Introduction to ML.NET

Jeff Prosise

jeffpro@wintellect.com

@jprosise

Wintellect®

Wintellect NOW

ML.NET

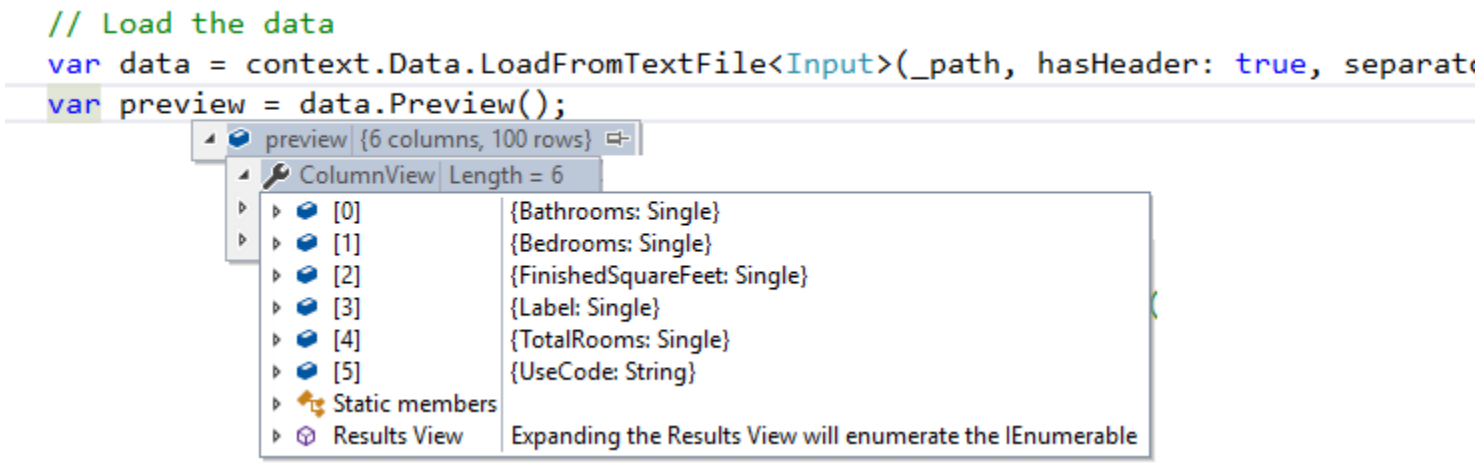
- Derived from **TLC**, the internal library used by products such as Windows Hello, Bing Ads, and Azure Machine Learning Studio
  - Free, open-source, and cross-platform
  - Runs on .NET and .NET Core (Windows, macOS, and Linux)
  - Supports regression, classification (binary and multiclass), anomaly detection, recommendations, and clustering (k-means)
  - https://github.com/dotnet/machinelearning
- Modular and extensible, with support for Open Neural Network Exchange (ONNX) and loading pretrained neural networks
- Built for speed, large datasets, and parallel processing

Demo

Simple Regression

Data Views

- Represent data in ML.NET apps like DataFrames in Pandas
  - Implement **IDataView** interface (forward read-only cursor)
  - Created and manipulated with **DataOperationsCatalog** methods
  - Contain data and schema information and are lazily evaluated
- Use **Preview** method in debugger to look inside



Loading Data from CSV and TSV Files

```
// Load data from a CSV file that contains a header row
var data = context.Data.LoadFromTextFile<Input>("PATH_TO_DATA_FILE", hasHeader: true,
    separatorChar: ',');

// Load data from a TSV file without a header row. Allow quotes and trim whitespace.
var data = context.Data.LoadFromTextFile<Input>("PATH_TO_DATA_FILE", allowQuoting: true,
    trimWhitespace: true);

// Load data from multiple CSV files (all files must have the same schema)
var loader = context.Data.CreateTextLoader<Input>(hasHeader: true, separatorChar: ',');
var data = loader.Load("PATH1", "PATH2", "PATH3");
```

