ML.NET Model Builder Example

Work with Visual Studio and script classes to create machine learning models.

How-to Guides **Solution Examples** **Feature Examples** **Business Operations Examples** ML.NET Model Builder Example

---

Solution (.dbsln) and model (.mlnet): MLNET-ModelBuilder.zip

Visual Studio Files: myMLApp.zip

Notice: Please ensure that the solution and the model are located in the same folder when running the application. The model's output will be displayed in the TraceWindow.

Summary

Model Builder is a Visual Studio extension that lets you build, train, and generate code to consume ML models through a guided UI.

In this example, we created a binary sentiment model to predict whether a customer review is Positive or Negative (same example as the microsoft page), then integrated the generated consumption code into FrameworX Multiplatform.

Official website of the tool: ML.NET Model Builder - machine learning in Visual Studio | .NET

This example demonstrates the Visual Studio Model Builder workflow — you build and train models in VS, then import the generated code and `.mlnet` file into FrameworX. This is ideal when you need full control over model training with custom datasets.

For **real-time anomaly detection that trains automatically from live sensor data** (no Visual Studio required), see the AnomalyML Library class — import it via Scripts Classes New Import from Library AnomalyML.

Technical Information

Requirements

- Microsoft Visual Studio 2026
- .NET 8.0
- .NET 10.0
- FrameworX Multiplatform

How the model was created (high level)

We followed Microsoft's ML.NET Tutorial - Get started in 10 minutes | .NET flow and trained a model for sentiment classification. The key benefit: Model Builder generates the training/consumption code and the model file, so integration is mostly about copying the generated consumer code and loading the `.mlnet` model.

Integration with the Solution

1 - Have the necessary dependencies

- Go to Scripts / Classes and click in the new button
- Select "Add ML.NET namespaces"
- Click OK

Copy the code below into Scripts / Tasks. This code is used to consume models inside the Scripts module. Once added, you can feed the model using your process variables in real time and read the model outputs for use in your logic.

All the code below was generated by the Model Builder tool from Visual Studio, no changes were required. Just copy and paste it into your solution.

2 - Define the Inputs and Outputs

The input schema must match what the model was trained with (column order/types). Keep the same structure you got from the generated project.

```
public class ModelInput {
    [LoadColumn(0)]
    [ColumnName(@"col0")]
    public string Col0 { get; set; }

    [LoadColumn(1)]
    [ColumnName(@"col1")]
    public float Col1 { get; set; }
}

public class ModelOutput {
    [ColumnName(@"PredictedLabel")]
    public float PredictedLabel { get; set; }
}
```

Notes

- Define **all input columns** the model expects.
- For outputs, you only need to define what you will actually use (e.g., `PredictedLabel`).

3 - Implement the consumption methods

Declare global variable and initialize the constructor

```
    string modelPath = System.IO.Path.Combine(TK.GetSolutionPath(), "SentimentModel.mlnet");
    ITransformer mlModel = mlContext.Model.Load(modelPath, out var _);

 private static PredictionEngine<ModelInput, ModelOutput> CreatePredictEngine() {
    var mlContext = new MLContext();
    ITransformer mlModel = mlContext.Model.Load(@"C:\Users\eduar\source\repos\myMLApp\myMLApp\SentimentModel.
mlnet", out var _);
    return mlContext.Model.CreatePredictionEngine<ModelInput, ModelOutput>(mlModel);
}
```

Create the prediction method:

```
public static ModelOutput Predict(ModelInput input) {
    var predEngine = PredictEngine.Value;
    return predEngine.Predict(input);
}
```



`TK.GetSolutionPath()` returns the directory where the `.dbsln` file is located. Place the `.mlnet` model file in the same folder as your solution so it is found automatically, regardless of where the solution is installed.

4 - Create the Main method

```
public async Task Main() {

    var sampleData = new ModelInput() {
        Col0 = "This restaurant was bad."
    };

    var result = SentimentModel.Predict(sampleData);
    var sentiment = result.PredictedLabel == 1 ? "Positive" : "Negative";

    @Info.Trace($"Text: {sampleData.Col0}");
    @Info.Trace($"Sentiment: {sentiment}");
}
```

5 - Output

The output in the TraceWindow will look like this:

Text: Great quality!
Sentiment: Positive

> Make sure the Scripts module is enabled in the Trace Window settings so you can view the output.
>
> You can use any LLM to assist during implementation. For this implementation, we used Visual Studio with Claude Code.

In this section...
