Using IntegratedML

Version 2023.3
2024-05-16

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

Using IntegratedML
InterSystems IRIS Data Platform Version 2023.3 2024-05-16

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

1

Introduction to IntegratedML

IntegratedML is a feature within InterSystems IRIS® data platform 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 Introduction to Machine Learning

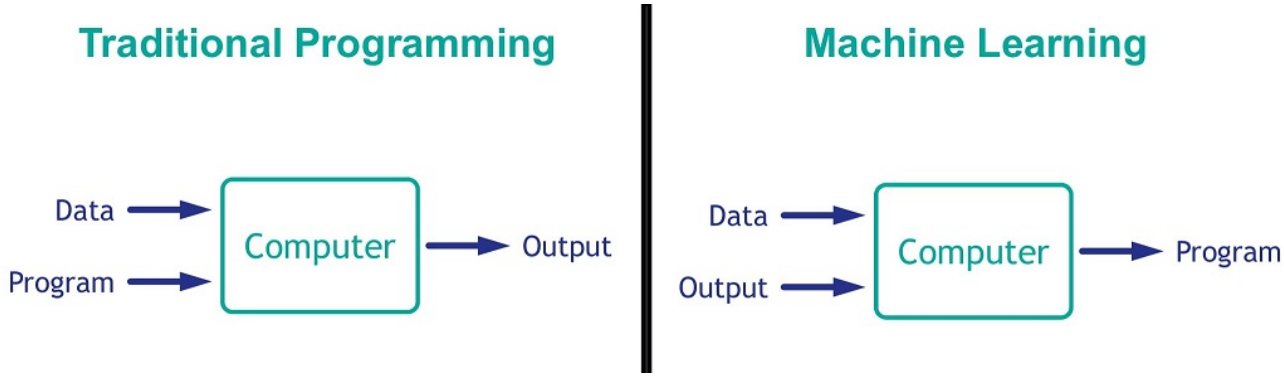
To understand IntegratedML, you need an introductory understanding of several commonly used terms:

- Machine learning
- Models
- Regression versus classification
- 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 contains the same features as the training and testing data, but without the label column; the label is the output of the model.

Regression versus Classification

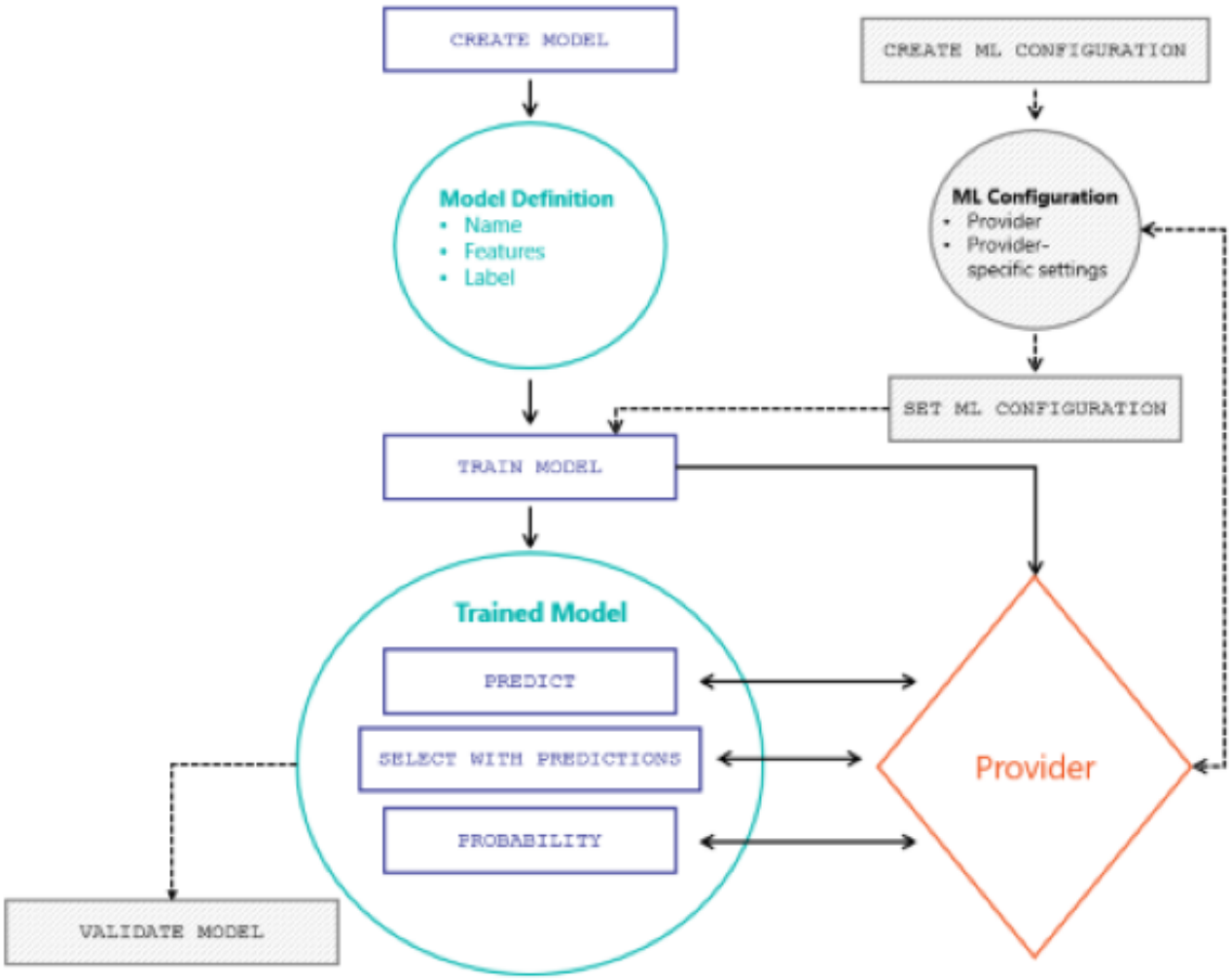
A regression model is used to predict continuous numeric values, such as cost, lab result, and so on, and may therefore output a label value (for example, $12.52) that does not appear in the training data. A classification model is used to predict discrete values such as true/false, country name, and so on, where possible label values are defined by those that appear in the training data; for example, if the label is country name, predicted values are restricted to the country names that actually appear in the training data. When using a classification model you can also output the probability of a specified value being the label value for each prediction, allowing you to evaluate the relative strength of predictions of that value.