Loading Data from Databases and Other Sources

```
// TODO: Load data into an array or other IEnumerable from an external data source.
// The following example simply creates an array in memory.

var input = new[]
{
    new Input { Age = 30, YearsExperience = 10, ... },
    new Input { Age = 40, YearsExperience = 20, ... },
    new Input { Age = 50, YearsExperience = 30, ... }
};

var data = context.Data.LoadFromEnumerable<Input>(input);
```

Preparing Data

- **DataOperationsCatalog** also has methods for preparing data
  - Filter rows by the values they contain
  - Remove rows with missing values
  - Normalize values in specified columns
  - Bin (quantize) values in specified columns
- **TransformsCatalog** and **TextCatalog** contain additional methods
  - Identify and replace missing values
  - Remove columns, copy columns, and select columns
  - Vectorize ("featurize") text, load and resize images, and more

Filtering and Preparing Data

```
// Remove rows with missing values in the "Age" and "YearsExperience" columns
var view = context.Data.FilterRowsByMissingValues(data, "Age", "YearsExperience");

// Remove rows where "Age" is less than 20 or greater than 80
var view = context.Data.FilterRowsByColumn(data, "Age", lowerBound: 20, upperBound: 80);

// Remove the "Age" column
var estimator = context.Transforms.DropColumns("Age");
var view = estimator.Fit(data).Transform(data);

// Replace missing values in the "Age" column
var estimator = context.Transforms.ReplaceMissingValues("Age",
    replacementMode: MissingValueReplacingEstimator.ReplacementMode.Mean);
var view = estimator.Fit(data).Transform(data);
```

Splitting Data

- **DataOperationsCatalog.TrainTestSplit** splits a dataset into a training set and a testing set
  - **testFraction** specifies split (e.g., 80/20, 50/50)
  - **seed** specifies random seed for splitting

```
var trainTestData = context.Data.TrainTestSplit(data, testFraction: 0.2, seed: 0);
var trainData = trainTestData.TrainSet;
var testData = trainTestData.TestSet;
```

Regression Trainers

Class	Description
LbfgsPoissonRegressionTrainer	Predicts counts using Poisson regression
LightGbmRegressionTrainer	Performs regression using Gradient Boosting Machines (GBM)
SdcaRegressionTrainer	Performs regression using Stochastic Dual Coordinate Ascent (SDCA)
OlsTrainer	Performs Ordinary Least Squares (OLS) linear regression
OnlineGradientDescentTrainer	Performs regression using Online Gradient Descent (OGD)
FastTreeRegressionTrainer	Performs regression using decision trees
FastTreeTweedieTrainer	Performs regression for Tweedie distributions using decision trees
FastForestRegressionTrainer	Performs regression using random forests
GamRegressionTrainer	Performs regression using Generalized Additive Models (GAM)

Regression with Random Forests

```
// Split the data into a training set and a test set
var trainTestData = context.Data.TrainTestSplit(data, testFraction: 0.2, seed: 0);
var trainData = trainTestData.TrainSet;
var testData = trainTestData.TestSet;

// Build the pipeline
var pipeline = context.Transforms.Concatenate("Features", "Col1", "Col2", "Col3")
    .Append(context.Regression.Trainers.FastForest());

// Train the model
var model = pipeline.Fit(trainData);
```

```
featureColumnName:
labelColumnName:
numberOfLeaves:
numberOfTrees:
```

Specifying Training Options

```
var options = new FastForestRegressionTrainer.Options
{
    // Only use 80% of features to reduce over-fitting
    FeatureFraction = 0.8,
    // Simplify the model by penalizing usage of new features
    FeatureFirstUsePenalty = 0.1,
    // Limit the number of trees to 50
    NumberOfTrees = 50
};

var pipeline = context.Transforms.Concatenate("Features", "Col1", "Col2", "Col3")
    .Append(context.Regression.Trainers.FastForest(options));
```

Scoring Regression Models

- **RegressionCatalog.Evaluate** returns a **RegressionMetrics** object with properties for R-squared, MAE, MSE, and other regression metrics
- Call **TransformerChain.Transform** on trained model to generate predictions to score against

```
// Evaluate the model
var predictions = model.Transform(testData);
var metrics = context.Regression.Evaluate(predictions);
Console.WriteLine($"R2 score: {metrics.RSquared:0.##}");
```

