

Introduction to recommender systems with ML.NET

We are on the verge of one of the biggest changes in human history. Processes are optimized faster than ever, and increasingly more mundane and even mental tasks are being automated, all driven by machine learning models. Scientists are already training models for art and music creation. We have better speech recognition like Cortana, Siri and Alexa. Nowadays, it is even possible to participate in contests purely aimed at helping people. One step closer to uniting people everywhere through machines, so welcome to the age of machine learning.

Contents

Machine learning 101	1
What are we building?.....	1
Project setup.....	1
Model trainer.....	2
Value predictor	11
Next steps	13
From the author	13

Machine learning 101

Machine learning is a subset of AI (Artificial Intelligence) that specializes in the creation of mathematical models. These models are built to learn, improve and act in a human like way. All of this is possible by combining different fields of mathematics like statistics, probability theory, multivariate calculus, linear algebra and so on.

Hardware enthusiasts and professionals already know the computational leap we took in the past few years. We went from mainstream consumer CPU’s with 2 cores to CPU’s with 8 cores in the last 2 years alone. Additionally, a lot of processing is moved to the GPU for easier parallelization and faster memory access as well as more memory bandwidth. All this new hardware augmented with new and improved software are creating new possibilities we could not imagine 10 years ago.

Autonomous vehicles or crop yield optimizations thanks to better weather forecasting as well as better timed harvesting. These are only a few of the big challenges we can solve by using machine learning. Of course, we will not be trying to solve one of those but focus on a smaller challenge. Our goal will be creating personal value that you can easily transfer to a specific set of challenges found in most enterprise environments.

What are we building?

A lot of people all over the world struggle to combine their work, social interaction and leisure. I am one of those persons who likes to fill his time with events and courses, but occasionally I need to wind down by enjoying a good book. Unfortunately, it is hard to research beforehand which book is worth my time and I do not like reading one third of a book just to be disappointed and chuck it out altogether.

Fortunately, many of my friends have the same taste as me, what brought me to the idea to create a system that analyzes my rated books and matches them to my friends’ book ratings. The higher the rating, the more likely my interest. To solve this first world problem and save some of my time, we will be creating a machine learning model. This project will focus on analyzing the relationship between books and users and try to predict the rating for a book that the user has not read yet.

Project setup

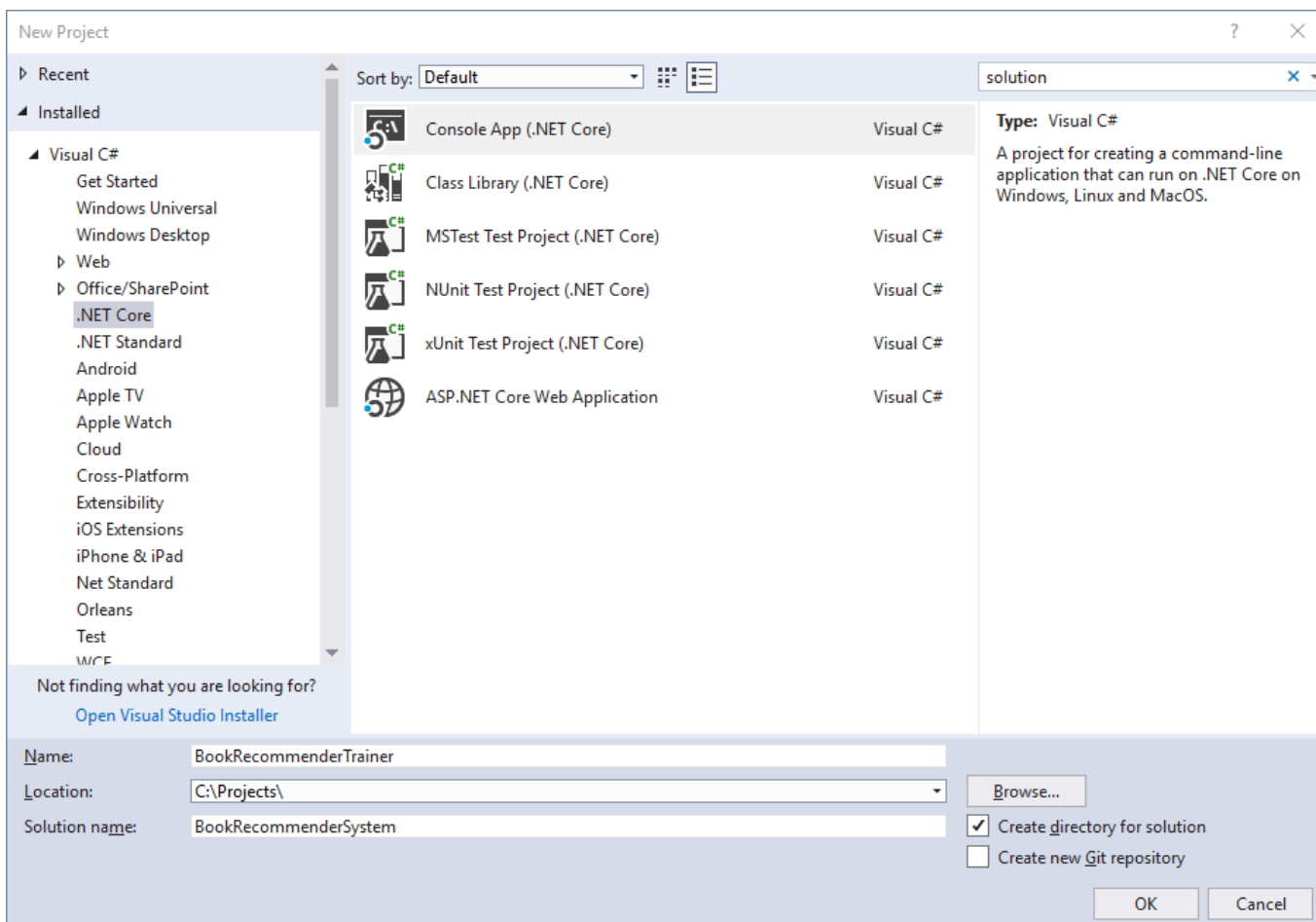
Creating machine learning models is mostly popular in languages like R and Python, but recently Microsoft started investing in a new open source framework called ML.NET. This framework brings machine learning pipelines straight to .Net. It already provides some simple but very powerful integrated trainers and optimizers to solve more than a handful of challenges. Even when things get out of hand and the complexity cannot be handled by ML.NET you can still import external models from TensorFlow or even the ONNX opensource file format. We could even export our models to run on other devices or in other projects. That is exactly what we are going to create together.

