NF 25

CONNECTING THE WORLD

NF 25

CONNECTING THE WORLD

NIAGARA FORUM APAC | 9-10 SEPTEMBER 2025

Developing
Machine Learning Apps
in Niagara

Tong Zhong

Sr Appl/Sys Sales Engineer

Tridium

INTRO

NIAGARA FORUM
CONNECTING THE WORLD

Machine Learning

Traditional Machine Learning

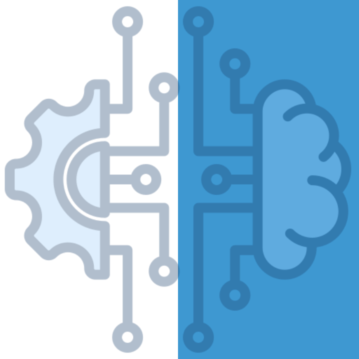
Wide Application
Used for various tasks such as classification, regression, and clustering.

Diverse Algorithms
Includes linear regression, decision trees, support vector machines, random forests, etc.

Data Type
Primarily handles structured data.

Feature Engineering
Relies on manual feature selection and extraction.

Model Size
Typically smaller, with relatively low computational resource requirements.



Large Language Models

Focus on NLP
Mainly used for natural language processing tasks such as text generation, translation, and question-answering systems.

Based on Deep Learning
Often uses neural network architectures, especially Transformers.

Data Type
Handles unstructured data, primarily text.

Automatic Feature Learning
Learns features autonomously from large datasets, without manual feature engineering.

Large Model Size
Requires substantial data and computational resources for training.

Traditional Machine Learning

- **Data Driven Statistical Modeling**

  By manually designing features (such as signal frequency and error curves), algorithms (like SVM, Random Forests, or Reinforcement Learning) are used to extract patterns from the data.

- **Advantages**

  Strong interpretability, high real-time performance, and low resource dependency.

- **Scenarios:**

Adaptive Control

Adaptive Controller Design: Use machine learning algorithms to adjust controller parameters in real-time, adapting to environmental changes and system dynamics.

Model Predictive Control (MPC): Combine machine learning to predict future system states, enhancing the accuracy of control strategies.

Fault Detection & Diagnosis

Anomaly Detection: Use machine learning models to identify abnormal patterns in system operations, providing early warnings of potential faults.

Fault Classification: Automatically classify and diagnose fault types by learning from historical fault data.

Process Optimization

Parameter Optimization: Use machine learning to optimize industrial process parameters, enhancing production efficiency and product quality.

Energy Management: Utilize machine learning to analyze energy consumption data and optimize energy usage strategies.

- **Bottlenecks**

  Reliance on expert-designed features and difficulty in unstructured data (such as natural language commands) process.

Machine Learning in Niagara

TRIDIUM

Supervisor

powered by niagara framework®

History Data

Model File

1. Establish algorithm models based on equipment characteristics.
2. Train the model using historical data.
3. Update the model in JACE.