≤77ms elapsed

metrics {Microsoft.ML.Data.RegressionMetrics}

LossFunction	1803768391616.08
MeanAbsoluteError	735413.94875
MeanSquaredError	1803768384573.228
RSquared	0.7371489894268981
RootMeanSquaredError	1343044.4462389278

Cross-Validating Regression Models

- **RegressionCatalog.CrossValidate** divides a dataset into "folds" and score each fold separately
  - R-squared, Mean Absolute Error (MAE), Mean Squared Error (MSE), etc.
- Use mean of scores from all folds to get a more robust measure of accuracy

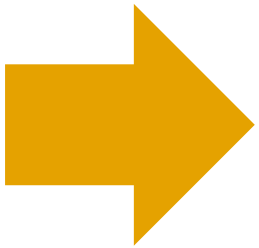
```
// Evaluate the model using cross-validation
var scores = context.Regression.CrossValidate(data, pipeline, numberOfFolds: 5);
var mean = scores.Average(x => x.Metrics.RSquared);
Console.WriteLine($"Mean cross-validated R2 score: {mean:0.##}");
```

One-Hot Encoding

- **CategoricalCatalog.OneHotEncoding** method transforms categorical values into numbers using one-hot encoding

```
var encoder = context.Transforms.Categorical.OneHotEncoding(
    inputColumnName: "company", outputColumnName: "company_encoded"
);
var encodedData = encoder.Fit(data).Transform(data);
```

company	...	label
microsoft	...	0
google	...	1
microsoft	...	0
facebook	...	0
facebook	...	1



microsoft	google	facebook	...	label
1	0	0	...	0
0	1	0	...	1
1	0	0	...	0
0	0	1	...	0
0	0	1	...	1

Demo

Multiple Regression

Binary Classification Trainers

Class	Description
AveragedPerceptronTrainer	Performs classification using perceptrons
SdcaLogisticRegressionBinaryTrainer	Performs classification using calibrated logistic regression
SdcaNonCalibratedBinaryTrainer	Performs classification using non-calibrated logistic regression
SymbolicSgdLogisticRegressionBinaryTrainer	Performs classification using SGD logistic regression
LbfgsLogisticRegressionBinaryTrainer	Performs classification using LBFGS logistic regression
LightGbmBinaryTrainer	Performs classification using Gradient Boosting Machines
FastTreeBinaryTrainer	Performs classification using decision trees
FastForestBinaryTrainer	Performs classification using random forests
GamBinaryTrainer	Performs classification using GAM regression
FieldAwareFactorizationMachineTrainer	Performs classification using FFMs
LinearSvmTrainer	Performs classification using SVM with a linear kernel

Scoring Binary Classification Models

- **BinaryClassificationCatalog.Evaluate** returns a **CalibratedBinaryClassificationMetrics** object with properties for accuracy, F1, AUC, and other metrics, including a confusion matrix

```
// Evaluate the model
var predictions = model.Transform(testData);
var metrics = context.BinaryClassification.Evaluate(predictions, "Label");

Console.WriteLine();
Console.WriteLine($"Accuracy: {metrics.Accuracy:P2}");
Console.WriteLine($"AUC: {metrics.AreaUnderPrecisionRecallCurve:P2}");
Console.WriteLine($"F1: {metrics.F1Score:P2}");

// Use the model to make predict
var predictor = context.Model.Cr

foreach (var sample in _samples)
{
    var input = new Input { Sent
    var prediction = predictor.P
```

metrics {Microsoft.ML.Data.CalibratedBinaryClassificationMetrics}

Accuracy	0.75233644859813087
AreaUnderPrecisionRecallCurve	0.82525891641857518
AreaUnderRocCurve	0.85
ConfusionMatrix	{Microsoft.ML.Data.ConfusionMatrix}
Entropy	0.99691053503547855
F1Score	0.73891625615763534
LogLoss	0.73740000939546568
LogLossReduction	0.2603147589675911
NegativePrecision	0.77477477477477474
NegativeRecall	0.75438596491228072
PositivePrecision	0.72815533980582525
PositiveRecall	0.75

