Skill ML.NET Classification

Predict yes/no outcomes from multiple input tags using ML.NET FastTree Binary Classification. The AI generates the full C# Script Class with multi-feature training pipeline, connects it to live tags, creates output tags for predicted label and probability, and configures model persistence.

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

- 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 Binary Classification Pipeline
  - Full Class Example — Binary Classification
- Step 3: Write the Class via MCP
- Step 4: Create the Trigger
  - Task (Periodic) — best for multi-input classification
  - ServerStartup — always wire LoadModel
- Step 5: Verify
- Common Pitfalls
- See also

When to Use This Skill

- User chose **Binary Classification — FastTree** in the ML.NET router skill (Step 0)
- Predicting a **yes/no outcome** from **2–5 input features**
- User goals: fault prediction, quality control pass/fail, predictive maintenance with multiple sensors

---

Prerequisites

- Solution open with input feature tags and a boolean 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 boolean label, (3) what the yes/no outcome 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/PredictedLabel", "DataType": "Boolean", "Description": "Predicted outcome (true
/false)" },
    { "Name": "<AssetPath>/ML/Probability", "DataType": "Double", "Description": "Prediction probability (0-
1)" },
    { "Name": "<AssetPath>/ML/LastPrediction", "DataType": "DateTime", "Description": "Timestamp of last
prediction" }
  ]
}
```

Replace `<AssetPath>` with the actual asset folder path.

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 Binary Classification Pipeline

```
var pipeline = mlContext.Transforms.Concatenate("Features", "Feature1", "Feature2", "Feature3")
    .Append(mlContext.BinaryClassification.Trainers.FastTree(
        labelColumnName: "Label",
        featureColumnName: "Features"));
```

Output: `bool PredictedLabel`, `float Score`, `float Probability`

Full Class Example — Binary Classification

```
public class ProcessData
{
    public float Feature1 { get; set; }  // e.g., Vibration
    public float Feature2 { get; set; }  // e.g., Temperature
    public float Feature3 { get; set; }  // e.g., Current
    public bool Label { get; set; }      // e.g., DidFault (true/false)
}

public class ClassificationPrediction
{
    public bool PredictedLabel { get; set; }
    public float Score { get; set; }
    public float Probability { get; set; }
}

private static MLContext mlContext = new MLContext(seed: 0);
private static ITransformer model;
private static IDataView lastTrainingDataView;
private static PredictionEngine<ProcessData, ClassificationPrediction> 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 int Predict(double input1, double input2, double input3, bool label)
{
    trainingBuffer.Add(new ProcessData
    {
        Feature1 = (float)input1,
        Feature2 = (float)input2,
        Feature3 = (float)input3,
        Label = label
    });

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

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

        return 1;
}

public void LoadModel()
{
    if (File.Exists(ModelPath))
```

```
    {
        model = mlContext.Model.Load(ModelPath, out _);
        predictionEngine = mlContext.Model.CreatePredictionEngine<ProcessData, ClassificationPrediction>
(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.BinaryClassification.Trainers.FastTree(
            labelColumnName: "Label",
            featureColumnName: "Features"));

    model = pipeline.Fit(lastTrainingDataView);
    predictionEngine = mlContext.Model.CreatePredictionEngine<ProcessData, ClassificationPrediction>(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/PredictedLabel.Value = result.PredictedLabel;
    @Tag.<AssetPath>/ML/Probability.Value = (double)result.Probability;
    @Tag.<AssetPath>/ML/LastPrediction.Value = DateTime.Now;
}
```

Note on training data: The `label` parameter is needed during training — it's the known fault/pass flag. After training, only the feature inputs are needed. The AI should adapt the signature based on whether a label tag exists.

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 classification

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

```
{
  "table_type": "ScriptsTasks",
  "data": [
    {
      "Name": "ML_Classify_Periodic",
      "Language": "CSharp",
      "Execution": "Periodic",
      "Period": 5000,
          "Code": "@Script.Class.<ClassName>.Predict(\n    @Tag.<FullPath_Feature1>,\n    @Tag.
<FullPath_Feature2>,\n    @Tag.<FullPath_Feature3>);"
     }
  ]
}
```

`<FullPath_FeatureN>` and `<FullPath_Label>` must be substituted with the exact `Name` field values from `UnsTags` — resolved via `get_objects('UnsTags', detail='summary')` in Step 0. Never infer paths from the user's shorthand. Example: if the user said "Well pressure" but the tag is `Field_A/Well_01.Pressure`, the correct reference is `@Tag.Field_A/Well_01.Pressure`.

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)
4. **Check output tags** — verify `LastPrediction` updates, `PredictedLabel` and `Probability` have reasonable values

---

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()`
`CS0029` bool-to-int on Digital tag	Digital tags are `int`, not `bool`	Use ternary: `@Tag.Path.Value = (condition) ? 1 : 0;`
`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
Imbalanced training data	Very few fault=true examples	Ensure training buffer has enough positive examples before training

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 Regression. Sister sub-skill for FastTree regression from multiple feature tags.