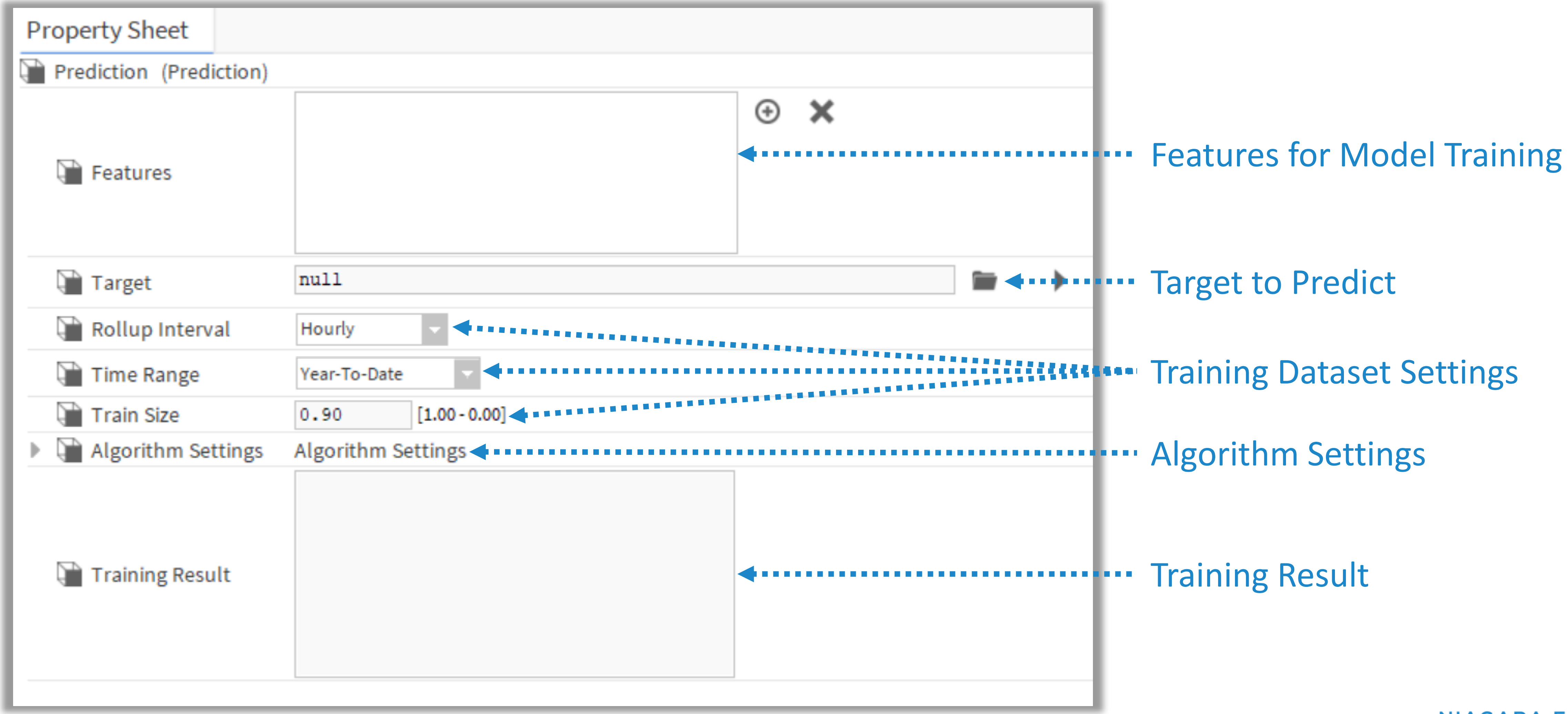
Algorithm Engine Module

1. Collect device data.
2. Predict future energy consumption based on the model.
3. Execute control strategies based on future energy consumption and thresholds.

NIAGARA FORUM
CONNECTING THE WORLD

PRACTICE

Design



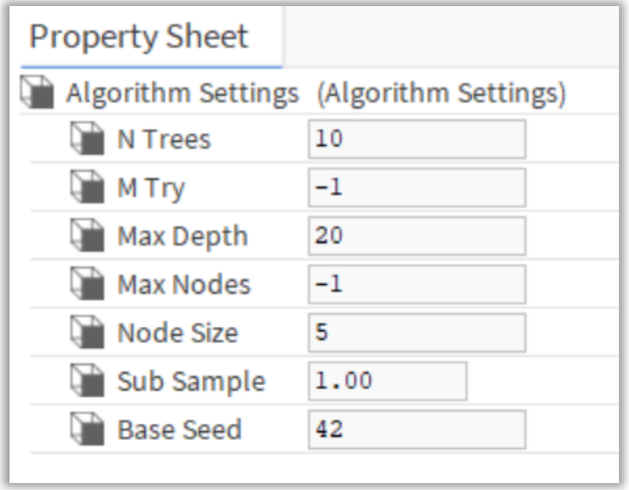
Property Sheet
Prediction (Prediction)
Features
Target
null
Rollup Interval
Hourly
Time Range
Year-To-Date
Train Size
0.90
[1.00 - 0.00]
Algorithm Settings
Algorithm Settings
Training Result
Features for Model Training
Target to Predict
Training Dataset Settings
Algorithm Settings
Training Result

Design

Random Forest is an ensemble learning method primarily used for classification and regression tasks. It operates by constructing multiple decision trees during training and outputs the mode of the classes (for classification) or mean prediction (for regression) of the individual trees. Key features include:

- Ensemble of Trees: Combines the predictions of multiple decision trees to improve accuracy and control overfitting.
- Random Subsets: Each tree is trained on a random subset of the data and a random subset of features, which helps in making the model robust and reducing variance.
- Non-linear Relationships: Capable of capturing complex, non-linear relationships between features.
- Feature Importance: Provides insights into feature importance, helping to understand which features contribute most to predictions.