Cross-Validating Binary Classification Models

- Use **BinaryClassificationCatalog.CrossValidate** to divide the dataset into "folds" and score each fold separately

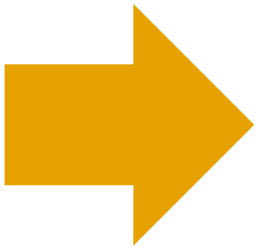
```
// Evaluate the model using cross-validation
var scores = context.BinaryClassification.CrossValidate(data,
    pipeline, numberOfFolds: 5);
var mean = scores.Average(x => x.Metrics.F1Score);
Console.WriteLine($"Mean cross-validated F1 score: {mean:P2}");
```

Vectorizing Text

- **TextCatalog.FeaturizeText** vectorizes text for use in machine learning
- Removes stop words, numbers, and punctuation characters, converts text to lowercase, and includes n-gram support

```
var featurizer = context.Transforms.Text.FeaturizeText("Features", "Text");
var transformedText = featurizer.Fit(data).Transform(data);
```

```
"The quick brown fox",
"Jumps over the brown dog."
```



quick	brown	fox	jumps	over	dog
1	1	1	0	0	0
0	1	0	1	1	1

Specifying Text-Featurization Options

```
var stop = new CustomStopWordsRemovingEstimator.Options {
    StopWords = new[] { "and", "or", "the", "i", "you", "we", "am", "are", "were", "a" }
};

var options = new TextFeaturizingEstimator.Options
{
    StopWordsRemoverOptions = stop, // Set to null to leave stop words intact
    KeepNumbers = true, // Default == false
    WordFeatureExtractor = new WordBagEstimator.Options { NgramLength = 2 },
    CaseMode = TextNormalizingEstimator.CaseMode.None // Default = Lower
};

var featurizer = context.Transforms.Text.FeaturizeText("Features", options, "Text");
var transformedText = featurizer.Fit(data).Transform(data);
```

Demo

Binary Classification

Multiclass Classification Trainers

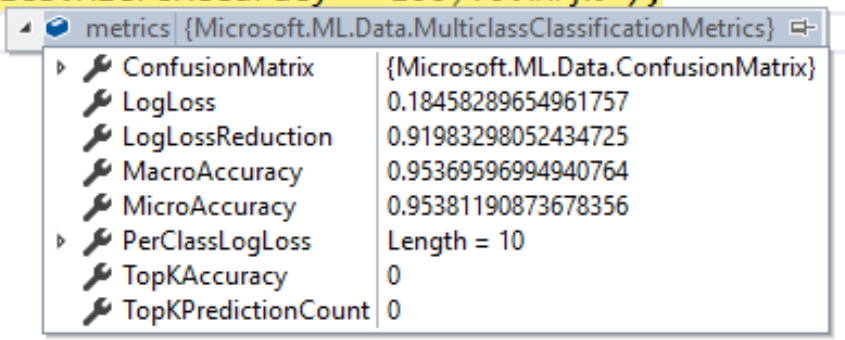
Class	Description
LightGbmMulticlassTrainer	Performs classification using Gradient Boosting Machines
SdcaMaximumEntropyMulticlassTrainer	Performs classification using calibrated Stochastic Dual Coordinate Ascent (SDCA)
SdcaNonCalibratedMulticlassTrainer	Performs classification using non-calibrated Stochastic Dual Coordinate Ascent (SDCA)
LbfgsMaximumEntropyMulticlassTrainer	Performs classification using Limited-memory BFGS
NaiveBayesMulticlassTrainer	Performs classification using naïve Bayes algorithm
OneVersusAllTrainer	Uses binary classifier to train one classifier per class
PairwiseCouplingTrainer	Performs classification using pairwise coupling algorithm

Scoring Multiclass Classification Models

- **MulticlassClassificationCatalog.Evaluate** returns a **MulticlassClassificationMetrics** object with properties for macro accuracy, micro accuracy, and other metrics, including a confusion matrix

```
// Evaluate the model
var predictions = model.Transform(testData);
var metrics = context.MulticlassClassification.Evaluate(predictions);

Console.WriteLine();
Console.WriteLine($"Macro accuracy = {(metrics.MacroAccuracy * 100):0.##}%");
Console.WriteLine($"Micro accuracy = {(metrics.MicroAccuracy * 100):0.##}%");
Console.WriteLine();
```



