CLUSTERING

Asmaliza Ahzan
IVERSON ASSOCIATES SDN BHD

Table of Contents

Categorize iris flowers using k-means clustering with ML.NET

https://docs.microsoft.com/en-us/dotnet/machine-learning/tutorials/iris-clustering

This exercise illustrates how to use ML.NET to build a clustering model for the iris flower data set.

In this exercise, you learn how to:

- Understand the problem
- Select the appropriate machine learning task
- Prepare the data
- Load and transform the data
- Choose a learning algorithm
- Train the model
- Use the model for predictions

Prerequisites

Visual Studio 2019 or later or Visual Studio 2017 version 15.6 or later with the ".NET Core cross-platform development" workload installed.

Understand the problem

This problem is about dividing the set of iris flowers in different groups based on the flower features. Those features are the length and width of a sepal and the length and width of a petal. For this exercise, assume that the type of each flower is unknown. You want to learn the structure of a data set from the features and predict how a data instance fits this structure.

Select the appropriate machine learning task

As you don't know to which group each flower belongs to, you choose the unsupervised machine learning task. To divide a data set in groups in such a way that elements in the same group are more similar to each other than to those in other groups, use a clustering machine learning task.

Create a console application

1. Open Visual Studio. Select File > New > Project from the menu bar. In the New Project dialog, select the Visual C# node followed by the .NET Core node. Then select the Console App (.NET Core) project template. In the Name text box, type "IrisFlowerClustering" and then select the OK button.
2. Create a directory named Data in your project to store the data set and model files:
    a. In Solution Explorer, right-click the project and select Add > New Folder. Type "Data" and hit Enter.
3. Install the Microsoft.ML NuGet package.
    a. In Solution Explorer, right-click the project and select Manage NuGet Packages. Choose "nuget.org" as the Package source, select the Browse tab, search for Microsoft.ML and select the Install button. Select the OK button on the Preview Changes dialog and then select the I Accept button on the License Acceptance dialog if you agree with the license terms for the packages listed.

Prepare the data

1. Download the iris.data data set and save it to the Data folder you've created at the previous step. For more information about the iris data set, see the Iris flower data set Wikipedia page and the Iris Data Set page, which is the source of the data set.
2. In Solution Explorer, right-click the iris.data file and select Properties. Change the value of Copy to Output Directory to Copy if newer.

The iris.data file contains five columns that represent:

- sepal length in centimeters
- sepal width in centimeters
- petal length in centimeters
- petal width in centimeters
- type of iris flower

Create data classes

Create classes for the input data and the predictions:

1. In Solution Explorer, right-click the project, and then select Add > New Item.
2. In the Add New Item dialog box, select Class and change the Name field to IrisData.cs. Then, select the Add button.
3. Add the following using directive to the new file:

```
using Microsoft.ML.Data;
```

Remove the existing class definition and add the following code, which defines the classes IrisData and ClusterPrediction, to the IrisData.cs file:

```
public class IrisData
{
    [LoadColumn(0)]
    public float SepalLength;

    [LoadColumn(1)]
    public float SepalWidth;

    [LoadColumn(2)]
    public float PetalLength;

    [LoadColumn(3)]
    public float PetalWidth;
}

public class ClusterPrediction
{
    [ColumnName("PredictedLabel")]
    public uint PredictedClusterId;

    [ColumnName("Score")]
    public float[] Distances;
   }
```

IrisData is the input data class and has definitions for each feature from the data set. Use the LoadColumn attribute to specify the indices of the source columns in the data set file.

The ClusterPrediction class represents the output of the clustering model applied to an IrisData instance. Use the ColumnName attribute to bind the PredictedClusterId and Distances fields to the PredictedLabel and Score columns respectively. In case of the clustering task those columns have the following meaning:

- PredictedLabel column contains the ID of the predicted cluster.
- Score column contains an array with squared Euclidean distances to the cluster centroids. The array length is equal to the number of clusters.

Define data and model paths

Go back to the Program.cs file and add two fields to hold the paths to the data set file and to the file to save the model:

- _dataPath contains the path to the file with the data set used to train the model.
- _modelPath contains the path to the file where the trained model is stored.

