

## Aplicação com Machine Learning com .NET Core 6

### ROTEIRO DE AULA PRÁTICA

#### INSTALAÇÃO DA FERRAMENTA MLNET

```
$ dotnet tool install -g mlnet-linux-x64
```

#### PARA TESTAR

```
$ mlnet
```

#### PRÁTICA 1: APLICAR ALGORITMO DE M.L. PELO C.L.I. (Análise de Sentimento)

##### CRIAR DIRETÓRIO DA APLICAÇÃO

```
$ mkdir MLApp
```

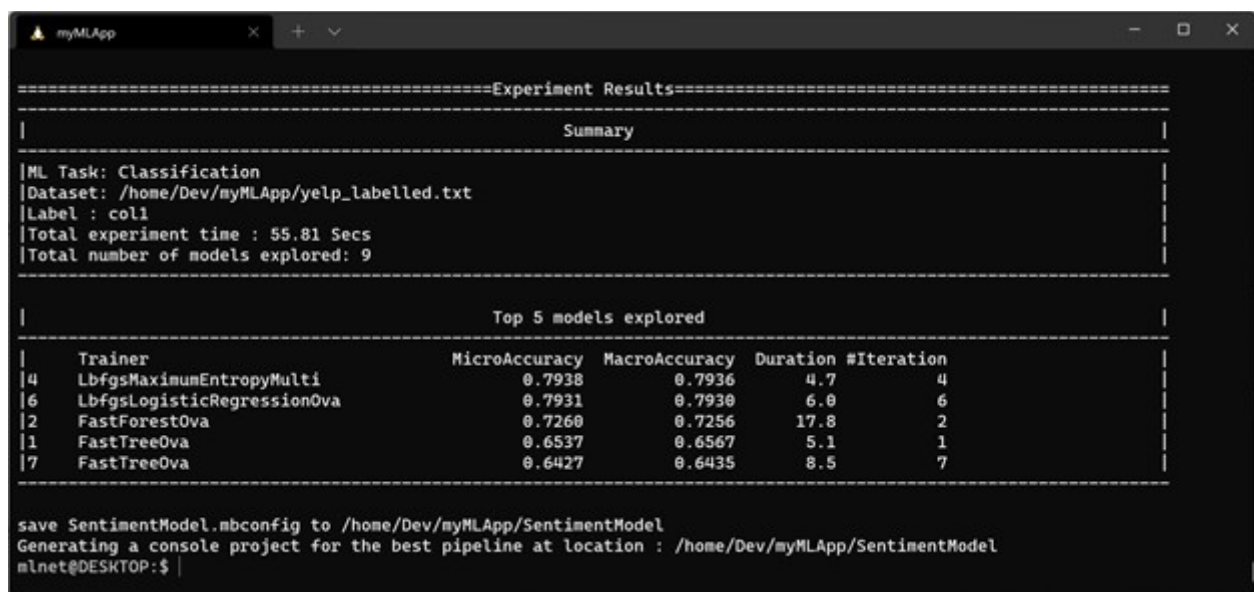
```
$ cd MLApp
```

##### COLOCAR O ARQUIVO DE DADOS NO DIRETÓRIO DA APLICAÇÃO yelp\_labelled.txt

##### NA PASTA DA APLICAÇÃO, ENTRE COM O SEGUINTE COMANDO

```
mlnet classification --dataset "yelp_labelled.txt" --label-col 1  
--has-header false --name SentimentModel --train-time 60
```

O resultado exibido será os 5 melhores modelos e suas métricas



```
=====Experiment Results=====
|
| Summary
|
| ML Task: Classification
| Dataset: /home/Dev/myMLApp/yelp_labelled.txt
| Label: coll
| Total experiment time : 55.81 Secs
| Total number of models explored: 9
|
| Top 5 models explored
|
| Trainer                MicroAccuracy  MacroAccuracy  Duration  #Iteration
|4  LbfgsMaximumEntropyMulti  0.7938         0.7936         4.7       4
|6  LbfgsLogisticRegressionOva 0.7931         0.7930         6.0       6
|2  FastForestOva             0.7260         0.7256         17.8      2
|1  FastTreeOva                0.6537         0.6567         5.1       1
|7  FastTreeOva                0.6427         0.6435         8.5       7
|
save SentimentModel.nbconfig to /home/Dev/myMLApp/SentimentModel
Generating a console project for the best pipeline at location : /home/Dev/myMLApp/SentimentModel
mlnet@DESKTOP:$
```

