ML.NET at .NET Conf 2018 Vancouver

## Medhat Elmasry Oct 3, 2018

File >> New >> Project >> Visual C# >> .NET Core node >> Console App (.NET Core)

Name application *TaxiFarePrediction*

Set the Platform target to x64 (*Properties* >> *Build*).

Set the language version to C# 7.1 or higher (*Properties* >> *Build* >> *Language version*).

Create directory named *Data* in the project

Install *Microsoft.ML* package.

Download datasets from:

* https://github.com/dotnet/machinelearning/blob/master/test/data/taxi-fare-train.csv
* https://github.com/dotnet/machinelearning/blob/master/test/data/taxi-fare-test.csv

For both data files, change *Properties >> Copy to Output Directory* to the value of *Copy if newer*.

The *label* is the identifier of the column you want to predict. The identified *features* are used to predict the label.

Open *taxi-fare-train.csv* and look at the column headers in the first row. Understand which columns are *features* and which one is the *label*.

|  |  |
| --- | --- |
| vendor\_id | The ID of the taxi vendor is a feature. |
| rate\_code | The rate type of the taxi trip is a feature. |
| passenger\_count | The number of passengers on the trip is a feature. |
| trip\_time\_in\_secs | The amount of time the trip took. You want to predict the fare of the trip before the trip is completed. At that moment you don't know how long the trip would take. Thus, the trip time is not a feature and you'll exclude this column from the model. |
| trip\_distance | The distance of the trip is a feature. |
| payment\_type | The payment method (cash or credit card) is a feature. |
| fare\_amount | The total taxi fare paid is the label. |

Create a folder named *Models* and add to it the classes *TaxiTrip* and *TaxiTripFarePrediction*:

### Define data and model paths

public class TaxiTripFarePrediction {

[ColumnName("Score")]

public float FareAmount;

}

public class TaxiTrip {

[Column("0")]

public string VendorId;

[Column("1")]

public string RateCode;

[Column("2")]

public float PassengerCount;

[Column("3")]

public float TripTime;

[Column("4")]

public float TripDistance;

[Column("5")]

public string PaymentType;

[Column("6")]

public float FareAmount;

}

Add the following code right above the *Main()* method in *Program.cs* to specify data paths:

static readonly string \_datapath = Path.Combine(Environment.CurrentDirectory, "Data", "taxi-fare-train.csv");

static readonly string \_testdatapath = Path.Combine(Environment.CurrentDirectory, "Data", "taxi-fare-test.csv");

static readonly string \_modelpath = Path.Combine(Environment.CurrentDirectory, "Data", "Model.zip");

Add the following additional *using* statements to *Program.cs*:

using Microsoft.ML;

using Microsoft.ML.Data;

using Microsoft.ML.Models;

using Microsoft.ML.Trainers;

using Microsoft.ML.Transforms;

Replace *Console.WriteLine("Hello World!")* with the following code:

PredictionModel<TaxiTrip, TaxiTripFarePrediction> model = Train();

Add a blank *Train()* method:

public static PredictionModel<TaxiTrip, TaxiTripFarePrediction> Train() {

}

Instantiate a learning pipeline:

var pipeline = new LearningPipeline();

Load data into the pipeline:

pipeline.Add(new TextLoader(\_datapath).CreateFrom<TaxiTrip>(useHeader: true, separator: ','));

As we want to predict the taxi trip fare, copy the *FareAmount* column into the *Label* column. To do that, use *ColumnCopier* and add the following code:

pipeline.Add(new ColumnCopier(("FareAmount", "Label")));

Transform categorical data (*VendorId*, *RateCode*, and *PaymentType*) values into numbers:

pipeline.Add(new CategoricalOneHotVectorizer("VendorId", "RateCode", "PaymentType"));

Combine all of the feature columns into the *Features* column using the *ColumnConcatenator* transformation class.

pipeline.Add(new ColumnConcatenator("Features",

"VendorId",

"RateCode",

"PassengerCount",

"TripDistance",

"PaymentType"));

### Choose a learning algorithm

The learner trains the model. You chose a regression task for this problem, so you use a *FastTreeRegressor* learner, which is one of the regression learners provided by ML.NET.

Add the following code into the *Train* method following the data processing code added in the previous step:

pipeline.Add(new FastTreeRegressor());

### Train the model

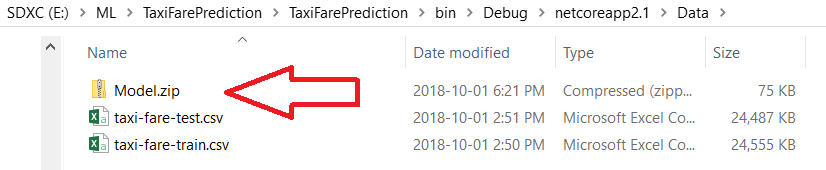
*pipeline.Train<TInput, TOutput>* method produces the model. It takes in an instance of the *TInput* type and outputs an instance of the *TOutput* type.