Add the following code right above the Main method to specify those paths:

```
static readonly string _dataPath = Path.Combine(Environment.CurrentDirectory, "Data",
"iris.data");
static readonly string _modelPath = Path.Combine(Environment.CurrentDirectory, "Data",
"IrisClusteringModel.zip");
```

To make the preceding code compile, add the following using directives at the top of the Program.cs file:

```
using System;
using System.IO;
```

Create ML context

Add the following additional using directives to the top of the Program.cs file:

```
using Microsoft.ML;
using Microsoft.ML.Data;
```

In the Main method, replace the Console.WriteLine("Hello World!"); line with the following code:

```
var mlContext = new MLContext(seed: 0);
```

The Microsoft.ML.MLContext class represents the machine learning environment and provides mechanisms for logging and entry points for data loading, model training, prediction, and other tasks. This is comparable conceptually to using DbContext in Entity Framework.

Set up data loading

Add the following code to the Main method to set up the way to load data:

```
IDataView dataView = mlContext.Data.LoadFromTextFile<IrisData>(_dataPath, hasHeader:
false, separatorChar: ',');
```

The generic MLContext.Data.LoadFromTextFile extension method infers the data set schema from the provided IrisData type and returns IDataView which can be used as input for transformers.

Create a learning pipeline

For this exercise, the learning pipeline of the clustering task comprises two following steps:

- concatenate loaded columns into one Features column, which is used by a clustering trainer;
- use a KMeansTrainer trainer to train the model using the k-means++ clustering algorithm.

Add the following code to the Main method:

```
string featuresColumnName = "Features";
var pipeline = mlContext.Transforms
                .Concatenate(featuresColumnName, "SepalLength", "SepalWidth",
"PetalLength", "PetalWidth")
                .Append(mlContext.Clustering.Trainers.KMeans(featuresColumnName,
numberOfClusters: 3));
```

The code specifies that the data set should be split in three clusters.

Train the model

The steps added in the preceding sections prepared the pipeline for training, however, none have been executed. Add the following line to the Main method to perform data loading and model training:

```
var model = pipeline.Fit(dataView);
```

Save the model

At this point, you have a model that can be integrated into any of your existing or new .NET applications. To save your model to a .zip file, add the following code to the Main method:

```
using (var fileStream = new FileStream(_modelPath, FileMode.Create, FileAccess.Write,
FileShare.Write))
{
       mlContext.Model.Save(model, dataView.Schema, fileStream);
}
```

Use the model for predictions

To make predictions, use the PredictionEngine<TSrc,TDst> class that takes instances of the input type through the transformer pipeline and produces instances of the output type. Add the following line to the Main method to create an instance of that class:

```
var predictor = mlContext.Model.CreatePredictionEngine<IrisData,
ClusterPrediction>(model);
```

The PredictionEngine is a convenience API, which allows you to perform a prediction on a single instance of data. PredictionEngine is not thread-safe. It's acceptable to use in single-threaded or prototype environments. For improved performance and thread safety in production environments, use the PredictionEnginePool service, which creates an ObjectPool of PredictionEngine objects for use throughout your application. See this guide on how to use PredictionEnginePool in an ASP.NET Core Web API.

Create the TestIrisData class to house test data instances:

1. In Solution Explorer, right-click the project, and then select Add > New Item.
2. In the Add New Item dialog box, select Class and change the Name field to TestIrisData.cs. Then, select the Add button.
3. Modify the class to be static like in the following example:

```
static class TestIrisData
 {
    }
```

This exercise introduces one iris data instance within this class. You can add other scenarios to experiment with the model. Add the following code into the TestIrisData class:

```
internal static readonly IrisData Setosa = new IrisData
     {
         SepalLength = 5.1f,
         SepalWidth = 3.5f,
```

```
            PetalLength = 1.4f,
            PetalWidth = 0.2f
           };
```

To find out the cluster to which the specified item belongs to, go back to the Program.cs file and add the following code into the Main method:

```
var prediction = predictor.Predict(TestIrisData.Setosa);
Console.WriteLine($"Cluster: {prediction.PredictedClusterId}");
Console.WriteLine($"Distances: {string.Join(" ", prediction.Distances)}");
```

Run the program to see which cluster contains the specified data instance and squared distances from that instance to the cluster centroids. Your results should be similar to the following:

text Copy

```
Cluster: 2
Distances: 11.69127 0.02159119 25.59896
```

Congratulations! You've now successfully built a machine learning model for iris clustering and used it to make predictions. You can find the source code for this exercise at the dotnet/samples GitHub repository.