No arquivo Program.cs, pode ser feito teste com outras frases

```
$ dotnet build
```

```
$ dotnet run
```

## PRATICA 2: TREINAR E CONSUMIR UMA APLICAÇÃO DE M.L. EM UMA APLICAÇÃO DE CONSOLE (Doenças Cardíacas)

### CRIANDO APLICAÇÃO CONSOLE COM ML

```
$ dotnet new console -o HeartDesease
$ dotnet add package Microsoft.ML
$ dotnet add package Microsoft.ML.FastTree
```

Criar pasta Data na raiz do projeto e copiar o arquivo de dados para essa pasta

No arquivo program.cs apagar seu conteúdo e adicionar as seguintes diretivas

```
using Microsoft.ML;
using Microsoft.ML.Data;
using static Microsoft.ML.DataOperationsCatalog;
using System;
using System.IO;
using HeartDisease.Data;
```

Definir o caminho para a base de dados e para o modelo (será criado posteriormente)

```
string TrainingDatasetPath = "Data/HeartTraining.csv";
string TestDatasetPath = "Data/HeartTest.csv";
string ModelPath = "MLModels/HeartClassification.zip";
```

Na pasta Data, criar um arquivo para definir nossa classe de entrada (HeartData.cs) e um para definir nossa classe de previsão (HeartPrediction.cs)

```
using Microsoft.ML.Data;

namespace HeartDisease.Data
{
    public class HeartData
    {
        [LoadColumn(0)]
        public float Age { get; set; }
        [LoadColumn(1)]
        public float Sex { get; set; }
        [LoadColumn(2)]
        public float Cp { get; set; }
        [LoadColumn(3)]
        public float TrestBps { get; set; }
        [LoadColumn(4)]
        public float Chol { get; set; }
        [LoadColumn(5)]
        public float Fbs { get; set; }
        [LoadColumn(6)]
        public float RestEcg { get; set; }
        [LoadColumn(7)]
        public float Thalac { get; set; }
        [LoadColumn(8)]
```

```

        public float Exang { get; set; }
        [LoadColumn(9)]
        public float OldPeak { get; set; }
        [LoadColumn(10)]
        public float Slope { get; set; }
        [LoadColumn(11)]
        public float Ca { get; set; }
        [LoadColumn(12)]
        public float Thal { get; set; }
        [LoadColumn(13), ColumnName("Label")]
        public int Label { get; set; }
    }
}

```

E a classe HeartPrediction

```

using Microsoft.ML.Data;

namespace HeartDisease.Data
{
    public class HeartPrediction : HeartData
    {
        [ColumnName("PredictionLabel")]
        public int Prediction {get; set;}
        public float Probability {get; set;}
        public float Score {get; set;}
    }
}

```

No arquivo program.cs, instanciar um Machine Learning Context  
 MLContext mlContext = new MLContext();

Carregar os dados e retornar um objeto TrainTestData.

```

var trainingDataView =
mlContext.Data.LoadFromTextFile<HeartData>(TrainingDatasetPath,
hasHeader: true, separatorChar: ';');

```

Chamar um método para criar e treinar o modelo com base no conjunto de dados de treinamento  
 ITransformer model = BuildAndTrainModel(mlContext,
splitDataView.TrainSet);

Implementar o método de treinamento do Modelo

```
ITransformer BuildAndTrainModel(MLContext mlContext, IDataView
TrainSet)
{
    // Mapear as Saídas como valores chaves (label) e concatenar
os atributos em um array chamado Features
    // Adicionar Trainer do tipo FastTree
    var pipeline =
mlContext.Transforms.Concatenate("Features","Age", "Sex", "Cp",
"TrestBps", "Chol", "Fbs", "RestEcg", "Thalac", "Exang",
"OldPeak", "Slope", "Ca", "Thal")
        .Append(mlContext.BinaryClassification.Trainers.FastTree(l
abelColumnName: "Label", featureColumnName: "Features"));

    // Treinamento do Modelo
    Console.WriteLine("===== Criar e Treinar o Modelo
=====");
    var model = pipeline.Fit(TrainSet);
    Console.WriteLine("===== Final do Treinamento
=====");
    Console.WriteLine();

    return model;
}
```