Prerequisites:

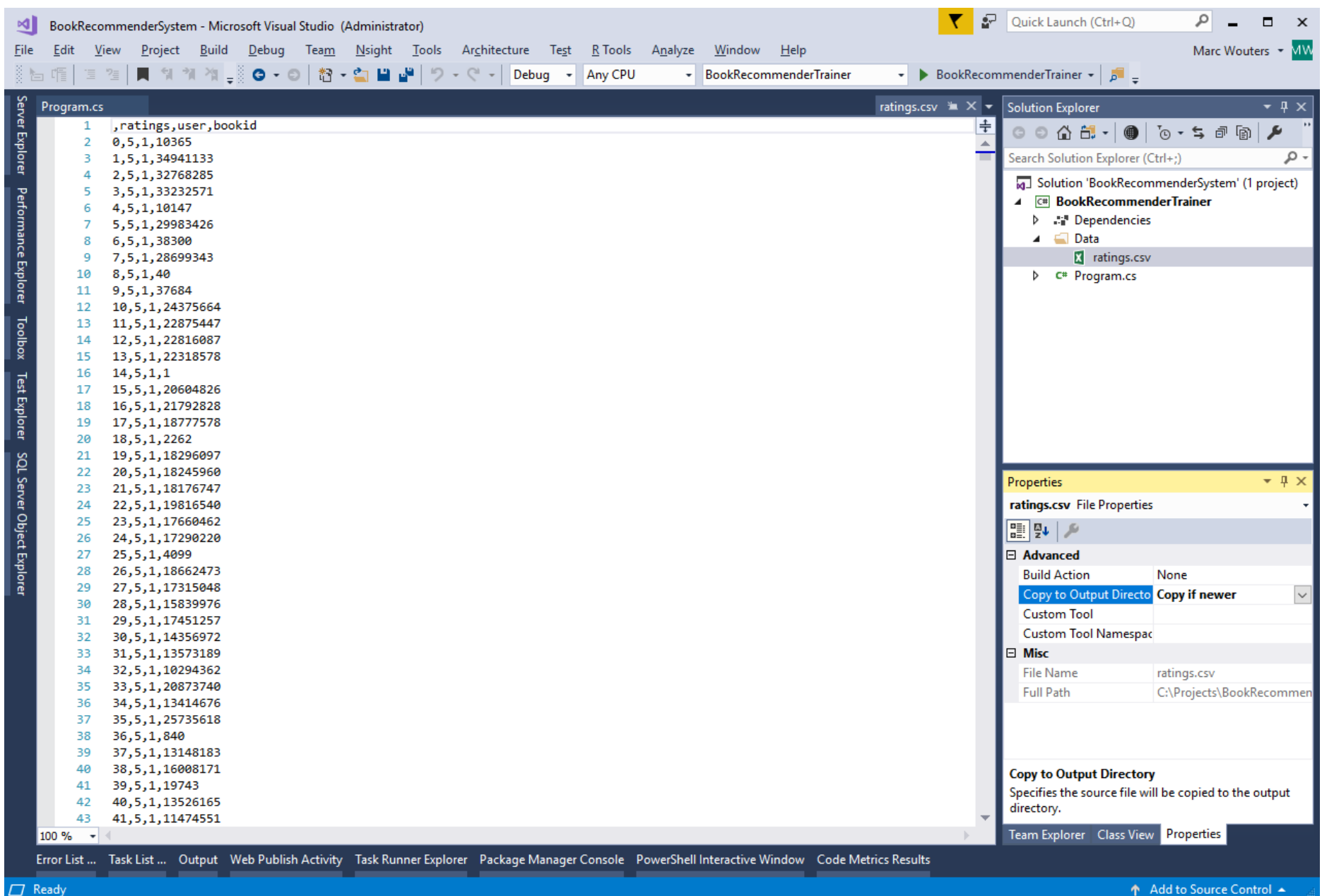
- At least a basic understanding of object-oriented programming
- Knowledge of C# or any other C based language
- Some form of IDE, I will be using Visual Studio 2017 v15.9.4
- .Net Core 2.0 or higher, this project will be using version 2.2
- Dataset downloaded from <https://www.kaggle.com/dicoderdisha/goodreads-dataset/version/1>
The dataset we are downloading from Kaggle is already cleaned and preprocessed. This process can be time consuming, so by skipping this step we can focus on the training and implementation of our machine learning model.