Cross-Validating Multiclass Classification Models

- Use **MulticlassClassificationCatalog.CrossValidate** to divide the dataset into "folds" and score each fold separately

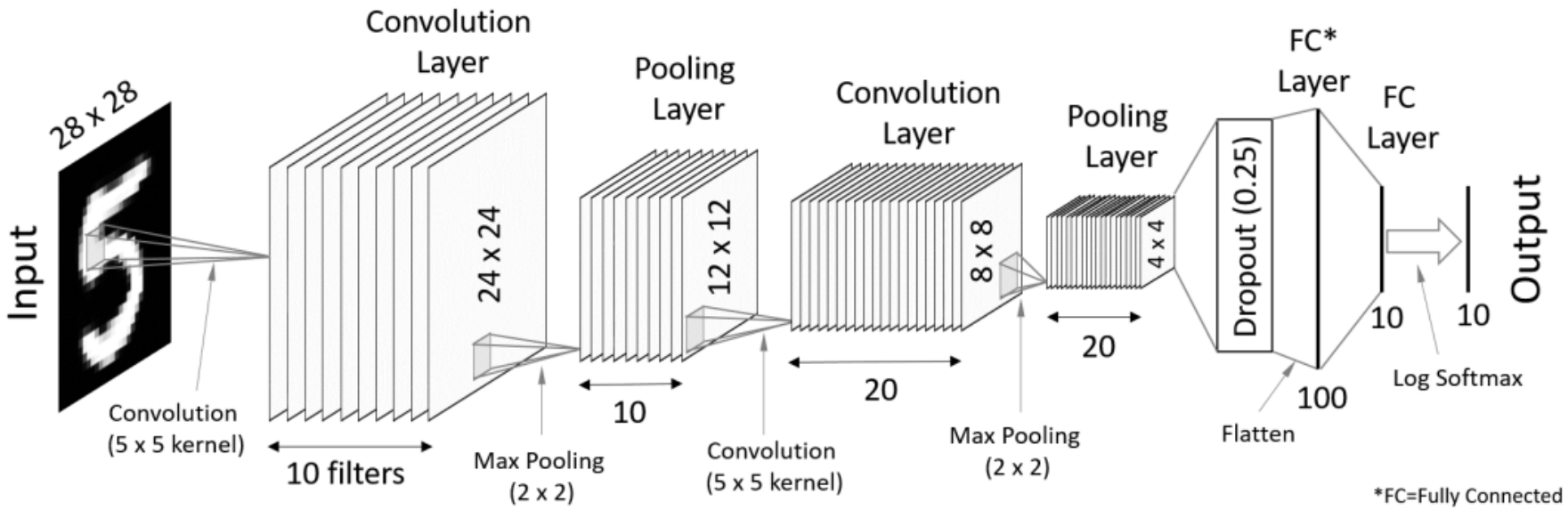
```
// Evaluate the model using cross-validation
var scores = context.MulticlassClassification.CrossValidate(data,
    pipeline, numberOfFolds: 5);
var mean = scores.Average(x => x.Metrics.MacroAccuracy);
Console.WriteLine($"Mean cross-validated macro accuracy: {mean:P2}");
```

Demo

Multiclass Classification

Classifying Images

- Deep convolutional neural networks excel at image classification
- ML.NET's **ModelOperationsCatalog** and **TensorFlowModel** classes have methods for loading and working with TensorFlow models
- **TransformsCatalog** class provides APIs for working with images



Transfer Learning

- Leverages pretrained deep CNNs to achieve acceptable accuracy with exponentially less data, compute power, and training time
  - Replaces fully connected layers in pretrained model with newly trained layers, reusing pretrained model's convolutional base for feature recognition
  - Allows image-classification models to be trained with as few as 50-100 images
  - Eliminates need for GPU-equipped HPC clusters (train on a laptop)
- Pretrained DNNs available from Microsoft, Google, and others
  - Frequently made available through GitHub
- Some libraries (e.g., Keras) include popular pretrained DNNs

