqft numeric, num_beds integer, num_baths
 numeric)
```

Selecting Training Parameters with USING

The following command uses the optional **USING** clause to specify parameters for the provider to train with. See Parameter Customization for further discussion of the **USING** clause.

```
CREATE MODEL HousePriceModel PREDICTING (Price) FROM HouseData USING {"seed": 3}
```

See More

You can view model definitions in the INFORMATION_SCHEMA.ML_MODELS view.

See Model Maintenance for more operations you can perform with your model definitions.

For complete information about the **CREATE MODEL** statement, see the reference.

2.1.1 Preparing Data for your Model

Before creating a model definition, you should consider the following items to prepare your dataset:

- Organize your data into a singular view or table.
- Evaluate your features:
  - If you have a column that is missing/NULL values for several rows, you may want to remove the column as this could adversely affect your trained model. You can also consider using a **CASE** expression to replace NULLs in your columns however you like.
  - Text-heavy data makes model training much slower.

2.2 Training Models

After creating a model definition, you can use the **TRAIN MODEL** statement to train a predictive model. IntegratedML trains this model using the provider specified by your ML configuration. The provider uses a structured process to compare the performance of different machine learning model types (linear regression, random forest, etc.) with the data and return the appropriate model.

Syntax

The **TRAIN MODEL** statement has the following syntax:

```
TRAIN MODEL model-name [ AS preferred-model-name ] [ NOT DEFAULT ] [ FOR label-column ] [ WITH
feature-column-clause ] [ FROM model-source ] [ USING json-object-string ]
```

Examples

The following examples highlight use of different clauses for your **TRAIN MODEL** statements:

Simplest Syntax

The following command trains a model with the `HousePriceModel` model definition:

```
TRAIN MODEL HousePriceModel
```

Important: If you did not use **FROM** in your **CREATE MODEL** statement, **FROM** is required in your **TRAIN MODEL** statement.

Selecting Training Data with FROM

The following command trains a model with the `HousePriceModel` model definition and `HouseData` as training data:

```
TRAIN MODEL HousePriceModel FROM HouseData
```

Naming the Training Run with AS

The following command trains a model with the `HousePriceModel` model definition. This trained model is saved with the name `HousePriceModelTrained`

```
TRAIN MODEL HousePriceModel AS HousePriceModelTrained FROM HouseData
```

Matching Feature Columns with WITH

The following command trains a model with the `HousePriceModel` model definition, and uses the **FOR** and **WITH** clauses to explicitly match the label and feature columns, respectively, between the training set and the model definition:

```
TRAIN MODEL HousePriceModel FOR house_price WITH (TotSqft = house_area, num_beds = beds, num_baths
 = bathrooms) FROM OtherHouseData
```

Selecting Training Parameters with USING

The following command uses the optional **USING** clause to specify parameters for the provider to train with. See Parameter Customization for further discussion of the **USING** clause.

```
TRAIN MODEL HousePriceModel USING {"seed": 3}
```

See More

You can view trained models and the results of training runs in the INFORMATION_SCHEMA.ML_TRAINED_MODELS view and INFORMATION_SCHEMA.ML_TRAINING_RUNS view, respectively. Trained models are associated with the model definition from which they were trained.

See Model Maintenance for more operations you can perform with your trained models.

For complete information about the **TRAIN MODEL** statement, see the reference.

2.2.1 Parameter Customization

The **USING** clause allows you to specify values for parameters that affect how your provider trains models. Machine learning experts can use this feature to fine-tune training runs to their needs.

```
TRAIN MODEL my-model USING {"seed": 3}
```

You can use the **USING** clause to pass provider-specific parameters for training. This clause accepts a JSON string containing key-value pairs of parameters and parameter values. These values are case-sensitive.

You can pass a **USING** clause in your **CREATE MODEL** and **TRAIN MODEL** statements, as well as in your ML configurations. They resolve as follows:

- Any parameters you specify with a **USING** clause in your **TRAIN MODEL** command override values for the same parameters you may have specified in your **CREATE MODEL** command or in your default ML configuration.
- Any parameters you specify with a **USING** clause in your **CREATE MODEL** command are implicitly used for your **TRAIN MODEL** command, and override values for the same parameters you may have specified in your default ML configuration.
- If you do not specify a **USING** in your **CREATE MODEL** or **TRAIN MODEL** commands, your model uses the **USING** clause specified by your default ML configuration.

All parameter names must be passed as strings, and the values must be passed in the type specific to the parameter. Lists should be input in the form of a string with commas as delimiters.

See Providers for information about the parameters available to each provider.

2.3 Validating Models

While training, the provider performs validation throughout the process of outputting a trained model. IntegratedML supplies the **VALIDATE** statement so that you can perform your own validation on a model. **VALIDATE** returns simple metrics for both regression and classification models based on the provided testing set.

Syntax

The **VALIDATE MODEL** statement has the following syntax:

```
VALIDATE MODEL trained-model-name [ AS validation-run-name ] [ USE preferred-trained-model-name ] [
WITH feature-column-clause ] FROM testing-data-set
```

Examples

The following examples highlight use of different clauses for your **VALIDATE MODEL** statements:

Simplest Syntax

The following command validates the trained `HousePriceModel` using `HouseTesting` as a testing data set:

```
VALIDATE MODEL HousePriceModel From HouseTesting
```

Naming the Validation Run with AS

The following command validates the trained `HousePriceModel` and saves the validation run as `HousePriceValidation` using `HouseTesting` as a testing data set:

```
VALIDATE MODEL HousePriceModel AS HousePriceValidation From HouseTesting
```

Matching Feature Columns with WITH

The following command validates the trained `HousePriceModel` and uses a **WITH** clause to explicitly match feature columns from the testing data set, `HouseTesting` :

```
VALIDATE MODEL HousePriceModel WITH (TotSqft = area, num_beds = beds, num_baths = baths) From
HouseTesting
```

See More

You can see validation runs and their results in the INFORMATION_SCHEMA.ML_VALIDATION_RUNS view and INFORMATION_SCHEMA.ML_VALIDATION_METRICS view, respectively

For complete information about the **VALIDATE MODEL** statement and validation metrics, see the reference.

2.4 Making Predictions

Each trained model has a specialized function, **PREDICT**, that calls on the provider to predict the result for each row in the applicable row-set. Classification models additionally have the **PROBABILITY** function, that calls on the provider to return the probability that the specified value is the correct result for the model.

These are scalar functions. and can be used anywhere in a SQL query and in any combination with other fields and functions.

2.4.1 PREDICT

You can use the **PREDICT** function to return the estimated (for regression models) or most likely (for classification models) value for the label column, by applying the given model (and hence provider) to each row in the applicable row-set. Each row provides the input columns (feature columns), from which the model returns the output (label).

Syntax

The **PREDICT** function has the following syntax:

```
PREDICT(model-name [ USE trained-model-name ] [ WITH feature-column-clause ] )
```

Examples

The following statements use the specialized **PREDICT** function of the model `HousePriceModel` in various forms:

```
SELECT *, PREDICT(HousePriceModel) FROM NewHouseData
```

```
SELECT * FROM NewHouseData WHERE PREDICT(HousePriceModel) > 500000
```

See More

For complete information about the **PREDICT** function, see the reference.

2.4.2 PROBABILITY

For classification models, you can use the **PROBABILITY** function to return the probability that a specified value is true for the given input:

Syntax

The **PROBABILITY** function has the following syntax:

```
PROBABILITY(model-name [ USE trained-model-name ] FOR label-value [ WITH feature-column-clause ] )
```

Examples

The following statements use the specialized **PROBABILITY** function of the model `EmailSpamModel` in various forms:

```
SELECT *, PROBABILITY(Iris_Model FOR 'iris-setosa') FROM Iris_Flower_Set
```

```
SELECT * FROM Iris_Flower_Set WHERE PROBABILITY(Iris_Model FOR 'iris-setosa') < 0.3
```

The following statement uses the specialized **PROBABILITY** function of the model `EmailFilter`. Since this is a binary classification model, with boolean values of 0 or 1 as the sole output, it can use the implicit **FOR** value of 1 to omit the **FOR** clause:

```
SELECT * EmailData WHERE PROBABILITY(EmailFilter) > 0.7
```

See More

For complete information about the **PROBABILITY** function, see the reference.

2.5 Walkthrough

This walkthrough illustrates the simple and powerful syntax IntegratedML offers through application to a real world scenario. Using a small number of SQL queries, the user develops a validated predictive model using their data.

Administrators in a health system have grown concerned about the increasing readmission rate for patients. Clinicians could be more cautions across the board when evaluating patient systems, but there are no defined criteria that would inform them of what to look for. Before investing fully into a new analytical solution, they task their data analyst with quickly developing a model to find trends in the profiles of patients that are readmitted. With their data stored on the InterSystems IRIS® database platform, the analyst knows that using IntegratedML would be far faster than any other solution that requires manually formatting and moving their data outside the platform.

Preparing the Data

Before using IntegratedML, the analyst prepares the data to make sure it is clean and ready for training. Any data the analyst needs from multiple tables are put into a singular view, for ease of use. In this example, the view is named `Hospital.PatientDataView`.

Customizing the Configuration

The analyst chooses to go with the default configuration for using IntregratedML. While the analyst is aware of the different providers they could use to train the model, for speed and ease of use they have gone with the default configuration with no additional syntax required.

Creating the Model

Data in hand, organized into a singular view, the analyst creates the model definition to be trained by an automated machine learning function. This definition, named `PatientReadmission`, specifies `IsReadmitted` as the label column to be predicted:

```
CREATE MODEL PatientReadmission PREDICTING (IsReadmitted) FROM Hospital.PatientDataView
```

Training the Model

The analyst now trains the model:

```
TRAIN MODEL PatientReadmission
```

The analyst does not need to specify any customized parameters for training.

Validating the Model

The analyst validates the model using a testing dataset they prepared (`Hospital.PatientDataViewTesting`), to get metrics on performance, and views these metrics:

```
VALIDATE MODEL PatientReadmission FROM Hospital.PatientDataViewTesting
SELECT * FROM INFORMATION_SCHEMA.ML_VALIDATION_METRICS
```

Making Predictions with the Model

With the model trained and validated, the analyst can now apply the model to make predictions on different datasets with the same schema. The analyst applies the model to `Hospital.NewPatientDataView`, a dataset containing information for patients that have been admitted in the past week, to see if any are susceptible for readmission:

```
SELECT ID FROM Hospital.NewPatientDataView WHERE PREDICT(PatientReadmission) = 1
```

Summary

In total, the analyst entered the following SQL queries to go from raw data to an active predictive model:

```
CREATE MODEL PatientReadmission PREDICTING (IsReadmitted) FROM Hospital.PatientDataView
TRAIN MODEL PatientReadmission
VALIDATE MODEL PatientReadmission FROM Hospital.PatientDataViewTesting
SELECT * FROM INFORMATION_SCHEMA.ML_VALIDATION_METRICS
SELECT ID FROM Hospital.NewPatientDataView WHERE PREDICT(PatientReadmission) = 1
```

3 Providers

Providers are powerful machine learning frameworks that are accessible in a common interface in IntegratedML. To choose a provider for training, select an ML configuration which specifies the desired provider.

You can pass additional parameters specific to these providers with a **USING** clause. See Parameter Customization for further discussion.

3.1 AutoML

AutoML is an automated machine learning system developed by InterSystems, housed within InterSystems IRIS®. IntegratedML, AutoML trains models quickly to produce accurate results. Additionally, AutoML features basic *natural language processing* (NLP), allowing the provider to smartly incorporate feature columns with unstructured text into machine learning models.

%AutoML is the system-default ML configuration for IntegratedML, and points to AutoML as the provider.

CAUTION: AutoML is currently not supported on Windows. *%H2O* is the system-default ML configuration for IntegratedML on Windows.

3.1.1 Feature Engineering

AutoML uses feature engineering to modify existing features, create new ones, and remove unnecessary ones. These steps improve training speed and performance, including:

- Column type classification to correctly use features in models
- Feature elimination to remove redundancy and improve accuracy
- One-hot encoding of categorical features
- Filling in missing or null values in incomplete datasets
- Creating new columns pertaining to hours/days/months/years, wherever applicable, to generate insights in your data related to time.

3.1.2 Model Selection

If a regression model is determined to be appropriate, AutoML uses a singular process for developing a regression model.

For classification models, AutoML uses the following selection process to determine the most accurate model:

1. If the dataset is too large, AutoML samples down the data to speed up the model selection process. The full dataset is still used for training after model selection.
2. AutoML determines if the dataset presents a binary classification problem, or if multiple classes are present, to use the proper scoring metric.
3. Using Monte Carlo cross validation, AutoML selects the model with the best scoring metrics for training on the entire dataset.

3.1.3 Known Issues

The AutoML provider may be inoperable on RHEL7 installations, due to missing libraries required by AutoML. Typical errors encountered during a **TRAIN MODEL** statement may include text such as `xgboost.core.XGBoostError`. Users may be able to remedy the issue by installing required packages mentioned by these error messages.

3.1.4 See More

For more information about AutoML, see About AutoML.

3.2 H20

You can specify H2O as your provider by setting `%H2O` as your ML configuration.

You can also create a new ML configuration where PROVIDER points to H2O.

3.2.1 Parameters

You can pass the parameters listed in the H2O documentation with a **USING** clause. Please consult this source for information regarding expected input and how these parameters are handled. Unknown parameters result in an error during training.

By default, `max_models` is set to 5.

3.2.2 Model Selection

Label columns that are classified as type string, integer, or binary result in a classification model. All other types result in a regression model. If you want an integer type column to be trained by H2O as a regression model, you need to add the key value pair: `"model_type":"regression"` to your **USING** clause.

3.2.3 Training Log Output

You can query the LOG column of the INFORMATION_SCHEMA.ML_TRAINING_RUNS view after training models using H2O.

3.2.4 Known Issues

- When training with the H2O provider, you may see the following error message:

```
LogMessage: %ML Provider '%ML.H2O.Provider' is not available on this instance
  > ERROR #5002: ObjectScript error: <READ>%GetResponse+4^%Net.Remote.Object.1
```

If you do, you can address this issue by performing the following:

1. Log into the Management Portal.
2. Go to **System Administration** > **Configuration** > **Connectivity** > **Object Gateways**.
3. Find the server named **%IntegratedML Server**, and select the **Edit** button.
4. Add the following to the **JVM arguments** field:

```
-Djava.net.preferIPv6Addresses=true -Djava.net.preferIPv4Addresses=false
```

- Setting the seed parameter with a **USING** clause for the H2O provider does not guarantee reproducible training runs. This is because the default training settings for H2O include the parameter `max_models` being set to 5, which triggers an early stopping mode. Reproducibility for the Gradient Boosting Model algorithm in H2O is a complex topic, as documented by H2O.

3.2.5 See More

For more information about H2O, see their documentation.

3.3 DataRobot

Important: You must have a business relationship with DataRobot to use their AutoML capabilities and view their documentation.

DataRobot clients can use IntegratedML to train models with data stored within InterSystems IRIS®.

You can specify DataRobot as your provider by selecting a DataRobot configuration as your default ML configuration:

```
SET ML CONFIGURATION datarobot_configuration
```

where `datarobot_configuration` is the name of an ML configuration where `PROVIDER` points to `DataRobot`.

3.3.1 Parameters

IntegratedML uses the DataRobot API to make an HTTP request to start modeling. Please consult their documentation for information regarding expected input and how these parameters are handled. Unknown parameters result in an error during training.

You can pass parameters with a **USING** clause.

By default, `quickrun` is set to `true`.

3.4 PMML

IntegratedML supports PMML as a PMML consumer, making it easy for you to import and execute your PMML models using SQL.

3.4.1 How PMML Models work in IntegratedML

As with any other provider, you use a **CREATE MODEL** statement to specify a model definition, including features and labels. This model definition must contain the same features and label that your PMML model contains.

The **TRAIN MODEL** statement operates differently. Instead of "training" data, the **TRAIN MODEL** statement imports your PMML model. No training is necessary because the PMML model exhibits the properties of a trained model, including information on features and labels. The model is identified by a **USING** clause.

Important: The feature and label columns specified in your model definition must match the feature and label columns of the PMML model.

While you still require a **FROM** clause in either your **CREATE MODEL** or **TRAIN MODEL** statement, the data specified is not used whatsoever.

Using your "trained" PMML model to make predictions works the same as any other trained model in IntegratedML. You can use the **PREDICT** function with any data that contains feature columns matching your PMML definition.

3.4.2 How to import a PMML Model

Before you can use a PMML model, set `%PMML` as your ML configuration, or select a different ML configuration where `PROVIDER` points to `PMML`.

You can specify a PMML model with a **USING** clause. You can choose one of the following parameters:

By Class Name

You can use the `"class_name"` parameter to specify the class name of a PMML model. For example:

```
USING {"class_name" : "IntegratedML.pmml.PMMLModel"}
```

By Directory Path

You can use the `"file_name"` parameter to specify the directory path to a PMML model. For example:

```
USING {"file_name" : "C:\temp\mydir\pmml_model.xml"}
```

3.4.3 Examples

The following examples highlight the multiple methods of passing a **USING** clause to specify a PMML model.

Specifying a PMML Model in an ML Configuration

The following series of statements creates a PMML configuration which specifies a PMML model for house prices by file name, and then imports the model with a **TRAIN MODEL** statement.

```
CREATE ML CONFIGURATION pmml_configuration PROVIDER PMML USING {"file_name" :
"C:\PMML\pmml_house_model.xml"}
SET ML CONFIGURATION pmml_configuration
CREATE MODEL HousePriceModel PREDICTING (Price) WITH (TotSqft numeric, num_beds integer, num_baths
numeric)
TRAIN MODEL HousePriceModel FROM HouseData
SELECT * FROM NewHouseData WHERE PREDICT(HousePriceModel) > 500000
```

Specifying a PMML Model in the TRAIN MODEL Statement

The following series of statements uses the provided `%PMML` configuration, and then specifies a PMML model by class name in the **TRAIN MODEL** statement.