PredictionModel<TaxiTrip, TaxiTripFarePrediction> model = pipeline.Train<TaxiTrip, TaxiTripFarePrediction>();

To save the model to a .zip file, add the following code at the end of the Train() method:

await model.WriteAsync(\_modelpath);

return model;

The model will be saved in the *bin/Debug/netcoreapp2.1/Data directory*: 

Modify the signature of *Train()* method as shown in the following code:

public static async Task<PredictionModel<TaxiTrip, TaxiTripFarePrediction>> Train()

Modify the signature of *Main()* method as shown in the following code:

static async Task Main(string[] args)

In the *Main()* method, put the keyword *await* before *Train()*.

### Evaluate model

Evaluation is the process of checking how well the model predicts *label* values. Let’s see how well the model performs on test data.

In the *Main()* method, add the following code beneath the call to the *Train()* method:

Evaluate(model);

Add the following empty *Evaluate()* method to *Program.cs*:

private static void Evaluate(PredictionModel<TaxiTrip, TaxiTripFarePrediction> model) {

}

Add the following code into the *Evaluate()* method to load the test data:

var testData = new TextLoader(\_testdatapath).CreateFrom<TaxiTrip>(useHeader: true, separator: ',');

Add the following code to evaluate the model and produce the evaluation metrics:

var evaluator = new RegressionEvaluator();

RegressionMetrics metrics = evaluator.Evaluate(model, testData);

RMS is one of the evaluation metrics of the regression model. The lower it is, the better the model is. Add the following code into the *Evaluate()* method to display the RMS value:

Console.WriteLine($"Rms = {metrics.Rms}");

*RSquared* takes values between 0 and 1. The closer its value is to 1, the better the model is. Add the following code into the *Evaluate()* method to display the *RSquared* value:

Console.WriteLine($"RSquared = {metrics.RSquared}");

### Use model for predictions

Add the following method that pertains to a possible trip:

private static TaxiTrip getTrip() {

return new TaxiTrip() {

VendorId = "VTS",

RateCode = "1",

PassengerCount = 1,

TripDistance = 10.33f,

PaymentType = "CSH",

FareAmount = 0 // predict it. actual = 29.5

};

}

To predict the fare of a specified trip, add the following code into the *Main()* method:

TaxiTripFarePrediction prediction = model.Predict(getTrip());

Console.WriteLine("Predicted fare: {0}, actual fare: 29.5", prediction.FareAmount);

[](https://www.bing.com/images/search?view=detailV2&ccid=g6nYHgjm&id=9DCD800E35FF6162FE4C44ABC6C70A5783B46357&thid=OIP.g6nYHgjml5I7LyjheiVEGwElEs&q=run+logo&simid=608050058236200411&selectedIndex=67)Run the application.

Not adding a normalizer.

Making per-feature arrays

Changing data from row-wise to column-wise

Processed 1048575 instances

Binning and forming Feature objects

Reserved memory for tree learner: 45864 bytes

Starting to train ...

Not training a calibrator because it is not needed.

Rms = 3.30299146626885

RSquared = 0.885729301000846

Predicted fare: 31.14972, actual fare: 29.5

### Consume Model

Let us use the model that was saved in *bin/Debug/netcoreapp2.1/Data directory/Model.zip*.

Add a method *Predict()* method to *Program.cs*:

public static async Task<TaxiTripFarePrediction> Predict(TaxiTrip trip) {

PredictionModel<TaxiTrip, TaxiTripFarePrediction> \_model = await PredictionModel.ReadAsync<TaxiTrip, TaxiTripFarePrediction>(\_modelpath);

var prediction = \_model.Predict(trip);

return prediction;

}

Comment out all the code in the *Main()* method and add this code instead:

TaxiTripFarePrediction prediction = await Predict(getTrip());

Console.WriteLine("Predicted fare: {0}, actual fare: 29.5", prediction.FareAmount);

[](https://www.bing.com/images/search?view=detailV2&ccid=g6nYHgjm&id=9DCD800E35FF6162FE4C44ABC6C70A5783B46357&thid=OIP.g6nYHgjml5I7LyjheiVEGwElEs&q=run+logo&simid=608050058236200411&selectedIndex=67)Run the application. You should get the same predicted result.

Predicted fare: 31.14972, actual fare: 29.5

### Reference:

[Tutorial: Use ML.NET to predict New York taxi fares (regression)](https://docs.microsoft.com/en-ca/dotnet/machine-learning/tutorials/taxi-fare)