Skill ML.NET Regression

Predict a continuous value from multiple input tags using ML.NET FastTree Regression. The AI generates the full C# Script Class with multi-feature training pipeline, connects it to live tags, creates output tags, and configures model persistence.

AI Integration Platform Skills Library Skill ML.NET Skill ML.NET Regression

- When to Use This Skill
- Prerequisites
- MCP Tools and Tables
- Step 1: Create Output Tags
- Step 2: Create the Script Class
  - ML.NET Namespace Declaration
  - Tag References Inside Script Classes
  - FastTree Regression Pipeline
  - Full Class Example — Regression
- Step 3: Write the Class via MCP
- Step 4: Create the Trigger
  - Task (Periodic) — best for multi-input models
  - ServerStartup — always wire LoadModel
- Step 5: Verify
- Common Pitfalls
- See also

When to Use This Skill

- User chose **Regression — FastTree** in the ML.NET router skill (Step 0)
- Predicting a **continuous value** from **2–5 input features**
- User goals: energy consumption prediction, process output modeling, production rate from multiple inputs

Prerequisites

- Solution open with input feature tags and label tag already created and receiving data
- Solution target platform set to **Multiplatform** (ML.NET requires .NET 8+)
- The user has confirmed: (1) which tags are features, (2) which tag is the label (value to predict), (3) what the predicted value represents

MCP Tools and Tables

Category	Items
Tools	`get_table_schema`, `write_objects`, `get_objects`
Tables	`UnsTags`, `ScriptsClasses`, `ScriptsTasks`

Step 1: Create Output Tags

```
get_table_schema('UnsTags')
```

```
{
  "table_type": "UnsTags",
  "data": [
    { "Name": "<AssetPath>/ML/PredictedValue", "DataType": "Double", "Description": "Model predicted value"
},
    { "Name": "<AssetPath>/ML/LastPrediction", "DataType": "DateTime", "Description": "Timestamp of last
prediction" }
  ]
}
```

Replace `<AssetPath>` with the actual asset folder path (e.g., `Plant/Reactor1`).

Step 2: Create the Script Class

ML.NET Namespace Declaration

> **Critical:** The field `AddMLNetNamespaces` does **not exist** and is silently ignored. Always use `NamespaceDeclarations`.

```
"NamespaceDeclarations": "Microsoft.ML;Microsoft.ML.Data;Microsoft.ML.Transforms;Microsoft.ML.Transforms.
TimeSeries;Microsoft.ML.Transforms.Text;Microsoft.ML.Trainers;Microsoft.ML.TimeSeries"
```

Tag References Inside Script Classes

Always use `@Tag.` prefix. ML.NET expects `float` but FrameworX tags use `double` — always cast with `(float)` when feeding ML.NET and `(double)` when writing to tags.

FastTree Regression Pipeline

```
var pipeline = mlContext.Transforms.Concatenate("Features", "Feature1", "Feature2", "Feature3")
    .Append(mlContext.Regression.Trainers.FastTree(
        labelColumnName: "Label",
        featureColumnName: "Features",
        numberOfLeaves: 20,
        numberOfTrees: 100,
        minimumExampleCountPerLeaf: 10,
        learningRate: 0.2));
```

Output: `float Score` (predicted continuous value)

Full Class Example — Regression

```
public class ProcessData
{
    public float Feature1 { get; set; }  // e.g., Temperature
    public float Feature2 { get; set; }  // e.g., Pressure
    public float Feature3 { get; set; }  // e.g., Flow
    public float Label { get; set; }     // e.g., EnergyConsumption (what we predict)
}

public class RegressionPrediction
{
    public float Score { get; set; }
}

private static MLContext mlContext = new MLContext(seed: 0);
private static ITransformer model;
private static IDataView lastTrainingDataView;
private static PredictionEngine<ProcessData, RegressionPrediction> predictionEngine;
private static bool modelTrained = false;
private static List<ProcessData> trainingBuffer = new List<ProcessData>();
private const int MinTrainingSize = 200;
private static readonly string ModelPath = Path.Combine(@Info.GetExecutionPath(), "<ClassName>.mlnet");

public void Predict(double input1, double input2, double input3, double label)
{
    trainingBuffer.Add(new ProcessData
    {
        Feature1 = (float)input1,
        Feature2 = (float)input2,
        Feature3 = (float)input3,
        Label = (float)label
    });

    if (!modelTrained && trainingBuffer.Count >= MinTrainingSize)
        TrainModel();

    if (modelTrained)
        RunPrediction(input1, input2, input3);
}

public void LoadModel()
{
    if (File.Exists(ModelPath))
    {
        model = mlContext.Model.Load(ModelPath, out _);
        predictionEngine = mlContext.Model.CreatePredictionEngine<ProcessData, RegressionPrediction>(model);
        modelTrained = true;
    }
}
```

```
private void TrainModel()
{
    lastTrainingDataView = mlContext.Data.LoadFromEnumerable(trainingBuffer);
    var pipeline = mlContext.Transforms.Concatenate("Features",
            nameof(ProcessData.Feature1),
            nameof(ProcessData.Feature2),
            nameof(ProcessData.Feature3))
        .Append(mlContext.Regression.Trainers.FastTree(
            labelColumnName: "Label",
            featureColumnName: "Features",
            numberOfLeaves: 20,
            numberOfTrees: 100,
            minimumExampleCountPerLeaf: 10,
            learningRate: 0.2));

    model = pipeline.Fit(lastTrainingDataView);
    predictionEngine = mlContext.Model.CreatePredictionEngine<ProcessData, RegressionPrediction>(model);
    modelTrained = true;
    SaveModel();
}

private void SaveModel()
{
    mlContext.Model.Save(model, lastTrainingDataView.Schema, ModelPath);
}

private void RunPrediction(double input1, double input2, double input3)
{
    var input = new ProcessData
    {
        Feature1 = (float)input1,
        Feature2 = (float)input2,
        Feature3 = (float)input3
    };
    var result = predictionEngine.Predict(input);

    @Tag.<AssetPath>/ML/PredictedValue.Value = (double)result.Score;
    @Tag.<AssetPath>/ML/LastPrediction.Value = DateTime.Now;
}
```