```
SET ML CONFIGURATION %PMML
CREATE MODEL HousePriceModel PREDICTING (Price) WITH (TotSqft numeric, num_beds integer, num_baths
numeric)
TRAIN MODEL HousePriceModel FROM HouseData USING {"class_name" : "IntegratedML.pmml.PMMLHouseModel"}
SELECT * FROM NewHouseData WHERE PREDICT(HousePriceModel) > 500000
```

3.4.4 Additional Parameters

If your PMML file contains multiple models, IntegratedML uses the first model in the file by default. To point to a different model within the file, use the `model_name` parameter in your **USING** clause:

```
TRAIN MODEL my_pmml_model FROM data USING {"class_name" : my_pmml_file, "model_name" : "model_2_name"}
```

4

ML Configurations

An ML configuration is a collection of settings that IntegratedML uses to train a model. Primarily, a configuration specifies a machine learning provider that will perform training. Depending on the provider, the configuration may also specify requisite information for connection such as a URL and/or an API token.

You can use IntegratedML without any adjustment to your ML configuration necessary, as `%AutoML` is set as the system-default ML configuration upon installation.

CAUTION: AutoML is currently not supported on Windows. *%H2O* is set as the system-default ML configuration for IntegratedML on Windows.

4.1 Creating ML Configurations

While you can use the system-default ML configuration upon installation, you can also create new ML configurations for model training.

4.1.1 Creating ML Configurations using the System Management Portal

To create an ML configuration:

1. Log into the Management Portal.
2. Go to **System Administration** > **Configuration** > **Machine Learning Configurations**.
3. Select **Create New Configuration** and enter the following values for fields:
   - **Name** — The name of your ML configuration.
   - **Provider** — The machine learning provider your ML configuration connects to.

     If you select **DataRobot**, you must enter values for the following additional fields:
     - **URL** — The URL of a DataRobot endpoint
     - **API Token** — The API token of a DataRobot account
   - **Description** — Optional. A text description for your ML configuration.
   - **Using Clause** — Optional. A default **USING** clause for your ML configuration. See Parameter Customization for further discussion.
   - **Owner** — The owner of this ML configuration.

4. Select **Save** to save this new ML configuration.

To set this new ML configuration as the system-default, see Setting the System Default ML Configuration.

4.1.2 Creating ML Configurations using SQL

You can create a new configuration using the **CREATE ML CONFIGURATION** command.

Syntax

The **CREATE ML CONFIGURATION** statement has the following syntax:

```
CREATE ML CONFIGURATION ml-configuration-name PROVIDER provider-name [ %DESCRIPTION description-string
 ] [ USING json-object-string ] provider-connection-settings
```

Examples

The following examples highlight use of different clauses for your **CREATE ML CONFIGURATION** statements:

Simplest Syntax

The following command creates an ML Configuration, `H2OConfig`, that uses the H2O provider. No provider connection settings are needed when connecting to H2O:

```
CREATE ML CONFIGURATION H2OConfig PROVIDER H2O
```

Selecting Training Parameters with USING

The following command creates an ML Configuration, `H2OConfig`, that uses the H2O provider and specifies a default **USING** clause:

```
CREATE ML CONFIGURATION H2OConfig PROVIDER H2O USING {"nfolds": 4}
```

See More

To set this new ML configuration as the system-default, see Setting the System Default ML Configuration.

For complete information about the **CREATE ML CONFIGURATION** command, see the reference.

4.2 Setting the ML Configuration

IntegratedML provides the following configurations for immediate use:

- `%AutoML`
- `%H2O`
- `%PMML`

Upon installation, `%AutoML` is set as the system-default ML configuration. You can use IntegratedML without any adjustment to your configuration necessary. If you would like to specify a different ML configuration to use for your **TRAIN MODEL** statements, you can do so in one of the following methods:

- SQL — you can set the ML configuration for your given process
- System Management Portal — you can adjust the system-default ML configuration

You can see which ML configuration was used for your training run(s) by querying the INFORMATION_SCHEMA.ML_TRAINING_RUNS view.

4.2.1 Setting ML Configuration for the Given Process using SQL

You can use the **SET ML CONFIGURATION** statement to specify the ML configuration for your given process.

Syntax

The **SET ML CONFIGURATION** statement has the following syntax:

```
SET ML CONFIGURATION ml-configuration-name
```

See More

See the reference for more information about the **SET ML CONFIGURATION** statement.

4.2.2 Setting the System Default ML Configuration using the System Management Portal

You can set the system-default ML configuration in the **Machine Learning Configurations** page in the System Management Portal.

To set the system-default ML configuration:

1. Log into the Management Portal
2. Go to **System Administration** > **Configuration** > **Machine Learning Configurations**
3. Next to **System Default ML Configuration**, select the ML configuration of your choice.

Note: Setting the system-default ML configuration in this manner does not go into effect until you have started a new

4.3 Maintaining ML Configurations

You can perform the following operations to maintain your ML configurations:

- Altering ML Configurations
- Deleting ML Configurations

You can see which ML configuration was used for your training run(s) by querying the INFORMA-TION_SCHEMA.ML_TRAINING_RUNS view.

4.3.1 Altering ML Configurations

You can modify the properties of existing ML configurations.

4.3.1.1 Altering ML Configurations using the System Management Portal

To alter an ML configuration:

1. Log into the Management Portal
2. Go to **System Administration** > **Configuration** > **Machine Learning Configurations**
3. Select the name of a listed ML configuration and adjust the values of your choice.
4. Select **Save** to save this altered ML configuration.

4.3.1.2 Altering ML Configurations using SQL

You can alter a configuration using the **ALTER ML CONFIGURATION** statement.

Syntax

The **ALTER ML CONFIGURATION** statement has the following syntax:

```
ALTER ML CONFIGURATION ml-configuration-name alter-options
```

Where `alter-options` is one, or more, of the following:

- `PROVIDER` *`provider-name`*
- `%DESCRIPTION` *`description-string`*
- `USING` *`json-object-string`*
- *`provider-connection-settings`*

See More

For complete information about the **ALTER ML CONFIGURATION** command, see the reference.

4.3.2 Deleting ML Configurations

You can delete ML configurations.

4.3.2.1 Deleting ML Configurations using the System Management Portal

To delete an ML configuration:

1. Log into the Management Portal
2. Go to **System Administration** > **Configuration** > **Machine Learning Configurations**
3. Find the row of the ML configuration you want to delete and select **Delete**.

4.3.2.2 Deleting ML Configurations using SQL

You can delete a configuration using the **DROP ML CONFIGURATION** statement.

Syntax

The **DROP ML CONFIGURATION** statement has the following syntax:

```
DROP ML CONFIGURATION ml-configuration-name
```

See More

For complete information about the **DROP ML CONFIGURATION** command, see the reference.

5
Model Maintenance

You can perform the following operations to maintain your set of machine learning models:

- Viewing Models
- Altering Models
- Deleting Models

5.1 Viewing Models

When IntegratedML performs training or validation, this process is known as a “training run” or a “validation run.”

IntegratedML provides the following views, within the INFORMATION_SCHEMA class, that can be used to query information about models, trained models, training runs, and validation runs:

- ML_MODELS
- ML_TRAINED_MODELS
- ML_TRAINING_RUNS
- ML_VALIDATION_RUNS
- ML_VALIDATION_METRICS

5.1.1 ML_MODELS

This view returns one row for each model definition.

INFORMATION_SCHEMA.ML_MODELS contains the following columns:

Column Name	Description
CREATE_TIME_STAMP	Time when the model definition was created (UTC)
DEFAULT_SETTINGS	Default settings the model definition's provider uses
DEFAULT_TRAINED_MODEL_NAME	Default trained model name, if one has been trained
DEFAULT_TRAINING_QUERY	The FROM clause from the CREATE MODEL statement, if one was used
DESCRIPTION	Description of model definition
MODEL_NAME	Name of the model definition
PREDICTING_COLUMN_NAME	Name of the label column
PREDICTING_COLUMN_TYPE	Type of the label column
WITH_COLUMNS	Names of the feature columns

See More

See Creating Model Definitions for information about model definitions.

5.1.2 ML_TRAINED_MODELS

This view returns one row for each trained model.

INFORMATION_SCHEMA.ML_TRAINED_MODELS contains the following columns:

Column Name	Description
MODEL_INFO	Model information
MODEL_NAME	Name of the model definition
MODEL_TYPE	The model type (classification or regression)
PROVIDER	Provider used for training
TRAINED_MODEL_NAME	Name of the trained model
TRAINED_TIMESTAMP	Time when the trained model was created (UTC)

See More

See Training Models for information about trained models.

See Providers for information about providers.

5.1.3 ML_TRAINING_RUNS

This view returns one row for each training run.

INFORMATION_SCHEMA.ML_TRAINING_RUNS contains the following columns:

Column Name	Description
COMPLETED_TIMESTAMP	Time when the training run completed (UTC)
LOG	Training log output from the provider

Column Name	Description
ML_CONFIGURATION_NAME	Name of the ML configuration used for training
MODEL_NAME	Name of the model definition
PROVIDER	Name of the provider used for training
RUN_STATUS	Status of training run
SETTINGS	Any settings passed by a USING clause for the training run
START_TIMESTAMP	Time when the training run started (UTC)
STATUS_CODE	Training error (if encountered)
TRAINING_DURATION	Duration of training (in seconds)
TRAINING_RUN_NAME	Name of the training run
TRAINING_RUN_QUERY	Query used to source data from feature and label columns for training

See More

See Training Models for information about training runs.

5.1.4 ML_VALIDATION_RUNS

This view returns one row for each validation run.

INFORMATION_SCHEMA.ML_VALIDATION_RUNS contains the following columns:

Column Name	Description
COMPLETED_TIMESTAMP	Time when the validation run completed (UTC)
LOG	Validation log output
MODEL_NAME	Name of the model definition
RUN_STATUS	Validation status
SETTINGS	Validation run settings
START_TIMESTAMP	Time when the validation run started (UTC)
STATUS_CODE	Validation error (if encountered)
TRAINED_MODEL_NAME	Name of the trained model being validated
VALIDATION_DURATION	Validation duration (in seconds)
VALIDATION_RUN_NAME	Name of the validation run
VALIDATION_RUN_QUERY	Full query for dataset specified by FROM

See More

See Validating Models for information about validation runs.

5.1.5 ML_VALIDATION_METRICS

This view returns one row for each validation metric of each validation run.

INFORMATION_SCHEMA.ML_VALIDATION_METRICS contains the following columns:

Column Name	Description
METRIC_NAME	Validation metric name
METRIC_VALUE	Validation metric value
MODEL_NAME	Model name
TARGET_VALUE	Target value for validation metric
TRAINED_MODEL_NAME	Name of the trained model for this run
VALIDATION_RUN_NAME	Name of the validation run

See More

See Validation Metrics for information about the validation metrics that populate METRIC_NAME and METRIC_VALUE.

5.2 Altering Models

You can modify a model by using the **ALTER MODEL** statement.

Syntax

The **ALTER MODEL** statement has the following syntax:

```
ALTER MODEL model-name alter-action
```

Where *`alter-action`* can be one of the following:

- `PURGE ALL`
- `PURGE` *`integer`* `DAYS`
- `DEFAULT` *`preferred-model-name`*

Examples

This example uses the `PURGE` clause to delete all training run and validation run data associated with the model `WillLoanDefault`:

```
ALTER MODEL WillLoanDefault PURGE ALL
```

This example uses the `PURGE` clause to delete training run and validation run data associated with the model `WillLoanDefault` that is older than 7 days old:

```
ALTER MODEL WillLoanDefault PURGE 7 DAYS
```

See More

You can confirm that your alter statements succeeded by querying the views listed in Viewing Models.

For complete information about the **ALTER MODEL** command, see the reference.

5.3 Deleting Models

You can delete a model by using the **DROP MODEL** statement.

Syntax

The **DROP MODEL** statement has the following syntax:

```
DROP MODEL model-name
```

DROP MODEL deletes all training runs and validation runs for the associated model.

See More

You can confirm that your model has been deleted by querying the INFORMATION_SCHEMA.ML_MODELS view.

For complete information about the **DROP MODEL** command, see the reference.

6 About AutoML

Note: The following sections focus on the low-level details on the AutoML provider. For general information about choosing AutoML as the provider for IntergratedML, see AutoML.

AutoML is an automated machine learning system developed by InterSystems, housed within InterSystems IRIS® data platform. It is designed to quickly build accurate predictive models using your data, automating several key components of the machine learning process:

Figure 6–1: The Machine Learning Process

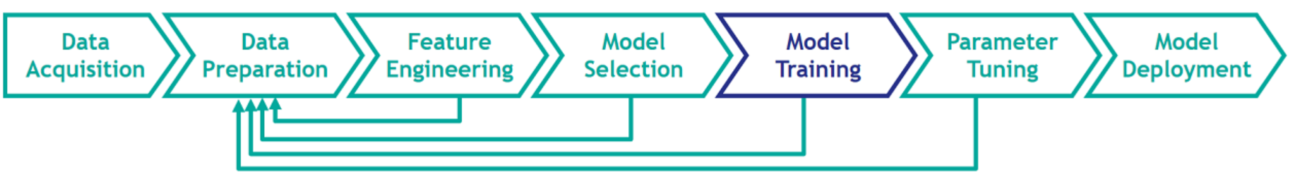
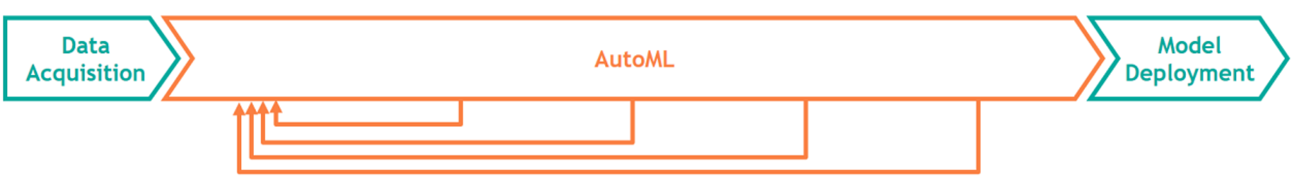


Figure 6–2: Automating the Machine Learning Process



After you train your model with AutoML, you can easily deploy your model by using the SQL syntax provided by IntegratedML.

6.1 Key Features

6.1.1 Natural Language Processing

AutoML leverages natural language processing (NLP) to turn your text features into numeric features for your predictive models. AutoML uses Term frequency-inverse document frequency (TFIDF) to evaluate key words in text and list columns.

6.1.2 Multi-Hot Encoding

While most of our data is *sparse*, machine learning algorithms can only understand *dense* data. In most data modeling workflows, data scientists are burdened with performing difficult, and cumbersome, manual transformations to convert their sparse data into dense data.

Unlike many workflows that require this manual step, AutoML performs this conversion seamlessly. Lists and one-to-many relationships are smartly "multi-hot encoded" to account for columns that are representing more than a single value.

For instance, assume a table that contains a list of medical conditions for each person:

Person	Conditions
Person A	['diabetes', 'osteoporosis', 'asthma']
Person B	['osteoporosis', 'hypertension']
Person C	['asthma', 'hypertension']
Person D	['hypertension', 'asthma']

Many machine learning functions treat these lists as separate entities, with one-hot encoding resulting in the following conversion:

Person	['diabetes', 'osteoporosis', 'asthma']	['osteoporosis', 'hypertension']	['asthma', 'hypertension']	['hypertension', 'asthma']
Person A	1	0	0	0
Person B	0	1	0	0
Person C	0	0	1	0
Person D	0	0	0	1

Instead, AutoML uses bag-of-words to create a separate column for each *value* in each list:

Person	'diabetes'	'osteoporosis'	'asthma'	'hypertension'
Person A	1	1	1	0
Person B	0	1	0	1
Person C	0	0	1	1
Person D	0	0	1	1

While other functions would have treated each person as having a separate list of medical conditions, AutoML's method allows a model to properly find patterns between each of these persons' set of medical conditions.

AutoML assumes that order does not matter. Person C and Person D share the same set of medical conditions, but just ordered differently. While other functions treat those two lists differently, AutoML identifies that they are the same.

6.2 Feature Engineering

AutoML performs two key steps of feature engineering:

- Column Type Classification
- Data Transformation

These steps help make the data compatible with the utilized machine learning models, and can greatly improve performance.

6.2.1 Column Type Classification

AutoML first examines the columns in the dataset and classifies them as a particular Python data type. For information about the conversion from DDL to Python data types, see DDL to Python Type Conversion.

The column types, along with how their classifications are made, are listed below:

Numeric Columns

Numeric columns are those that have the *numeric* pandas datatype, including *int8*, *int64*, *float32*, etc. All columns meeting this condition are included, except:

- Ignored Columns.
- Columns of the *timedelta* datatype.
- Columns with only one unique value.

Some columns with seemingly numeric data may be inappropriately classified as numeric columns. For example, an ID number of 1000 is not "half of" an ID number of 2000. You can properly treat these columns as category columns by recasting the numeric data with VARCHAR values.

Category Columns

Category columns are those that contain categorical values, meaning there are a relatively small, fixed number of values that appear. They satisfy the following criteria:

- Must be of *category* or *object* pandas datatype.
- Must not include Ignored Columns.
- Must not include List Columns.
- The number of unique values is less than 10% the total number of values.

Text Columns

Text columns are columns where the values look like sentences. AutoML looks for values that contain 4 or more words. They satisfy the following criteria:

- Must be of *category* or *object* pandas datatype.
- Must not include Ignored Columns.
- Must not include Category Columns.

- Must not include List Columns.
- The number of unique values is less than 10% the total number of values.

List Columns

List columns are those that contain list values. They satisfy the following criteria:

- Must be of *category* or *object* pandas datatype.
- Must not include Ignored Columns.
- Must be, or contain, one of the following types:
  - InterSystems IRIS data type %Library.String:list
  - InterSystems IRIS data type %Library.String:array
  - Python list. This is determined by checking the first 10 non-empty values of the column to see if the type of each value is a Python list.
  - String array. This is determined by checking the first 10 non-empty values of the column to see if the type of each value is a string, with starting character [, ending character], and of length at least 2.

Boolean Columns

Boolean columns are those that have the *bool* pandas datatype. They additionally satisfy the condition that they do not include Ignored Columns.

Ignored Columns

Ignored columns are those that are to be disregarded and removed before training. These include:

- The ID column.
- The label column.
- Columns with only one unique value (except for columns of *datetime* pandas datatype).

Date/Time Columns

Date/Time columns are those that have the *datetime* pandas datatype. They additionally satisfy the condition that they do not include Ignored Columns.

See below for discussion of additional date/time columns created.

6.2.1.1 DDL to Python Type Conversion

The following table maps DDL data types to the Python data types that AutoML uses to classify data columns.

DDL Data Type	Python Data Type
BIGINT	integer
BINARY	bytes
BIT	Boolean
DATE	datetime64 (numpy)
DECIMAL	decimal
DOUBLE	float

DDL Data Type	Python Data Type
INTEGER	integer
NUMERIC	float
REAL	float
SMALLINT	integer
TIME	datetime64 (numpy)
TIMESTAMP	datetime64 (numpy)
TINYINT	integer
VARBINARY	bytes
VARCHAR	string

For information about DDL data types, and their associated InterSystems IRIS® data types, see "Data Types" in the *InterSystems SQL Reference*.

6.2.2 Data Transformation

The Transform Function transforms the entire dataset into the form to be used by the machine learning models. It is applied on the training set before training, and on any future datasets before predictions are made.

Adding Additional Columns

Additional Date/Time columns are created. For every *datetime* column, the following separate columns are added whenever applicable:

- Hour of day.
- Day of week.
- Month of year.

AutoML also creates duration columns. Each column added represents one of the original date/time columns, and each value in this column is the duration between the dates of that particular date/time column and all other date/time columns. For example, consider patient data that has three date/time columns:

- Date of birth.
- Time of admission.
- Time of exit.

AutoML creates two useful duration columns from these columns: age (duration between date of birth and time of admission) and length of stay (duration between time of admission and exit).

Finally, for each list column present, another column is added simply with the size of the lists. That is, each value in the new column is the length of the corresponding list in the old column.

Replacing Missing Values

Datasets can often be incomplete, with missing values in some of their columns. To help compensate for this and improve performance, AutoML fills in missing/NULL values:

- For categorical and date columns, AutoML replaces missing values with the mode (most popular value) of the column.
- For numeric and duration columns, AutoML replaces missing values with the mean (average) of the column.

- For list and text columns, AutoML replaces missing values with an empty string.

Transforming Numeric Columns

For each numeric column, a standard scalar is fit. These include the original numeric columns, along with the duration and list size columns as well.

Numerical column values are also binned and then used as categorical values. These new categorical bin columns are added on separately in addition to the already present numerical columns. Each numerical column is separated into four bins, each representing a quartile of the values in that column. The new binned columns are treated as categorical columns.

Transforming Text and List Columns

For each text and list column, a vectorizer is fit to transform the data to the appropriate form needed for training. This is done with SciKit Learn's TFIDF Vectorizer. Please see their documentation.

The following parameters are used:

Parameter	Value
Convert to lowercase	True
Stop Words	None
N-Gram Range	(1,1)
Max Features	10000
Norm	L2

Binary Columns

Binary columns are simply transformed to be composed of 1's and 0's, with true values mapping to 1's.

Categorical Columns

Categorical columns are one-hot encoded before being used for training.

Feature Elimination

As the last step before training, feature elimination is performed to remove redundancy, improve training speed, and improve the accuracy of models. This is done using Scikit Learn's SelectFPR function.

The following parameters are used:

Parameter	Value
Scoring function	f_classif
alpha	0.2

6.3 Algorithms Used

AutoML uses four different models to make predictions.

For regression models:

- XGBRegressor

For classification models:

- Neural Network
- Logistic Regression
- Random Forest Classifier

6.3.1 XGBRegressor

For regression models, AutoML uses XGBoost's XGBRegressor class.

The model hyperparameters are detailed below:

Hyperparameter	Value
Max Depth	3
Learning Rate	0.1
Number of Estimators	100
Objective	Squared Error
Booster	Gbtree
Tree Method	Auto
Number of Jobs	1
Gamma (min loss reduction for partition on leaf)	0
Min Child Weight	1
Max Delta Step	0
L2 Regularization Lambda	1
Scale Positive Weight	1
Base/Initial Score	0.5

6.3.2 Neural Network

For the Neural Network model, AutoML uses TensorFlow with Keras as a wrapper.

The input layer has its size based on the number of features. This layer is then densely connected to a single hidden layer composed of 100 neurons, which implement the ReLU Activation Function. This hidden layer is densely connected to the final output layer, which implements the Softmax Activation Function. The number of neurons in the output layer is equivalent to the number of classes present for classification.

The model hyperparameters are detailed below:

Hyperparameter	Value
Optimizer (name)	Adam
Beta_1	0.9
Beta_2	0.999
Epsilon	1e-07
Amsgrad	False
Loss	Sparse Categorical Crossentropy

6.3.3 Logistic Regression

For the Logistic Regression Model, AutoML uses SciKit Learn's Logistic Regression class.

The model hyperparameters are detailed below:

Hyperparameter	Value
Penalty	L2
Dual Formulation	False
Tolerance	1e-4
C (Inverse Regularization Parameter)	1
Fit Intercept	True
Intercept Scaling	1
Class Weight	Balanced
Solver	liblinear
Max Iterations	100
Multiclass	One-vs-Rest
Warm Start	False
Number of Jobs	1

6.3.4 Random Forest Classifier

For the Random Forest Classifier model, AutoML uses SciKit Learn's Random Forest Classifier class.

The model hyperparameters are detailed below:

Hyperparameter	Value
Number of Estimators	100
Criterion	Gini Impurity
Max Depth	None
Min Samples to Split	2
Min Samples to be Leaf Node	1

Hyperparameter	Value
Min Fraction of Total Sum of Weights to be Leaf	0
Max Features	Square root of number of features
Max Leaf Nodes	None
Min Impurity Decrease for Split	0
Bootstrap	True
Number of Jobs	1
Warn Start	False
Class Weight	Balanced

6.4 Model Selection Process

If the label column is of type float or complex, AutoML trains a regression model using XGBRegressor.

For classification models, AutoML uses the following selection process to determine the most accurate model:

1. If the dataset is too large, AutoML samples down the data to speed up the model selection process. The full dataset is still used for training after model selection.

   The size of the dataset is calculated by multiplying the number of columns by the number of rows. If this calculated size is larger than the target size, sampling is needed. The number of rows that can be utilized is calculated by dividing the target size by the number of columns. This number of rows is randomly selected from the entire dataset to be used *only* for the purposes of model selection.

2. AutoML determines if the dataset presents a binary classification problem, or if multiple classes are present.

   - If it is a binary classification problem, the ROC AUC scoring metric is used.
   - Otherwise, the F1 scoring metric is used.

3. These scoring metrics are then computed for each model using Monte Carlo cross validation, with three training/testing splits of 70%/30%, to determine the best model.

SQL Commands

ALTER ML CONFIGURATION

Modifies an ML configuration.

