# ML.Net Samples Notes

Jan ‘19

D:\Projects\back\MLNet

Machine Learning

<https://blogs.msdn.microsoft.com/dotnet/2019/01/10/announcing-ml-net-0-9-machine-learning-for-net/>

Written 10Jan’19

ML.NET is an open-source and cross-platform machine learning framework (Windows, Linux, macOS) for .NET developers. Using ML.NET, developers can leverage their existing tools and skillsets to develop and infuse custom AI into their applications by creating custom machine learning models.

ML.NET allows you to create and use machine learning models targeting common tasks such as classification, regression, clustering, ranking, recommendations and anomaly detection. It also supports the broader open source ecosystem by proving integration with popular deep-learning frameworks like TensorFlow and interoperability through ONNX. Some common use cases of ML.NET are scenarios like Sentiment Analysis, Recommendations, Image Classification, Sales Forecast, etc. Please see our samples for more scenarios.

# Feature Contribution Calculation (FCC)

The Feature Contribution Calculation (FCC for short) shows which features are most influential for a model’s prediction on a particular and individual data sample by determining the amount each feature contributed to the model’s score for that particular data sample.

FCC is particulary important when you initialy have a lot of features/attributes in your historic data and you want to select and use only the most important features because using too many features (especially if including features that don’t influence the model) can reduce the model’s performance and accuracy. Therefore, with FCC you can identify the most influential positive and negative contributions from the initial attribute set.

You can use FCC to produce feature contributions with code like the following: blah blah blah

# Getting the code

<https://github.com/dotnet/machinelearning> to D:\Projects\back\MLNet\machinelearning

# Getting the samples

<https://dotnet.microsoft.com/apps/machinelearning-ai/ml-dotnet>

<https://github.com/dotnet/machinelearning-samples> to D:\Projects\back\MLNet\MlSamples

# Tutorial Guide

<https://dotnet.microsoft.com/learn/machinelearning-ai/ml-dotnet-get-started-tutorial>

<https://docs.microsoft.com/en-us/dotnet/machine-learning/tutorials/>

The following tutorials enable you to understand how to use [ML.NET](https://docs.microsoft.com/en-us/dotnet/machine-learning/index) to build custom machine learning solutions and integrate them into your .NET applications:

* [Sentiment analysis](https://docs.microsoft.com/en-us/dotnet/machine-learning/tutorials/sentiment-analysis): demonstrates how to apply a **binary classification** task using ML.NET.
* [Taxi fare predictor](https://docs.microsoft.com/en-us/dotnet/machine-learning/tutorials/taxi-fare): demonstrates how to apply a **regression** task using ML.NET.
* [Iris clustering](https://docs.microsoft.com/en-us/dotnet/machine-learning/tutorials/iris-clustering): demonstrates how to apply a **clustering** task using ML.NET.

For more examples that use ML.NET, check the [dotnet/machinelearning-samples](https://github.com/dotnet/machinelearning-samples) GitHub repository.

## Sentiment Analysis Binary Classification

<https://docs.microsoft.com/en-us/dotnet/machine-learning/tutorials/sentiment-analysis>

D:\Projects\back\MLNet\MlSamples\machinelearning-samples\samples\csharp\getting-started\BinaryClassification\_SentimentAnalysis\SentimentAnalysis-Solution.sln

I’m confused as to how the projects are organized, there seem to be samples and then a separate tutorial series……….

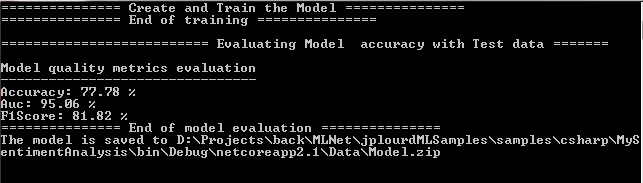
<https://github.com/dotnet/samples> contains a **machine-learning/tutorials** directory

I’ve cloned it here D:\Projects\back\MLNet\dotnetSamples

The code for this sentiment project is here: D:\Projects\back\MLNet\dotnetSamples\samples\machine-learning\**tutorials**\SentimentAnalysis

This version works and the one I’m building by following along with the tutorial does as well (D:\Projects\back\MLNet\jplourdMLSamples\samples\csharp\**MySentimentAnalysis**\Program.cs)

The output:



***Classification*** is a machine learning task that uses data to **determine** the category, type, or class of an item or row of data. For example, you can use classification to:

* Identify sentiment as positive or negative.
* Classify email as spam, junk, or good.
* Determine whether a patient's lab sample is cancerous.
* Categorize customers by their propensity to respond to a sales campaign.

Classification tasks are frequently one of the following types:

* Binary: either A or B.
* Multiclass: multiple categories that can be predicted by using a single model.