Random Forest is popular due to its simplicity, robustness, and effectiveness in handling high-dimensional data.



Design

Slot Sheet

Slot	#	Name	Display Name	Definition	Flags	Type
Property	0	features	Features	Frozen		baja:OrdList
Property	1	target	Target	Frozen		baja:Ord
Property	2	rollupInterval	Rollup Interval	Frozen		history:RollupInterval
Property	3	timeRange	Time Range	Frozen		bql:DynamicTimeRange
Property	4	trainSize	Train Size	Frozen		baja:Double
Property	5	algorithmSettings	Algorithm Settings	Frozen		machineLearning:AlgorithmSettings
Property	6	trainingResult	Training Result	Frozen	t	baja:String
Action	7	train	Train	Frozen		void (void)
Action	8	predict	Predict	Frozen		void (baja:String)
Topic	9	predicted	Predicted	Frozen		baja:Double

- Train: An action to train the model using selected dataset and preset parameters
- Predict: An action to predict with a JSON String input using the trained model
- Predicted: A topic to fire with the predicted value

3rd ML Party Library

- **Apache Spark MLlib** *https://spark.apache.org/mllib/*

  MLlib is Apache Spark's scalable machine learning library. Unfortunately, due to technical limitations (the excessive content of the Spark MLlib library), it is **not possible** to apply Apache Spark MLlib in Niagara.

- **Weka** *https://waikato.github.io/weka-wiki/*

  Weka is a collection of machine learning algorithms for solving real-world data mining problems. It is written in Java and runs on almost any platform. However, **WEKA is licensed under the GNU General Public license** (GPL 2.0 for Weka 3.6 and GPL 3.0 for Weka > 3.7.5).

- **Smile** *https://github.com/haifengl/smile*

  Smile is a fast and comprehensive machine learning framework in Java. It can be included in Niagara’s module and built successfully. **It is important to note that Smile switched to GPLv3 licensing after version 3, but version 2 used the LGPL licensing.**

NIAGARA FORUM
CONNECTING THE WORLD

Development

1. Add dependency for smile-core 2.6.0

```
uberjar("com.github.haifengl:smile-core:2.6.0")
```

2. Define the Prediction component

```
@NiagaraType
@NiagaraProperty(name = "features", type = "BOrdList", defaultValue = "BOrdList.DEFAULT", facets = {@Facet(name = "BFac
@NiagaraProperty(name = "target", type = "BOrd", defaultValue = "BOrd.DEFAULT")
@NiagaraProperty(name = "rollupInterval", type = "BRollupInterval", defaultValue = "BRollupInterval.hourly")
@NiagaraProperty(name = "timeRange", type = "BDynamicTimeRange", defaultValue = "BDynamicTimeRange.YEAR_TO_DATE")
@NiagaraProperty(name = "trainSize", type = "double", defaultValue = "0.9D", facets = {@Facet(name = "BFacets.MAX", val
@NiagaraProperty(name = "algorithmSettings", type = "BAlgorithmSettings", defaultValue = "new BAlgorithmSettings()")
@NiagaraProperty(name = "trainingResult", type = "String", defaultValue = "\"\"", flags = Flags.TRANSIENT, facets = {@F
@NiagaraAction(name = "train")
@NiagaraAction(name = "predict", parameterType = "BString", defaultValue = "BString.DEFAULT")
@NiagaraTopic(name = "predicted", eventType = "BDouble")
public class BPrediction extends BComponent
{
```

Development - Train

1. Collect Training Data of Features

```
//---------------------------------------------------------------
// 1. prepare training data from Niagara history data
// 1.1. get feature data
getFeatures().forEach(f -> {

  BHistory history = (BHistory) f.get(this);
  String featureName = history.getId().getHistoryName();

  // the query format would be:
  // history:/<STATION>/<HISTORY_ID>?period=<RANGE>|bql:history:HistoryRollup.rollup(history:RollupInterval '<INTERVAL>')
  String query = String.format("%s?%s|bql:history:HistoryRollup.rollup(history:RollupInterval '%s')",
    f.toString(cx),
    getTimeRange().toOrdParams(),
    getRollupInterval().getTag()
  );

  BITable<BHistoryRollupRecord> records = (BITable<BHistoryRollupRecord>) BOrd.make(query).get(this);
  try (TableCursor<BHistoryRollupRecord> cursor = records.cursor())
  {
    featureValues.put(featureName, cursor.stream().mapToDouble(BHistoryRollupRecord::getAvg).toArray());
  }

});
```