```
ALTER ML CONFIGURATION ml-configuration-name [ PROVIDER provider-name]
 [ %DESCRIPTION description] [ USING json-object-string ]
 [ provider-connection-settings ]
```

Arguments

ml-configuration-name	The name for the ML configuration being altered.
PROVIDER *provider-name*	A string specifying the name of a machine learning provider, where values are: • AutoML • H2O • DataRobot • PMML
%DESCRIPTION *description*	*Optional* — String. A text description for the ML configuration. See details below.
USING *json-object-string*	*Optional* — A JSON string specifying one or more key-value pairs; see details below.
provider-connection-settings	Any additional settings, required for connection, that vary by the machine learning provider. See details below.

Description

The **ALTER ML CONFIGURATION** statement alters one, or several, parameters within an ML configuration definition. You can alter:

- The provider
- The description
- The USING clause
- Provider connection settings

ML Configuration Description

%DESCRIPTION accepts a text string enclosed in single quotes, which you can use to provide a description for documenting your configuration. This text can be of any length, and can contain any characters, including blank spaces.

USING

You can specify a default **USING** clause for your configuration. This clause accepts a JSON string with one or more key-value pairs. When **TRAIN MODEL** is executed, by default the **USING** clause of the configuration is used.

```
ALTER ML CONFIGURATION MyConfiguration USING {"seed": 3}
```

You must make sure that the parameters you specify are recognized by the provider you select. Failing to do so may result in an error when training.

Provider Connection Settings

Depending on the provider specified by your configuration, there may be additional fields you must enter to establish a successful connection.

DataRobot

You must specify the following values to successfully connect to DataRobot:

- `URL [ = ]` *`url-string`* — where *`url-string`* is the URL of a DataRobot endpoint.
- `APITOKEN [ = ]` *`token-string`* — where *`token-string`* is your client API token to access the DataRobot AutoML server.

Altering an ML configuration for DataRobot could be performed with a query as follows:

```
ALTER ML CONFIGURATION datarobot-configuration URL url-string APITOKEN token-string
```

With proper values for `url-string` and `token-string`

Required Security Privileges

Calling **ALTER ML CONFIGURATION** requires %ALTER_ML_CONFIGURATION privileges; otherwise, there is a SQLCODE –99 error (Privilege Violation). To assign %ALTER_ML_CONFIGURATION privileges, use the GRANT command.

Examples

The following SQL query edits an existing configuration named `TestH2O` to add a **USING** clause that the user wants used for every model being trained:

```
ALTER ML CONFIGURATION TestH2O USING {"seed": 2}
```

See Also

- CREATE ML CONFIGURATION, DROP ML CONFIGURATION

ALTER MODEL

Modifies a model

```
ALTER MODEL model-name PURGE [ ALL ] [ integer DAYS ]
```

Or

```
ALTER MODEL model-name DEFAULT [ TRAINED MODEL ] trained-model-name
```

Arguments

model-name	The name of the machine learning model to alter.
DEFAULT *trained-model-name*	A trained machine learning model.
integer DAYS	An integer.

Description

An **ALTER MODEL** statement modifies a machine learning model. You can perform only one type of operation in each ALTER MODEL statement.

- A PURGE deletes all training runs and validation runs for the associated model based on the given scope:
  - If no scope is given, all records are deleted except for those associated with the default trained model.
  - If `integer DAYS` is given, all records older than `integer` days are deleted.
  - If `ALL` is given, all records are deleted regardless of when they occurred.
- A DEFAULT (or DEFAULT TRAINED MODEL) sets the default trained model to be the model specified. This is useful when you have made several **TRAIN MODEL** statements using the same model definition, saving each trained model to a different name, and you wish to switch which model the default name points to. Specifying a nonexistent model results in an error.

Required Security Privileges

Calling **ALTER MODEL** requires %MANAGE_MODEL privileges; otherwise, there is a SQLCODE –99 error (Privilege Violation). To assign %MANAGE_MODEL privileges, use the GRANT command.

Examples

The following query uses a PURGE clause to delete all training and validation run data for the `SpamFilter` model:

```
ALTER MODEL SpamFilter PURGE ALL
```

The following query uses a DEFAULT clause to change the default trained model of `SpamFilter` to `SpamFilter3`

```
ALTER MODEL SpamFilter DEFAULT SpamFilter3
```

See Also

- CREATE MODEL, DROP MODEL

CREATE ML CONFIGURATION

Creates an ML configuration.