2

IntegratedML Basics

IntegratedML is a feature within InterSystems IRIS® data platform 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. Only the structure of the data is stored within the model definition, not the data itself.

- *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 in the active 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. This process varies by the provider.
   - *Optional* — After training the model, 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 computational 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.

IntegratedML can be used to create three kinds of models: regression, classification, and time series. The syntax for creating models for regression or classification tasks differs from the syntax for creating models for time series tasks. You can refer to these varying syntaxes on the **CREATE MODEL** reference page.

2.1.1 Examples — CREATE MODEL

The following examples highlight use of different clauses for your **CREATE MODEL** statements. The first three examples outline various options when creating a regression or classification model, while the last example shows how to create a time series model.

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 for a classification or regression model, **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 Adding Training Parameters (the USING clause) for further discussion of the **USING** clause.

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

Creating a Time Series Model

As opposed to the other examples, which create models used for regression or classification tasks, this example creates a time series model that predicts subsequent rows at 3 steps into the future.

```
CREATE TIME SERIES MODEL ForecastModel PREDICTING (*) BY (date) FROM WeatherData USING
{"FORWARD":3 }
```

2.1.2 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 values for several rows, or contains 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.

2.1.3 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 InterSystems SQL Reference.

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.

Refer to the **TRAIN MODEL** reference page for a syntax overview and full description.

2.2.1 Examples — TRAIN MODEL

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 Adding Training Parameters (the USING clause) for further discussion of the **USING** clause.

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

2.2.2 Adding Training Parameters (the USING clause)

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.

For example:

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

You can use the **USING** clause to pass provider-specific training parameters. This clause accepts a JSON string containing key-value pairs of parameter names and parameter values. These value pairs 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 below for information about the parameters available to each of the following providers:

- AutoML
- H2O
- DataRobot

2.2.3 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 InterSystems SQL Reference.

2.3 Validating Models

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

Refer to the **VALIDATE MODEL** reference page for a syntax overview and full description.

2.3.1 Examples — VALIDATE MODEL

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
```

2.3.2 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 InterSystems SQL 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). A row-set can be any set of rows that includes the required feature columns and the label column.

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
```

You can use the **WITH** clause to make a prediction based on specified values corresponding to columns in the trained model, rather than a full dataset. Arguments must be ordered exactly as specified in your **CREATE MODEL** statement; missing arguments can be indicated by empty commas. For example, the following statement makes predictions based on two sets of column data:

```
SELECT PREDICT(HousePriceModel WITH ({4200, 5, 4},{3800, , 3}))
```

You can also use **WITH** to specify the mapping of columns between the model and a dataset different from the one it was created on. The following statement maps three feature columns from the model to columns in the alternate dataset:

```
SELECT PREDICT(HousePriceModel WITH (TotSqft = house_area, num_beds = beds, num_baths = bathrooms) FROM
 OtherHouseData
```

See More

For complete information about the **PREDICT** function, see the InterSystems SQL Reference.

2.4.2 PROBABILITY

When using classification models, which predict discrete values such as true/false or country name (as opposed to regression models, which predict continuous numeric values such as cost or lab result), you can use the **PROBABILITY** function to return for each row the probability that the specified label value is the predicted label value. This allows you to evaluate the relative strength of predictions of that value.

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 `Iris_mdoel` 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 InterSystems SQL Reference.