SentimentData is the input dataset class and has a float (Sentiment) that has a value for sentiment of either positive or negative, and a string for the comment   
(SentimentText). Both fields have Column attributes attached to them. This attribute describes the order of each field in the data file, and which is the Label field.   
SentimentPrediction is the class used for prediction after the model has been trained. It has a single boolean (Sentiment) and a PredictedLabel ColumnName attribute. The   
Label is used to create and train the model, and it's also used with a second dataset to evaluate the model. The PredictedLabel is used during prediction and evaluation. For   
evaluation, an input with training data, the predicted values, and the model are used.

ML.NET's transform pipelines compose a custom set of transforms that are applied to your data before training or testing. The transforms' primary purpose is   
data [featurization](https://docs.microsoft.com/en-us/dotnet/machine-learning/resources/glossary" \l "feature-engineering). Machine learning algorithms understand [featurized](https://docs.microsoft.com/en-us/dotnet/machine-learning/resources/glossary" \l "feature) data, so the next step is to transform our textual data into a format that our ML   
algorithms recognize. That format is a [numeric vector](https://docs.microsoft.com/en-us/dotnet/machine-learning/resources/glossary#numerical-feature-vector).

Next, call mlContext.Transforms.Text.FeaturizeText which featurizes the text column (SentimentText) column into a numeric vector   
called Features used by the machine learning algorithm. This is a wrapper call that returns an [EstimatorChain<TLastTransformer>](https://docs.microsoft.com/en-us/dotnet/api/microsoft.ml.runtime.data.estimatorchain-1) that will   
effectively be a pipeline. Name this pipeline as you will then append the trainer to the EstimatorChain.

To add the trainer, call the mlContext.Transforms.Text.FeaturizeText wrapper method which returns a [FastTreeBinaryClassificationTrainer](https://docs.microsoft.com/en-us/dotnet/api/microsoft.ml.trainers.fasttree.fasttreebinaryclassificationtrainer) object. This is   
a decision tree learner you'll use in this pipeline. The FastTreeBinaryClassificationTrainer is appended to the pipeline and accepts the   
featurized SentimentText (Features) and the Label input parameters to learn from the historic data.

You train the model, [TransformerChain<TLastTransformer>](https://docs.microsoft.com/en-us/dotnet/api/microsoft.ml.data.transformerchain-1), based on the dataset that has been loaded and transformed. Once the estimator has been   
defined, you train your model using the [Fit](https://docs.microsoft.com/en-us/dotnet/api/microsoft.ml.runtime.data.estimatorchain-1.fit)while providing the already loaded training data. This returns a model to use for   
predictions. pipeline.Fit() trains the pipeline and returns a Transformer based on the DataView passed in. The experiment is not executed until this   
happens.

## TaxiFare regression predictor

D:\Projects\back\MLNet\jplourdMLSamples\samples\csharp\**MyTaxiFarePrediction**

D:\Projects\back\MLNet\dotnetSamples\samples\machine-learning\tutorials\TaxiFarePrediction

Note: ‘Column’ names by convention carry extraordinary meaning!

//When the model is trained and evaluated, by default, the values in the **Label** column are considered as correct values to be predicted.

//As we want to predict the taxi trip fare, copy the FareAmount column into the Label column. To do that,

//use the CopyColumnsEstimator transformation class

var pipeline = mlContext.Transforms.CopyColumns("FareAmount", "**Label**")

//The algorithm that trains the model requires numeric features, so you have **to transform the categorical data**

**//(VendorId, RateCode, and PaymentType) values into numbers**. To do that, use the OneHotEncodingEstimator transformation

// class, which assigns different numeric key values to the different values in each of the columns,

.Append(mlContext.Transforms.Categorical.OneHotEncoding("VendorId"))

.Append(mlContext.Transforms.Categorical.OneHotEncoding("RateCode"))

.Append(mlContext.Transforms.Categorical.OneHotEncoding("PaymentType"))

// The last step in data preparation combines all of the feature columns into the Features column using the

//ColumnConcatenatingEstimator transformation class. By default, **a learning algorithm processes**

**// only features from the Features column**

.Append(mlContext.Transforms.Concatenate("**Features**", "VendorId", "RateCode", "PassengerCount", "TripTime", "TripDistance", "PaymentType"))

# ML Encog

<https://www.heatonresearch.com/encog/>

Pluralsight course:

<https://app.pluralsight.com/player?course=introduction-to-machine-learning-encog&author=abhishek-kumar&name=introduction-to-machine-learning-encog-m5-intro-encog3&clip=5&mode=live>

D:\Projects\back\MLNet\Encog

Built the core solution to get the runtime dlls