```
CREATE ML CONFIGURATION ml-configuration-name PROVIDER provider-name
  [ %DESCRIPTION description] [ USING json-object-string ]
  [ provider-connection-settings ]
```

Arguments

ml-configuration-name	The name for the ML configuration being created. A valid identifier, subject to the same additional naming restrictions as a table name. An ML configuration name is unqualified (*mlconfiguration-name*). An unqualified ML configuration name takes the default schema name.
PROVIDER *provider-name*	A string specifying the name of a machine learning provider, where values are: • AutoML • H2O • DataRobot • PMML
%DESCRIPTION *description*	*Optional* — String. A text description for the ML configuration. See details below.
USING *json-object-string*	*Optional* — A JSON string specifying one or more key-value pairs; see details below.
provider-connection-settings	Any additional settings, required for connection, that vary by the machine learning provider. See details below.

Description

The **CREATE ML CONFIGURATION** command creates an ML configuration for training models. You can specify one or more of the following properties:

- The provider (required)
- The description
- The USING clause
- Provider connection settings

ML Configuration Description

%DESCRIPTION accepts a text string enclosed in single quotes, which you can use to provide a description for documenting your configuration. This text can be of any length, and can contain any characters, including blank spaces.

USING

You can specify a default **USING** clause for your configuration. This clause accepts a JSON string with one or more key-value pairs. When **TRAIN MODEL** is executed, by default the **USING** clause of the configuration is used.

You must make sure that the parameters you specify are recognized by the provider you select. Failing to do so may result in an error when training.

An example with H2O as the provider:

```
CREATE ML CONFIGURATION h2o_config PROVIDER H2O USING {"seed":100, "nfolds":4}
```

Provider Connection Settings

Depending on the provider specified by your configuration, there may be additional fields you must enter to establish a successful connection.

DataRobot

You must specify the following values to successfully connect to DataRobot:

- `URL [ = ] url-string` — where *`url-string`* is the URL of a DataRobot endpoint.
- `APITOKEN [ = ] token-string` — where *`token-string`* is your client API token to access the DataRobot AutoML server.

A complete ML configuration for DataRobot could be created with a query as follows:

```
CREATE ML CONFIGURATION datarobot-configuration PROVIDER DataRobot1 URL url-string APITOKEN token-string
```

With proper values for *`url-string`* and *`token-string`*

Required Security Privileges

Calling **CREATE ML CONFIGURATION** requires %CREATE_ML_CONFIGURATION privileges; otherwise, there is a SQLCODE –99 error (Privilege Violation). To assign %CREATE_ML_CONFIGURATION privileges, use the GRANT command.

Configuration Naming Conventions

Configuration names follow identifier conventions, subject to the restrictions below. By default, configuration names are simple identifiers. A configuration name should not exceed 256 characters. Configuration names are not case-sensitive. For further details, see the "Identifiers" chapter of *Using InterSystems SQL*.

InterSystems IRIS® uses the configuration name to generate a corresponding class name. A class name contains only alphanumeric characters (letters and numbers) and must be unique within the first 96 characters. To generate this class name, InterSystems IRIS first strips punctuation characters from the configuration name, and then generates an identifier that is unique within the first 96 characters, substituting an integer (beginning with 0) for the final character when needed to create a unique class name. InterSystems IRIS generates a unique class name from a valid configuration name, but this name generation imposes the following restrictions on the naming of configurations:

- A configuration name must include at least one letter. Either the first character of the view name or the first character after initial punctuation characters must be a letter
- InterSystems IRIS supports 16-bit (wide) characters for configuration names. A character is a valid letter if it passes the *$ZNAME* test.
- If the first character of the configuration name is a punctuation character, the second character cannot be a number. This results in an SQLCODE -400 error, with a %msg value of "ERROR #5053: Class name 'schema.name' is invalid" (without the punctuation character). For example, specifying the configuration name %7A generates the %msg "ERROR #5053: Class name 'User.7A' is invalid".
- Because generated class names do not include punctuation characters, it is not advisable (though possible) to create a configuration name that differs from an existing configuration name only in its punctuation characters. In this case, InterSystems IRIS substitutes an integer (beginning with 0) for the final character of the name to create a unique class name.
- A configuration name may be much longer than 96 characters, but configuration names that differ in their first 96 alphanumeric characters are much easier to work with.

A configuration name can only be unqualified. An unqualified configuration name (`viewname`) takes the system-wide default schema name.

Examples

```
CREATE ML CONFIGURATION autoML_config PROVIDER AutoML %DESCRIPTION 'my AutoML configuration!'
```

See Also

- ALTER ML CONFIGURATION, DROP ML CONFIGURATION

CREATE MODEL

Creates a model definition.

```
CREATE MODEL model-name PREDICTING ( label-column ) FROM model-source
  [ USING json-object ]
```

Or

```
CREATE MODEL model-name PREDICTING ( label-column ) WITH feature-column-clause
  [ USING json-object ]
```

Or

```
CREATE MODEL model-name PREDICTING ( label-column ) WITH feature-column-clause
 FROM model-source [ USING json-object ]
```

Arguments

This synopsis shows the valid forms of CREATE MODEL. The CREATE MODEL command must have either a FROM or WITH clause (or both).

model-name	The name for the model definition being created. A valid identifier, subject to the same additional naming restrictions as a table name. A model name is unqualified (`modelname`). An unqualified model name takes the default schema name.
PREDICTING (*label-column*)	The name of the column being predicted, aka, the label column. A standard identifier. See details below.
WITH *feature-column-clause*	Inputs to the model, aka the feature columns, as either the name of a column and it's datatype or as a comma-separated list of the names of columns and datatypes. Each column name is a standard identifier.
FROM *model-source*	The table or view from which the model is being built. This can be a table, view, or results of a join.
USING *json-object-string*	*Optional* — A JSON string specifying one or more key-value pairs. See more details below.

Description

The **CREATE MODEL** command creates a model definition of the structure specified. This includes, at a minimum:

- The model name
- The label column
- The feature column(s)

Predicting

You must specify the output column (or label column) that your model predicts, given the input columns (or feature columns). For example, if you are designing a `SpamFilter` model which identifies emails that are spam mail, you may have a label column named `IsSpam`, which is a boolean value designating whether a given email is spam or not. You can also specify the data type of this column; otherwise, IntegratedML infers the type:

```
CREATE MODEL SpamFilter PREDICTING (IsSpam) FROM EmailData
CREATE MODEL SpamFilter PREDICTING (IsSpam binary) FROM EmailData
```

WITH and FROM

A model definition must contain a **WITH** and/or **FROM** to specify the schema characteristics of the model.

WITH

Using **WITH**, you can specify which input columns (features) to include in your model definition. Unless you use **FROM** in your statement, you must also specify the data type of each column:

```
CREATE MODEL SpamFilter PREDICTING (IsSpam) WITH (email_length int, subject_title varchar)
CREATE MODEL SpamFilter PREDICTING (IsSpam) WITH (email_length, subject_title) FROM EmailData
```

FROM

FROM allows you to use every single column from a specified table or view, without having to identify each column individually:

```
CREATE MODEL SpamFilter PREDICTING (IsSpam) FROM EmailData
```

This clause is fully general, and can specify any subquery expression. IntegratedML infers the data types of each column. By using **FROM**, you supply a default data set for future **TRAIN MODEL** statements using this model definition. You can use **FROM** along with **WITH** to both supply a default data set and to explicitly name feature columns.

Without a **WITH** clause, IntegratedML infers the data types of each column, and implicitly uses the result of the **FROM** clause as if it were the following query:

```
SELECT * FROM model-source
```

USING

You can specify a default **USING** clause for your model definition. This clause accepts a JSON string with one or more key-value pairs. When **TRAIN MODEL** is executed, by default the **USING** clause of the model definition is used. All parameters specified in the **USING** clause of your ML configuration overwrite those same parameters in the **USING** clause of your model definition.

You must make sure that the parameters you specify are recognized by the provider you select. Failing to do so may result in an error when training.

Required Security Privileges

Calling **CREATE MODEL** requires %MANAGE_MODEL privileges; otherwise, there is a SQLCODE –99 error (Privilege Violation). To assign %MANAGE_MODEL privileges, use the GRANT command.

Model Naming Conventions

Model names follow identifier conventions, subject to the restrictions below. By default, model names are simple identifiers. A model name should not exceed 256 characters. Model names are not case-sensitive. For further details, see the "Identifiers" chapter of *Using InterSystems SQL*.

InterSystems IRIS uses the model name to generate a corresponding class name. A class name contains only alphanumeric characters (letters and numbers) and must be unique within the first 96 characters. To generate this class name, InterSystems IRIS first strips punctuation characters from the model name, and then generates an identifier that is unique within the first 96 characters, substituting an integer (beginning with 0) for the final character when needed to create a unique class name. InterSystems IRIS generates a unique class name from a valid model name, but this name generation imposes the following restrictions on the naming of models:

- A model name must include at least one letter. Either the first character of the view name or the first character after initial punctuation characters must be a letter
- InterSystems IRIS supports 16-bit (wide) characters for model names. A character is a valid letter if it passes the *$ZNAME* test.

- If the first character of the model name is a punctuation character, the second character cannot be a number. This results in an SQLCODE -400 error, with a %msg value of "ERROR #5053: Class name 'schema.name' is invalid" (without the punctuation character). For example, specifying the model name %7A generates the %msg "ERROR #5053: Class name 'User.7A' is invalid".
- Because generated class names do not include punctuation characters, it is not advisable (though possible) to create a model name that differs from an existing model name only in its punctuation characters. In this case, InterSystems IRIS substitutes an integer (beginning with 0) for the final character of the name to create a unique class name.
- A model name may be much longer than 96 characters, but model names that differ in their first 96 alphanumeric characters are much easier to work with.

A model name can only be unqualified. An unqualified model name (`viewname`) takes the system-wide default schema name.

Examples

```
CREATE MODEL PatientReadmit PREDICTING (IsReadmitted) FROM patient_table USING {"seed": 3}
CREATE MODEL PatientReadmit PREDICTING (IsReadmitted) WITH (age, gender, encounter_type, admit_reason,
 starttime, endtime, prior_visits, diagnosis, comorbitities)
```

See Also

- ALTER MODEL, DROP MODEL, TRAIN MODEL

DROP ML CONFIGURATION

Deletes an ML configuration.