Development - Train

2. Collect Training Data of Target

```
// 1.2. get target data
BHistory targetHistory = (BHistory) getTarget().get(this);
String targetName = targetHistory.getId().getHistoryName();

// the query format would be:
// history:/<STATION>/<HISTORY_ID>?period=<RANGE>|bql:history:HistoryRollup.rollup(history:RollupInterval '<INTERVAL>')
String query = String.format("%s?%s|bql:history:HistoryRollup.rollup(history:RollupInterval '%s')",
  getTarget().toString(cx),
  getTimeRange().toOrdParams(),
  getRollupInterval().getTag()
);

BITable<BHistoryRollupRecord> records = (BITable<BHistoryRollupRecord>) BOrd.make(query).get(this);
double[] targetValues;
try (TableCursor<BHistoryRollupRecord> cursor = records.cursor())
{
  // we take sum for energy consumption here
  targetValues = cursor.stream().mapToDouble(BHistoryRollupRecord::getSum).toArray();
}
```

NIAGARA FORUM
CONNECTING THE WORLD

Development - Train

3. Transform Niagara History Data to Smile Data Frame (Fit & Test)

```
//----------------------------------------------------------------
// 2. transform Niagara history data to SMILE
// 2.1. convert feature values
List<String> featureNames = new ArrayList<>();
List<DoubleVector> vectorList = new ArrayList<>();
featureValues.forEach((k, v) -> {
  featureNames.add(k);
  vectorList.add(DoubleVector.of(k, v));
});

// 2.2. convert target value
vectorList.add(DoubleVector.of(targetName, targetValues));

DoubleVector[] vectors = vectorList.toArray(new DoubleVector[0]);
DataFrame dataFrame = DataFrame.of(vectors);

//----------------------------------------------------------------
// 3. split the data set into 2 for training and testing
DataFrame trainingData = dataFrame.slice(0, (int) (dataFrame.nrows() * getTrainSize()));
DataFrame testingData = dataFrame.slice((int) (dataFrame.nrows() * getTrainSize()), dataFrame.nrows());
```

Development - Train

4. Train the Model

```
//-----------------------------------------------------------
// 4. train the model
BAlgorithmSettings settings = getAlgorithmSettings();
model = RandomForest.fit(
  Formula.of(targetName, featureNames.toArray(new String[]{})),
  trainingData,
  settings.getNTrees(),
  settings.getMTry() < 0 ? featureNames.size() : settings.getMTry(),
  settings.getMaxDepth() < 0 ? Integer.MAX_VALUE : settings.getMaxDepth(),
  settings.getMaxNodes() < 0 ? Integer.MAX_VALUE : settings.getMaxNodes(),
  settings.getNodeSize(),
  settings.getSubSample(),
  LongStream.range(0, settings.getNTrees()).map(i -> settings.getBaseSeed() + i)
);
```