2.4.3 SELECT WITH PREDICTIONS

Since time series models predict rows of data, rather than specific elements, you must execute a **SELECT** statement to see a model's predictions.

Syntax

A **SELECT WITH PREDICTIONS** command has the following syntax:

```
SELECT WITH PREDICTIONS(model-name) columns FROM table-name
```

Examples

The following example selects all rows and columns from a table:

```
SELECT WITH PREDICTIONS(WeatherModel) * FROM Weather.Data
```

The following example selects only the rows after a certain date, allowing you to view a section of the table, such as the set of predicted rows:

```
SELECT WITH PREDICTIONS(WeatherModel) * FROM Weather.Data WHERE Date > 20220101
```

See More

For complete information about selecting with predictions, see the InterSystems SQL 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® data platform 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 Adding Training Parameters (the USING clause) for further discussion.

3.1 AutoML

AutoML is an automated machine learning system developed by InterSystems, housed within InterSystems IRIS® data platform. 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.

3.1.1 Training Parameters — AutoML

You can pass training parameters with a **USING** clause. For example:

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

With AutoML, you can pass the following parameters into your training queries:

Training Parameter	Description
`seed`	A seed to initialize the random number generator. You can manually set any integer as the seed for reproducibility between training runs. By default, `seed` is set to `"None"`.
`verbosity`	Determines how verbose the output of each training run is. This output can be found in the ML_TRAINING_RUNS view. You can specify any of the following options for `verbosity`: • `0` — Minimal/no output. • `1` — Moderate output. • `2` — Full output. This is the default setting for `verbosity`.
`TrainMode`	Determines the model selection metric for classification models. You can specify one of the following options for `TrainMode`: • `"TIME"` — Model selection prioritizes faster training time. • `"BALANCE"` — Model selection compares models by an equal proportion of each model's respective score and training time. • `"SCORE"` — Model selection does not factor training run time at all. This is the default setting for `TrainMode`. See the *AutoML Reference* for more information about these different modes.
`MaxTime`	The number of minutes allotted for initiating training runs. This does not necessarily limit training time. For example, if the `MaxTime` is set to 3000 minutes and there are 2 minutes remaining after a model is trained, another model could still be trained. By default, `MaxTime` is set to `14400` minutes. **Note:** This parameter is only applicable if `TrainMode` is set to `"TIME"`.
`MinimumDesiredScore`	The minimum score to allow for classification model selection, irrespective of the training mode selected. You can set any value between 0 and 1. By default, `MinimumDesiredScore` is set to 0. **Note:** This parameter is only applicable if `TrainMode` is set to `"TIME"`. If the trained logistic regression or random forest classifier model exceeds the `MinimumDesiredScore`, then AutoML does not train the neural network model. See the *AutoML Reference* for more information about the different models used for classification models.

3.1.2 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.

3.1.3 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.

Note: A more detailed description of this model selection process can be found in the *AutoML Reference*.

3.1.4 Platform Support and Known Issues

The AutoML provider is not supported on any IBM AIX® platform, Red Hat Enterprise Linux 8 for ARM, or Ubuntu 20.04 for ARM.

AutoML is implemented using Python, which may lead to improper isolation between AutoML Python packages and Embedded Python packages. As a result, AutoML may be unable to find packages it needs to work correctly. To avoid this issue, add <path to instance>/lib/automl to the Python *sys.path* within your instance of InterSystems IRIS. To do so, open a Python shell with %SYS.Python.Shell() and enter the following commands:

```
import sys
sys.path.append("<path to instance>\\lib\\automl")
```

3.1.5 See More

For more information about how AutoML works, see the *AutoML Reference*.

3.2 H2O

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.

Note: The H2O provider does not support the creation of time series models.

3.2.1 Training Parameters — H2O

You can pass training parameters with a **USING** clause. For example:

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

See the H2O documentation for information regarding expected input and how these parameters are handled. Unknown parameters result in an error during training.

When training a model using the H2O provider, the `max_models` parameter is set to 5 by default.

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. For example:

```
TRAIN MODEL h2o-model USING {"model_type": "regression"}
```

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 in to the Management Portal.
  2. Go to **System Administration** > **Configuration** > **Connectivity** > **External Language Servers**.
  3. Select the server named **%IntegratedML Server**.
  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.

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

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 Training Parameters — DataRobot

You can pass training parameters with a **USING** clause. For example:

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

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.

When training a model using the DataRobot provider, the `quickrun` parameter is set to `true` by default.

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.

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 in to 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 Adding Training Parameters (the USING clause) 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 InterSystems SQL 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 InterSystems SQL 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 in to 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 process.

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 INFORMATION_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 in to 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 InterSystems SQL 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 in to 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 InterSystems SQL Reference.

5

Model Maintenance

Model maintenance consists of viewing, altering, and 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, regression, or time series)
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

For information about the validation metrics that populate METRIC_NAME and METRIC_VALUE, see the InterSystems SQL Reference.

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 refernece.

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 InterSystems SQL Reference.