Note on training data: The `Predict()` method accepts a `label` parameter during training. This is the known actual value the model learns to predict. After training, the label is not needed for prediction-only calls. The AI should adapt the method signature based on whether the user has a label tag or trains from historical data.

Step 3: Write the Class via MCP

```
get_table_schema('ScriptsClasses')
```

```
{
  "table_type": "ScriptsClasses",
  "data": [
    {
      "Name": "<ClassName>",
      "Code": "CSharp",
      "Domain": "Server",
      "ClassContent": "Methods",
      "NamespaceDeclarations": "Microsoft.ML;Microsoft.ML.Data;Microsoft.ML.Transforms;Microsoft.ML.
Transforms.TimeSeries;Microsoft.ML.Transforms.Text;Microsoft.ML.Trainers;Microsoft.ML.TimeSeries",
      "Contents": "<AI-generated C# code from full class example above>"
    }
  ]
}
```

> Field names matter: `Code` = language (not `Language`), `Contents` = code body (not `Code`). Wrong names = silent data loss.

Step 4: Create the Trigger

Task (Periodic) — best for multi-input models

```
get_table_schema('ScriptsTasks')
```

```
{
  "table_type": "ScriptsTasks",
  "data": [
    {
      "Name": "ML_Predict_Periodic",
      "Language": "CSharp",
      "Execution": "Periodic",
      "Period": 5000,
      "Code": "@Script.Class.<ClassName>.Predict(\n    @Tag.Plant/Reactor1/Temperature.Value,\n    @Tag.Plant
/Reactor1/Pressure.Value,\n    @Tag.Plant/Reactor1/Flow.Value,\n    @Tag.Plant/Reactor1/EnergyConsumption.
Value);"
    }
  ]
}
```

> **The `@` prefix is mandatory** when referencing runtime objects inside ScriptsTasks. Without it: `CS0234`.

ServerStartup — always wire LoadModel

Read the existing `ServerStartup` task first (document object — read-modify-write), then append:

```
Script.Class.<ClassName>.LoadModel();
```

Step 5: Verify

1. **Confirm Multiplatform** — ML.NET requires .NET 8+. Instruct the user: *"Solution Settings Target Platform = Multiplatform, then Product Modify."*
2. **Do NOT start the runtime automatically.**
3. **Wait for training** — needs `MinTrainingSize` data points (default 200 for regression)
4. **Check output tags** — verify `LastPrediction` updates and `PredictedValue` is reasonable

---

Common Pitfalls

Mistake	Why It Happens	How to Avoid
Missing ML.NET namespaces	Used `AddMLNetNamespaces: true`	Always set `NamespaceDeclarations`
`CS0234` in ScriptsTasks	Missing `@` prefix	Always `@Script.Class.Name.Predict()`
`CS0246` Class not found	Build order issue	Set `BuildOrder: "1"` on ML Script Class
`System.Math` load error	.NET 4.8 target	Switch to Multiplatform
Wrong data types	ML.NET=float, tags=double	Cast `(float)`/`(double)`
Model lost on restart	SaveModel/LoadModel missing	Include both + ServerStartup
PredictionEngine null	Forgot to create after loading	Call `CreatePredictionEngine` in LoadModel

Feature count mismatch	Concatenate names don't match data class	Use `nameof()` for feature names in Concatenate

See also

- Skill ML.NET. Router skill that selects the appropriate ML.NET sub-skill based on user goals.
- Skill ML.NET Anomaly Detection. Sister sub-skill for SSA spike detection on a single sensor tag.
- Skill ML.NET Forecasting. Sister sub-skill for SSA time-series forecasting on a single sensor tag.
- Skill ML.NET Classification. Sister sub-skill for FastTree binary classification from multiple feature tags.