```
DROP ML CONFIGURATION ml-configuration-name
```

Arguments

ml-configuration-name	The name of the ML configuration to delete.

Description

The **DROP ML CONFIGURATION** command deletes an ML configuration and its corresponding class definition.

Conditions

- The ML configuration must exist in the current namespace. Attempting to delete a non-existent ML configuration generates an SQLCODE –30 error.
- You cannot delete the system default ML configuration. Attempting to do so results in a SQLCODE –189 error.

Required Security Privileges

Calling **DROP ML CONFIGURATION** requires %DROP_ML_CONFIGRATION privileges; otherwise, there is a SQLCODE –99 error (Privilege Violation). To assign %DROP_ML_CONFIGRATION privileges, use the GRANT command.

See Also

- ALTER ML CONFIGURATION, CREATE ML CONFIGURATION

DROP MODEL

Deletes a model.

```
DROP MODEL model-name
```

Arguments

model-name	The name of the model to delete.

Description

The **DROP MODEL** command deletes a model and its corresponding class definition. It also purges any training runs and validation runs associated with the model.

Deleting a Non-Existent Model

The model must exist in the current namespace. Attempting to delete a non-existent model generates an SQLCODE —30 error.

Required Security Privileges

Calling **DROP MODEL** requires %MANAGE_MODEL privileges; otherwise, there is a SQLCODE –99 error (Privilege Violation). To assign %MANAGE_MODEL privileges, use the GRANT command.

See Also

- ALTER MODEL, CREATE MODEL

SET ML CONFIGURATION

Sets an ML configuration as the default.

```
SET ML CONFIGURATION ml-configuration-name
```

Arguments

ml-configuration-name	The name of the ML configuration.

Description

The **SET ML CONFIGURATION** command sets the specified ML configuration as the system default for all ensuing **TRAIN MODEL** statements. Only one ML configuration can be set as system default by each **SET ML CONFIGURATION** statement.

Required Security Privileges

Calling **SET ML CONFIGURATION** requires a USE object privilege; otherwise, there is a SQLCODE –99 error (Privilege Violation). You can determine if the current user has USE privilege by invoking the %CHECKPRIV command or the $SYSTEM.SQL.Security.CheckPrivilege() method.

Examples

```
CREATE MODEL H2OMODEL PREDICTING (label) FROM data
SET ML CONFIGURATION %H2O
TRAIN MODEL H2OMODEL
```

See Also

- ALTER ML CONFIGURATION, CREATE ML CONFIGURATION

TRAIN MODEL

Trains a machine learning model.

```
TRAIN MODEL model-name [ AS preferred-name ] [ NOT DEFAULT] [ FOR label-column ]
  [ WITH feature-column-clause ] [ FROM model-source ] [ USING json-object ]
```

Arguments

model-name	The name of the machine learning model to train.
AS *preferred-name*	*Optional* — An alternative name to save the trained model as. See details below.
NOT DEFAULT	*Optional* — A clause to train a model without setting it as the default trained model. See details below.
FOR *label-column*	*Optional* — The name of the column being predicted, aka, the label column. See details below.
WITH *feature-column-clause*	*Optional* — Inputs to the model, aka the feature columns, as either the name of a column or as a comma-separated list of the names of columns.
FROM *model-source*	The table or view from which the model is being built. This can be a table, view, or results of a join. See details below.
USING *json-object-string*	*Optional* — A JSON string specifying one or more key-value pairs. See details below.

Description

The **TRAIN MODEL** statement tells a provider to train a model using the specified model definition. The provider is specified by the ML configuration.

FROM

The **FROM** clause supplies the data for training your model.

- This clause is *required* if your **CREATE MODEL** statement did NOT specify a **FROM** clause.
- This clause is *optional* if your **CREATE MODEL** statement specified a **FROM** clause.

Examples highlighting acceptable use and omission of **FROM**:

FROM in TRAIN MODEL

```
CREATE MODEL model_b PREDICTING ( label ) WITH ( column_1, column_2, column_3)
TRAIN MODEL model_b FROM table
```

FROM in CREATE MODEL

```
CREATE MODEL model_a PREDICTING ( label ) FROM table
TRAIN MODEL model_a
```

Note: Omitting **FROM** from your **TRAIN MODEL** statement means that you use the default query from **CREATE MODEL**.

WITH

WITH allows you to explicitly match the feature columns in your data to the model definition schema. Each column is a standard identifier.

FOR

FOR allows you to explicitly match the label column in your data to the model definition schema. For example, if your label column in your model definition is named `column_a` but is named `column_b` in your training data, you can match the columns as follows:

```
CREATE MODEL model_a PREDICTING ( column_a ) FROM table_a
TRAIN MODEL model_a FOR column_b FROM table_b
```

Naming

AS allows you to explicitly name your trained model.

Model definitions and trained models exist in the same schema. If a trained model is not explicitly named with **AS**, its name consists of the model definition name with an appended running integer. We can see the difference by querying the INFORMATION_SCHEMA.ML_TRAINED_MODELS table:

```
CREATE MODEL TitanicModel PREDICTING (Survived binary) FROM IntegratedML_dataset_titanic.passenger
TRAIN MODEL TitanicModel
TRAIN MODEL TitanicModel
TRAIN MODEL TitanicModel
TRAIN MODEL TitanicModel AS TrainedTitanic
SELECT MODEL_NAME, TRAINED_MODEL_NAME FROM INFORMATION_SCHEMA.ML_TRAINED_MODELS
```

MODEL_NAME	TRAINED_MODEL_NAME
TitanicModel	TitanicModel_t1
TitanicModel	TitanicModel_t2
TitanicModel	TitanicModel_t3
TitanicModel	TrainedTitanic

Not Default

Each model definition has a default trained model. Without user-specification, the most recently trained model becomes the default. Using the **NOT DEFAULT** clause allows you to train a new model without the result becoming the default trained model:

```
CREATE MODEL TitanicModel PREDICTING (Survived) FROM IntegratedML_dataset_titanic.passenger
TRAIN MODEL TitanicModel As FirstModel
TRAIN MODEL TitanicModel As SecondModel NOT DEFAULT
SELECT MODEL_NAME, DEFAULT_TRAINED_MODEL_NAME FROM INFORMATION_SCHEMA.ML_MODELS
```

MODEL_NAME	DEFAULT_TRAINED_MODEL_NAME
TitanicModel	FirstModel

Without using **NOT DEFAULT**, the DEFAULT_TRAINED_MODEL field would otherwise read “SecondModel”

USING Clause Considerations

You can pass provider-specific parameters in a **USING** clause for a more customized training run. This clause accepts a JSON string with one or more key-value pairs. The list of parameters that you can use depends on the provider.

For instance, when training with AutoML as your provider you can change the random seed:

```
TRAIN MODEL IsSpam USING {"seed": 3}
```

See Providers for information about which parameters you can pass for each provider.

Passing NULL Values

Passing data with NULL values in the label column, in a **TRAIN MODEL** statement, will result in a trained model with undefined behavior. Users should carefully screen for NULL values as part of their data preparation process.

Required Security Privileges

Calling **TRAIN MODEL** requires %MANAGE_MODEL privileges; otherwise, there is a SQLCODE –99 error (Privilege Violation). To assign %MANAGE_MODEL privileges, use the GRANT command.

Examples

```
TRAIN MODEL EmailFilter
```

```
TRAIN MODEL model_5 AS MyModel USING {"seed": 3}
```

```
TRAIN MODEL LoanDefault FROM LoanData
```

See Also

- CREATE MODEL, VALIDATE MODEL

VALIDATE MODEL

Validates a model.