Chamar um método Evaluate(..) para avaliar o modelo treinado sob o conjunto de dados de teste

```
Evaluate(mlContext, model, splitDataView.TestSet);
```

Implementar o método Evaluate(mlContext, model, TestDataSet)

```
void Evaluate(MLContext mlContext, ITransformer model)
{
    // Carregar conjunto de dados de Teste
    var testDataView =
mlContext.Data.LoadFromTextFile<HeartData>(TestDatasetPath,
hasHeader: true, separatorChar: ';');

    Console.WriteLine("===== Avaliando o Modelo com o conjunto
de dados de teste =====");
    // Transformar os dados de testes, como feito com os dados de
treinamento e obter as predições
    IDataView predictions = model.Transform(testDataView);

    // Calcular as métricas do modelo com base nas predições
feitas
```

```

        CalibratedBinaryClassificationMetrics metrics =
mlContext.BinaryClassification.Evaluate(predictions, "Label",
"Score");

        // Exibir Métricas do Modelo
        Console.WriteLine();
        Console.WriteLine("Avaliação das métricas de qualidade do
modelo");

Console.WriteLine("-----");
;
        Console.WriteLine($"Acurácia: {metrics.Accuracy:P2}");
        Console.WriteLine($"AAC-Roc: {metrics.AreaUnderRocCurve:P2}");
        Console.WriteLine($"F1-Score: {metrics.F1Score:P2}");
        Console.WriteLine("===== Final da avaliação do
modelo =====");
    }

```

Chamar o método para salvar o modelo

```
SaveTrainedModel(mlContext, model);
```

Implementar o método para salvar o modelo treinado em arquivo

```

void SaveTrainedModel (MLContext mlContext, ITransformer model)
{
    Console.WriteLine("===== Salvando modelo em arquivo
=====");
    mlContext.Model.Save(model, trainingDataView.Schema,
ModelPath);
    Console.WriteLine("");
    Console.WriteLine("");
    Console.WriteLine("===== Model Saved =====
");
}

```

Chamar o método para fazer previsões com o modelo treinado

```
TestPrediction(mlContext);
```

Implementar o Método TestPrediction(mlContext)

```

void TestPrediction(MLContext mlContext)
{
    // Carregar o modelo treinado
    ITransformer trainedModel = mlContext.Model.Load(ModelPath,
out var modelInputSchema);

    // Criar um prediction engine
    var predictionEngine =
mlContext.Model.CreatePredictionEngine<HeartData,
HeartPrediction>(trainedModel);

```

```

foreach(var heartData in DataSample.heartDataList)
{
    var prediction = predictionEngine.Predict(heartData);

    Console.WriteLine($"===== Single Prediction
=====");
    Console.WriteLine($"Age: {heartData.Age} ");
    Console.WriteLine($"Sex: {heartData.Sex} ");
    Console.WriteLine($"Cp: {heartData.Cp} ");
    Console.WriteLine($"TrestBps: {heartData.TrestBps} ");
    Console.WriteLine($"Chol: {heartData.Chol} ");
    Console.WriteLine($"Fbs: {heartData.Fbs} ");
    Console.WriteLine($"RestEcg: {heartData.RestEcg} ");
    Console.WriteLine($"Thalac: {heartData.Thalac} ");
    Console.WriteLine($"Exang: {heartData.Exang} ");
    Console.WriteLine($"OldPeak: {heartData.OldPeak} ");
    Console.WriteLine($"Slope: {heartData.Slope} ");
    Console.WriteLine($"Ca: {heartData.Ca} ");
    Console.WriteLine($"Thal: {heartData.Thal} ");
    Console.WriteLine($"Prediction Value:
{(Convert.ToBoolean(prediction.Prediction))} ");
    Console.WriteLine($"Prediction: {(prediction.Prediction ?
"A disease could be present" : "Not present disease" )} ");
    Console.WriteLine($"Probability: {prediction.Probability}
");

    Console.WriteLine($"=====
=====");
    Console.WriteLine("");
    Console.WriteLine("");
}
}

```