Development - Train

5. Test the Model

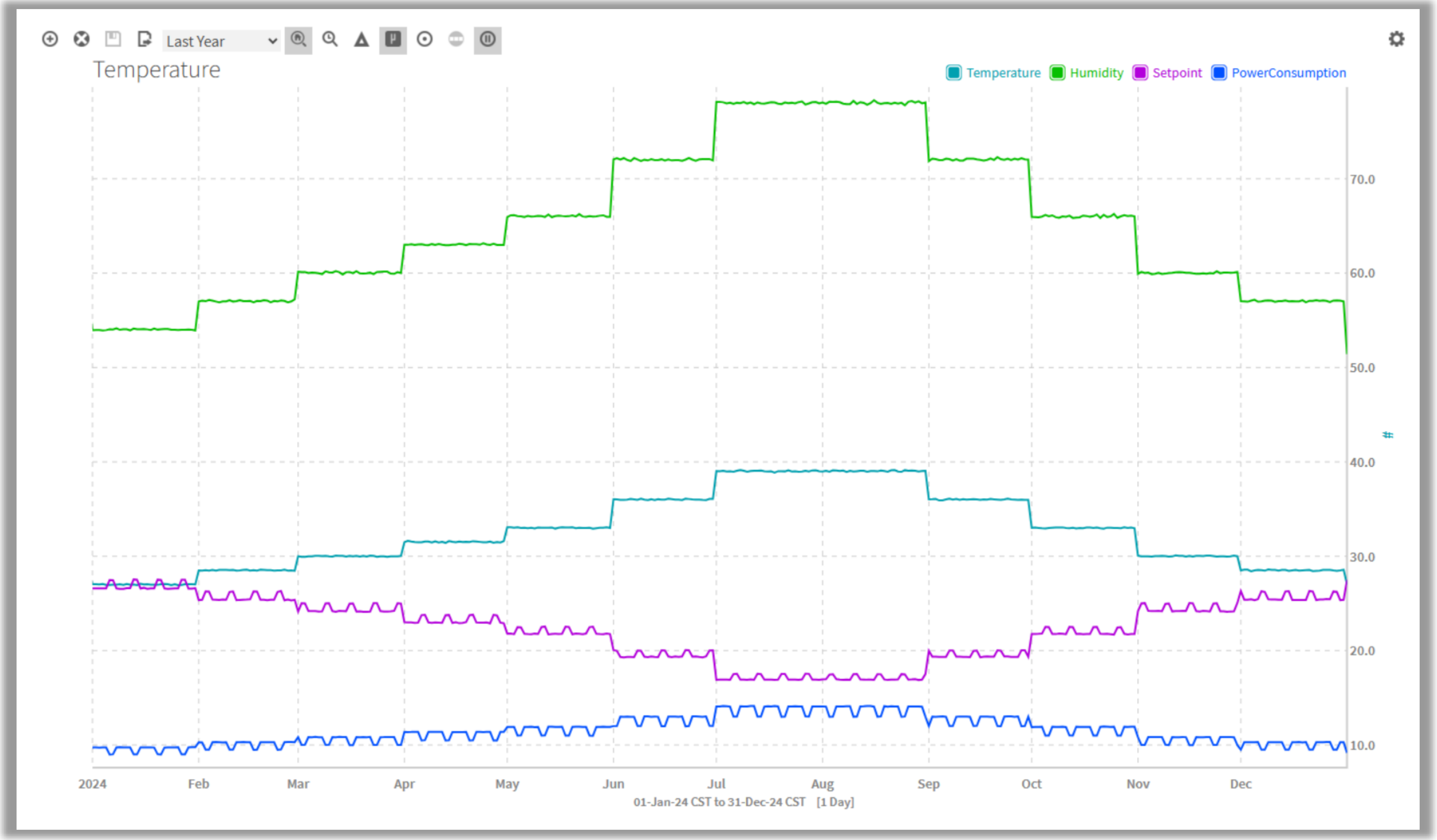
```java
//------------------------------------------------------------
// 5. test the model
DataFrame x_test = testingData.drop(targetName);
double[] y_test = testingData.column(targetName).toDoubleArray();
double[] y_test_result = model.predict(x_test);

double test_r2 = R2.of(y_test, y_test_result);
double test_mse = MSE.of(y_test, y_test_result);
double test_mae = MAD.of(y_test, y_test_result);

String trainingResult = String.format(
  "Model has been trained, test result is: \nR2: %f \nMES: %f \nMAE: %f",
  test_r2, test_mse, test_mae);
LOGGER.log(Level.INFO, trainingResult);
setTrainingResult(trainingResult);
```

Development - Predict

```java
// predict using the features in the input
JSONObject features = new JSONObject(param.getString());
ArrayList<String> featureNames = new ArrayList<>();
ArrayList<Double> featureValues = new ArrayList<>();

features.keys().forEachRemaining(
  k -> {
    featureNames.add(String.valueOf(k));
    featureValues.add(features.getDouble(String.valueOf(k)));
  }
);

List<StructField> fields = new ArrayList<>(featureNames.size());
for (String featureName : featureNames)
{
  StructField field = new StructField(featureName, DataTypes.DoubleType);
  fields.add(field);
}

StructType structType = new StructType(fields);
Tuple tuple = Tuple.of(featureValues.stream().mapToDouble(Double::doubleValue).toArray(), structType);
double predictValue = model.predict(tuple);

LOGGER.log(Level.INFO, msg: "Predicted value is: {0}", String.valueOf(predictValue));
firePredicted(BDouble.make(predictValue));
```


DEMO
NIAGARA FORUM
CONNECTING
THE WORLD

Demo



NIAGARA FORUM
CONNECTING THE WORLD

KEY POINTS

Key Points

- Performance
- Compatibility with Java Compact3 Profile
- Open-Source License for Commercial Use
- Criteria for Evaluating Model Quality
- How to Get the Future Parameters for Prediction
- How to Use the Predictive Data


THANK YOU
NF
25
CONNECTING
THE WORLD