```
VALIDATE model-name [ AS validation-run-name ] [ USE trained-model-name]
 [ WITH feature-column-clause ] FROM model-source
```

Arguments

model-name	The name of a model to validate.
AS *validation-run-name*	*Optional* — A name to save your validation run as. See details below.
USE *trained-model-name*	*Optional* — The name of a non-default trained model to be validated. See details below.
WITH *feature-column-clause*	*Optional* — The specific columns from your dataset that you want to use for validating your model.
FROM *model-source*	The table or view from which the model is being validated. This can be a table, view, or results of a join. See details below.

Description

The **VALIDATE MODEL** command calculates validation metrics for a given trained model, based on its performance on a specified testing dataset. Each command creates a *validation run*.

Naming

AS allows you to explicitly name your validation run.

If a validation run is not explicitly named with AS, its name consists of the trained model with an appended running integer. We can see the difference by querying the INFORMATION_SCHEMA.ML_VALIDATION_RUNS table:

```
CREATE MODEL TitanicModel PREDICTING (Survived) FROM IntegratedML_dataset_titanic.passenger
TRAIN MODEL TitanicModel
VALIDATE MODEL TitanicModel FROM IntegratedML_dataset_titanic.passenger
VALIDATE MODEL TitanicModel FROM IntegratedML_dataset_titanic.passenger
VALIDATE MODEL TitanicModel FROM IntegratedML_dataset_titanic.passenger
VALIDATE MODEL TitanicModel AS TitanicValidation FROM IntegratedML_dataset_titanic.passenger
SELECT MODEL_NAME, TRAINED_MODEL_NAME, VALIDATION_RUN_NAME FROM INFORMATION_SCHEMA.ML_VALIDATION_RUNS
```

MODEL_NAME	TRAINED_MODEL_NAME	VALIDATION_RUN_NAME
TitanicModel	TitanicModel_t1	TitanicModel_t1_v1
TitanicModel	TitanicModel_t1	TitanicModel_t1_v2
TitanicModel	TitanicModel_t1	TitanicModel_t1_v3
TitanicModel	TitanicModel_t1	TitanicValidation

USE

USE allows you to specify the trained model to perform validation on. If a trained model is not explicitly named by **USE**, the statement validates the default trained model for the specified model definition.

We can see the difference by querying the INFORMATION_SCHEMA.ML_VALIDATION_RUNS table:

```
CREATE MODEL TitanicModel PREDICTING (Survived) FROM IntegratedML_dataset_titanic.passenger
TRAIN MODEL TitanicModel AS FirstModel
TRAIN MODEL TitanicModel AS SecondModel
TRAIN MODEL TitanicModel AS ThirdModel
VALIDATE MODEL TitanicModel FROM IntegratedML_dataset_titanic.passenger
VALIDATE MODEL TitanicModel FROM IntegratedML_dataset_titanic.passenger
VALIDATE MODEL TitanicModel USE FirstModel FROM IntegratedML_dataset_titanic.passenger
VALIDATE MODEL TitanicModel USE SecondModel FROM IntegratedML_dataset_titanic.passenger
SELECT MODEL_NAME, TRAINED_MODEL_NAME FROM INFORMATION_SCHEMA.ML_VALIDATION_RUNS
```

MODEL_NAME	TRAINED_MODEL_NAME
TitanicModel	ThirdModel
TitanicModel	ThirdModel
TitanicModel	FirstModel
TitanicModel	SecondModel

FROM Considerations

While you used a *training* set to train your model, you should use other data, a *testing* data set, to validate your model. Using your training data to validate a model only evaluates goodness of fit, as opposed to evaluating the model's predictive performance on other data.

This data should be of the same schema as your training data, including the feature columns and label column.

Required Security Privileges

Calling **VALIDATE MODEL** requires %USE_MODEL privileges; otherwise, there is a SQLCODE –99 error (Privilege Violation). To assign %USE_MODEL privileges, use the GRANT command.

Validation Metrics

The output of **VALIDATE MODEL** is a set of validation metrics that is viewable in the INFORMATION_SCHEMA_ML_VALIDATION_METRICS table.

For regression models, the following metrics are saved:

- Variance
- R-squared
- Mean squared error
- Root mean squared error

For classification models, the following metrics are saved:

- Precision — This is calculated by dividing the number of true positives by the number of predicted positives (sum of true positives and false positives).
- Recall — This is calculated by dividing the number of true positives by the number of actual positives (sum of true positives and false negatives).
- F-Measure — This is calculated by the following expression:

  ```
  F = 2 * (precision * recall) / (precision + recall)
  ```

- Accuracy — This is calculated by dividing the number of true positives and true negatives by the total number of rows (sum of true positives, false positives, true negatives, and false negatives) across the entire test set.

Examples

```
VALIDATE MODEL PatientReadmission FROM Patient_test
VALIDATE MODEL PatientReadmission AS PatientValidation USE PatientReadmission_H2OModel FROM Patient_test
```

See Also

- CREATE MODEL, TRAIN MODEL, PREDICT

SQL Functions

PREDICT

A function that applies a specified trained model to predict the result for each input row provided.

```
PREDICT( model-name)
```

Or

```
PREDICT( model-name USE  trained-model-name )
```

Or

```
PREDICT( model-name WITH feature-columns-clause )
```

Or

```
PREDICT( model-name USE  trained-model-name WITH feature-columns-clause )
```

Arguments

model-name	The name of the model.
USE *trained-model-name*	*Optional* — The name of a non-default trained model. See details below.
WITH *feature-columns-clause*	*Optional* — The specific columns to provide as input for your trained model. See details below.

Description

PREDICT returns the result of applying a trained machine learning model onto a specified query. This is performed on a row-by-row basis.

USE

If a trained model is not explicitly named by USE, **PREDICT** uses the default trained model for the specified model definition.

For example, if multiple models are trained:

```
CREATE MODEL MyModel PREDICTING( label ) FROM data
TRAIN MODEL MyModel AS FirstModel
TRAIN MODEL MyModel AS SecondModel NOT DEFAULT
```

`FirstModel` is the default model for `MyModel`. This means that **PREDICT** queries would use `FirstModel` for predictions. To specify use of `SecondModel`:

```
PREDICT( MyModel USE SecondModel)
```

WITH

PREDICT is a smart function, mapping the feature columns of the specified dataset to those in the model implicitly when there is no **WITH** clause. You can use a **WITH** clause to specify the mapping of columns between the dataset and your model. For example:

```
SELECT PREDICT(Trained_Model WITH age = year) FROM dataset
```

This query matches the `age` column from `Trained_Model` to the `year` column from `dataset`.

Required Security Privileges

Calling **PREDICT** requires %USE_MODEL privileges; otherwise, there is a SQLCODE –99 error (Privilege Violation). To assign %USE_MODEL privileges, use the GRANT command.

Examples

```
CREATE MODEL HousePriceModel PREDICTING( HousePrice ) FROM housing_data_2019
TRAIN MODEL HousePriceModel
SELECT * FROM housing_data_2020 WHERE PREDICT( HousePriceModel ) > 500000
```

```
CREATE MODEL PatientReadmission PREDICTING ( IsReadmitted ) FROM patient_data
TRAIN MODEL PatientReadmission
SELECT *, PREDICT( PatientReadmission ) FROM new_patient_data
```

See Also

- TRAIN MODEL, PROBABILITY

PROBABILITY

A function that applies a specified trained model to return the probability that the specified value is true for each input value provided.

```
PROBABILITY ( model-name FOR label-value)
```

Or

```
PROBABILITY ( model-name USE  trained-model-name FOR label-value )
```

Or

```
PROBABILITY ( model-name FOR label-value WITH  feature-columns-clause )
```

Or

```
PROBABILITY ( model-name USE  trained-model-name FOR label-value
 WITH feature-columns-clause ] )
```

Arguments

model-name	The name of the trained model.
FOR *label-value*	The output value. See details below.
USE *trained-model-name*	*Optional* — The name of a non-default trained model. See details below.
WITH *feature-columns-clause*	*Optional* — The specific columns to provide as input for your trained model. See details below.

Description

The **PROBABILITY** function applies a given model to a given table, returning the probability that, for each row in the table, the model would predict the specified value. This probability is returned as a value from 0 to 1. This function can only be used with classification models (not regression models).

FOR

FOR provides the output value that **PROBABILITY** finds the probability of.

For example:

```
SELECT * FROM flower_dataset WHERE PROBABILITY(iris_flower FOR 'iris-setosa') > 0.6
```

Uses the `iris_flower` model to return each row in `flower_dataset` where the probability of the result being “iris-setosa” is greater than 0.6.

Omitting **FOR** implies a value of 1. For example:

```
SELECT PROBABILITY(IsSpam) FROM email_data
```

Implicitly forms this query:

```
SELECT PROBABILITY(IsSpam FOR 1) FROM email_data
```

When the value provided for **FOR** is invalid for the specified trained model, there is a SQLCODE –400 error with the following message:

```
[%msg: <PREDICT execution error: ERROR #2853: Specified positive label value not found
in the dataset.>]
```

USE

If a trained model is not explicitly named by USE, **PROBABILITY** uses the default trained model for the specified model definition.

For example, if multiple models are trained:

```
CREATE MODEL MyModel PREDICTING( label ) FROM data
TRAIN MODEL MyModel AS FirstModel
TRAIN MODEL MyModel AS SecondModel NOT DEFAULT
```

`FirstModel` is the default model for `MyModel`. This means that **PROBABILITY** queries would use `FirstModel` for predictions. To specify use of `SecondModel`:

```
PROBABILITY( MyModel FOR label-value USE SecondModel)
```

WITH

PROBABILITY is a smart function, mapping the feature columns of the specified dataset to those in the model implicitly when there is no **WITH** clause. You can use a **WITH** clause to specify the mapping of columns between the dataset and your model. For example:

```
SELECT PROBABILITY(iris_flower FOR 'iris-setosa' WITH petal_length = length_petal) FROM flower_dataset
```

This query matches the `petal_length` column from the `iris_flower` model to the `length_petal` column from `flower_dataset`.

Required Security Privileges

Calling **PROBABILITY** requires %USE_MODEL privileges; otherwise, there is a SQLCODE –99 error (Privilege Violation). To assign %USE_MODEL privileges, use the GRANT command.

Examples

```
CREATE MODEL PatientReadmission PREDICTING ( IsReadmitted ) FROM patient_data
TRAIN MODEL PatientReadmission
SELECT * FROM new_patient_data WHERE PROBABILITY( PatientReadmission FOR 1) > 0.8
```

See Also

- TRAIN MODEL, PREDICT