Taller ML.NET

Plataformas para el desarrollo de software

0. Puesta a punto

Para comenzar un proyecto en ML.NET necesitaremos instalar .NET Core SDK:

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

Para continuar necesitaremos también disponer de Visual Studio 2017 (versiones <= vs2015 no valen).

Existe una alternativa a utilizar Visual Studio 2017, y es utilizar Visual Studio Code, que es un editor de texto con características avanzadas y extensibles gracias a plugins.

Nosotros realizaremos el taller en Windows 10 con el editor de texto Visual Studio Code.

1. Creación del proyecto

Una vez instalado .NET SDK y Visual Studio Code, abrimos la línea de comandos de Windows y escribimos los siguientes mandatos:

```
mkdir tallerMLNET
cd tallerMLNET
dotnet new console -o PrediccionInfarto
cd PrediccionInfarto
```

Hemos creado un directorio llamado tallerMLNET y dentro hemos creado una proyecto de tipo 'consola' llamado `PrediccionInfarto`.

Como ya habrás podido deducir, con este ejercicio vamos a tratar de predecir si una persona va a sufrir un infarto dependiendo de diversos factores.

Acto seguido vamos a instalar en nuestro proyecto el paquete de ML.NET, para ello, en la misma línea de comandos ejecutamos:

```
dotnet add package Microsoft.ML
```

2. Visualización del *dataset*

Descargamos el *dataset* de entrenamiento y verificación desde el repositorio de Github:

https://github.com/osmartinez/TallerMLNET/blob/master/PrediccionInfarto/heart_train.csv https://github.com/osmartinez/TallerMLNET/blob/master/PrediccionInfarto/heart_test.csv

Ubicamos los ficheros descargados dentro de la carpeta de nuestro proyecto.

Si echamos un vistazo al fichero veremos un conjunto de líneas donde cada línea tiene un conjunto de valores delimitados por ';'. Esto es un fichero CSV (*comma-separated values*) y son muy utilizados en *Machine Learning*. Microsoft Excel permite abrir este tipo de ficheros y visualizarlos en formato tabular (las ';' marcan las divisiones por columnas).

Si nos fijamos, la primera fila del fichero (cabecera o *header*) define los descriptores del *dataset*. El resto de filas son los datos que debemos analizar.

3. Extracción de descriptores

Abrimos el fichero Program.cs con Visual Studio Code y añadimos las siguientes líneas al principio del fichero:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using Microsoft.Data.DataView;
using Microsoft.ML;
using Microsoft.ML.Core.Data;
using Microsoft.ML.Data;
using Microsoft.ML.Transforms.Text;
```

Vamos a extraer las categorías que queramos entrenar. Visualizamos y valoramos el dataset. Observamos que todos los descriptores aportan información valiosa para la predicción, así que vamos a ver el tipo de los datos de entrada.

Verificamos que todos los descriptores son valores numéricos. Excepto el resultado de la predicción (*target*) que es de tipo booleano (1 o 0).

Antes que nada, vamos a instanciar el contexto:

```
MLContext mlContext = new MLContext();
```

Ahora vamos a crear un objeto que va a actuar de lector. Este lector va a necesitar conocer la estructura de los datos de entrada y la ruta del fichero donde están nuestros datos almacenados.

```
var lector = mlContext.Data.CreateTextLoader(
new[] {new TextLoader.Column("Descriptores", DataKind.R4, new[] {new
TextLoader.Range(0, 12)}),
new TextLoader.Column("Resultado", DataKind.Bool, 13)},
separatorChar: ';',
hasHeader: true);
```

Con la anterior definición, estamos diciendo que:

- vamos a leer datos separados por el delimitador ';',
- el fichero tiene cabecera, así que se salte la primera fila,
- los descriptores están ubicados de la columna 0 a la 12 y son de tipo Real (R4),
- que el resultado de la predicción está en la columna 13 y este dato es de tipo booleano.

4. Lectura/carga del dataset

Una vez que tenemos nuestros descriptores definidos, vamos a proceder a leer el fichero y cargar los datos en memoria.

Antes que nada, definimos dos constantes que almacenarán la ruta a nuestros ficheros de entrenamiento y test.