Model trainer

Our journey starts with creating our basic project setup so open Visual Studio and create a new .Net Core console application, name this new solution **BookRecommenderSystem** and name the application **BookRecommenderTrainer**.



Create a folder in the BookRecommenderTrainer project named **Data**. Extract the data from the archive you have downloaded earlier. Next copy the **ratings.csv** file into the **Data** folder. Now select the **ratings.csv** file and open the properties pane. Change **Copy to Output Directory** to **Copy if newer**, this will make locating the file a bit easier.



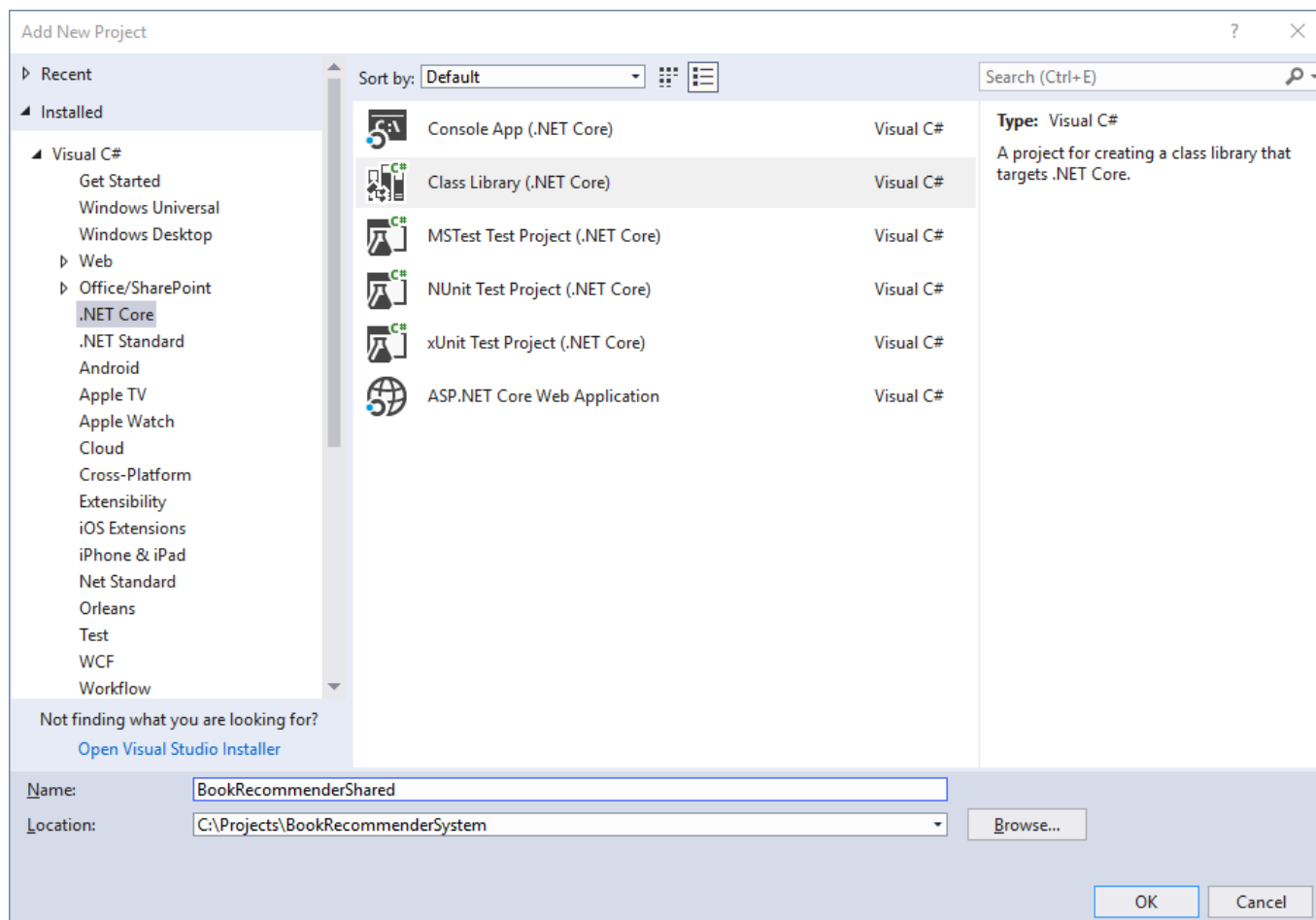
To create our machine learning pipeline, we need a few NuGet packages. Execute the commands below in the Package Manager Console or use the NuGet Package Manager.

```
Package Manager Console

1 Install-Package Microsoft.ML -Version 0.9.0
2 Install-Package Microsoft.ML.MatrixFactorization -Version 0.9.0
```