Saving and Loading a Trained Model

```
// Save a trained model to a local zip file
context.Model.Save(model, data.Schema, "PATH_TO_ZIP_FILE");

// Load a trained model from a local zip file
DataViewSchema schema;
model = context.Model.Load("PATH_TO_ZIP_FILE", out schema);

// Load a trained model from a stream
DataViewSchema schema;
model = context.Model.Load(stream, out schema);
```

Demo

Image Classification

AutoML

- API for finding the best model without manual experimentation
  - Regression
  - Binary classification
  - Multiclass classification
- Automatically performs a variety of tasks
  - Splits data into training set and test set
  - Cleans data, performs automatic feature selection, one-hot encodes categorical values, featurizes text, and more
  - Tries many learning algorithms with hyperparameter tuning and cross-validation

Automating a Regression Model

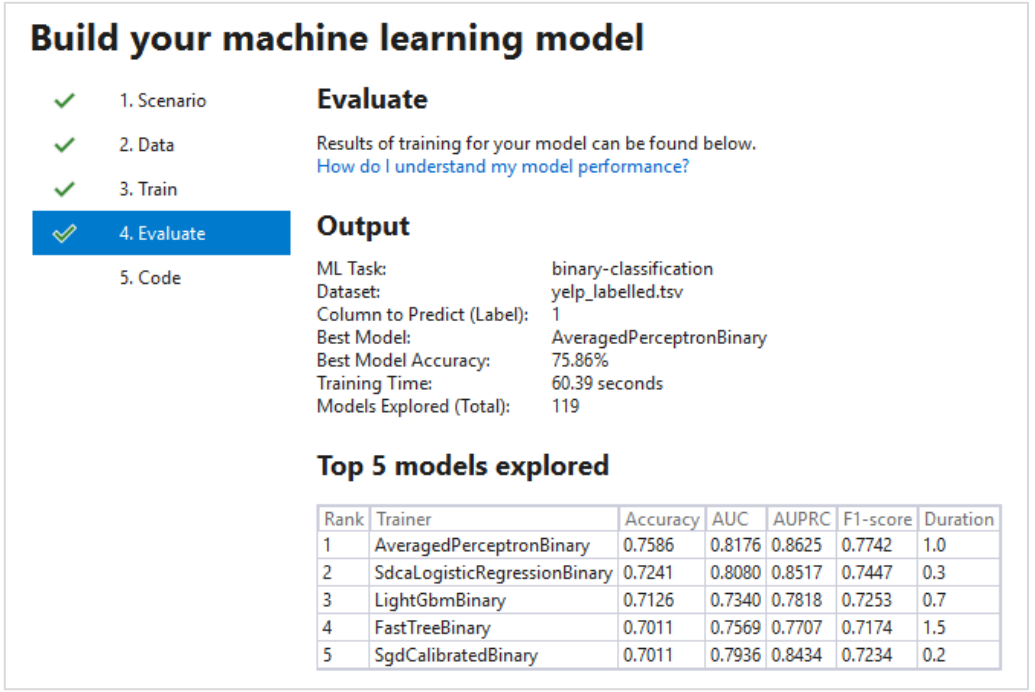
```
var settings = new RegressionExperimentSettings
{
    MaxExperimentTimeInSeconds = 600, // 10 minutes max
    OptimizingMetric = RegressionMetric.RSquared,
    CacheDirectory = null // Cache in memory
};

var experiment = context.Auto().CreateRegressionExperiment(settings);
var result = experiment.Execute(data);

var BestR2 = result.BestRun.ValidationMetrics.RSquared;
var BestModel = result.BestRun.Model;
```

Visual Studio Model Builder

- AutoML plugin for building machine-learning models in VS
  - http://aka.ms/mlnettemplates
  - Supports local files and SQL Server as data sources
  - Saves trained model and generates code to use it
- Use ML.NET CLI to automate model building on macOS and Linux
  - https://bit.ly/30cJroT



Get the Code

https://github.com/jeffprosise/ML.NET

ML.NET Cookbook

https://github.com/dotnet/machinelearning/blob/master/docs/code/MlNetCookBook.md