```
const string rutaEntrenamiento = "heart_train.csv";
const string rutaTest = "heart_test.csv";
```

Y seguidamente vamos a leer el dataset de entrenamiento utilizando el lector anteriormente definido.

```
var datosEntrenamiento = lector.Read(rutaEntrenamiento);
```

5. Elección y entrenamiento de un clasificador

Llegados a este paso, lo que normalmente se suele hacer es escoger un conjunto de clasificadores y entrenarlos para ver cuáles arrojan los mejores resultados. También se utilizan técnicas de optimización de hiperparámetros (argumentos de los clasificadores) que pueden afinar aún más las predicciones.

En nuestro caso, simplemente vamos a escoger algunos clasificadores disponibles y vamos a comparar los resultados para quedarnos con el mejor de ellos.

Para elegir un clasificador apto, es necesario estudiar la naturaleza del problema. En nuestro caso, estamos resolviendo un problema de clasificación binaria.

La mayoría de clasificadores soportan la clasificación binaria. Nosotros tomaremos tres clasificadores que utilizan técnicas distintas para entrenar:

- Clasificador lineal: *StochasticDualCoordinateAscent*.
- Clasificador basado en árboles de decisión: *FastTree*.
- Clasificador basado en regresión logística: *Logistic Regression*.

Vamos a definir los tres clasificadores.

```
var clfLineal =
mlContext.BinaryClassification.Trainers.StochasticDualCoordinateAsce
nt (labelColumn: "Resultado", featureColumn: "Descriptores");

var clfLR =
mlContext.BinaryClassification.Trainers.LogisticRegression
(labelColumn: "Resultado", featureColumn: "Descriptores");

var clfTrees =
mlContext.BinaryClassification.Trainers.FastTree
(labelColumn:"Resultado",featureColumn:"Descriptores",numLeaves: 50,
numTrees: 30, minDatapointsInLeaves: 20);
```

Y comenzamos el proceso de entrenamiento para nuestros tres clasificadores.

```
var modelLineal = clfLineal.Fit(datosEntrenamiento);

var modelLR = clfLR.Fit(datosEntrenamiento);

var modelTrees = clfTrees.Fit(datosEntrenamiento);
```

Con esto ya hemos convertido los clasificadores en modelos entrenados contra el *dataset* de entrenamiento.

6. Evaluación del modelo

Una vez tenemos nuestros modelos entrenados, vamos a evaluar cómo de bien se comportan. Es la hora de utilizar el fichero que contiene los datos para la verificación. Estos datos formaban parte del *dataset* inicial, pero los apartamos a un fichero distinto para que el clasificador no los conozca y podamos valorar el ajuste o comportamiento ante nuevos datos de entrada.

Leemos los datos del fichero de verificación.

```
var datosTest = lector.Read(rutaTest);
```

Para cada uno de nuestros modelos elaboramos una predicción.

```
var predLineal = modelLineal.Transform(datosTest);

var predLR = modelLR.Transform(datosTest);


var predTrees = modelTrees.Transform(datosTest);
```

Ahora que ya tenemos las predicciones, vamos a utilizar una métrica para valorar la precisión de cada predicción.

```
var metricaLineal =
mlContext.BinaryClassification.Evaluate(predLineal, "Resultado");

var metricaLR = mlContext.BinaryClassification.Evaluate(predLR,
"Resultado");

var metricaTrees =
mlContext.BinaryClassification.Evaluate(predTrees, "Resultado");
```

Finalmente mostramos por pantalla los resultados de las métricas.

```
Console.WriteLine($"Precisión modelo lineal:
{metricaLineal.Accuracy:P2}");

Console.WriteLine($"Precisión regresión logística:
{metricaLR.Accuracy:P2}");

Console.WriteLine($"Precisión árboles de decisión:
{metricaTrees.Accuracy:P2}");
```