One of the next steps is to create a few classes to hold our data. These classes will be shared between our Trainer and Predictor so to prevent code duplication we will be adding a class library named **BookRecommenderShared** to our solution.

After you have added this project, install the NuGet packages mentioned above in this new project. We will not be using the generated **Class1.cs** file so remove it from the newly created project.



Time to explore our dataset. Open **ratings.csv** in an editor of choice. The first line contains the column headers/titles of our data, followed by many lines/records of data. For brevity I will summarize all columns below:

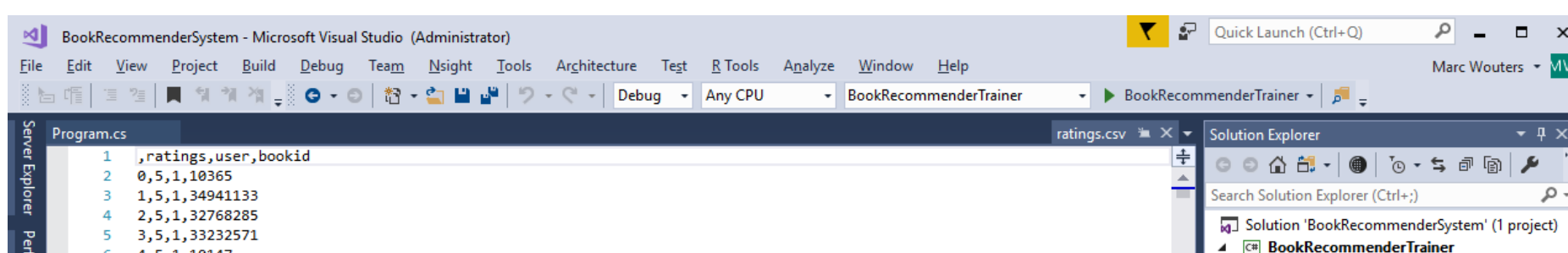
- Column 0: has no title but contains the rowId and was used as primary key
- Column 1: “ratings” contains a rating given by a user between 1 and 5
- Column 2: “user” refers to the userId and is used as a foreign key
- Column 3: “bookid” refers to the bookId and is used as a foreign key

Based on this data, we will be creating our class that will define the shape of the records used for training our model. Start by adding a new folder named **Models** to the shared project. Next create a new class named **BookRating** in the newly created folder and copy the contents below. If you are manually typing the code, watch out for the casing. The casing matches the headers of the csv exactly, so keep this in mind.

```
BookRating.cs
Raw

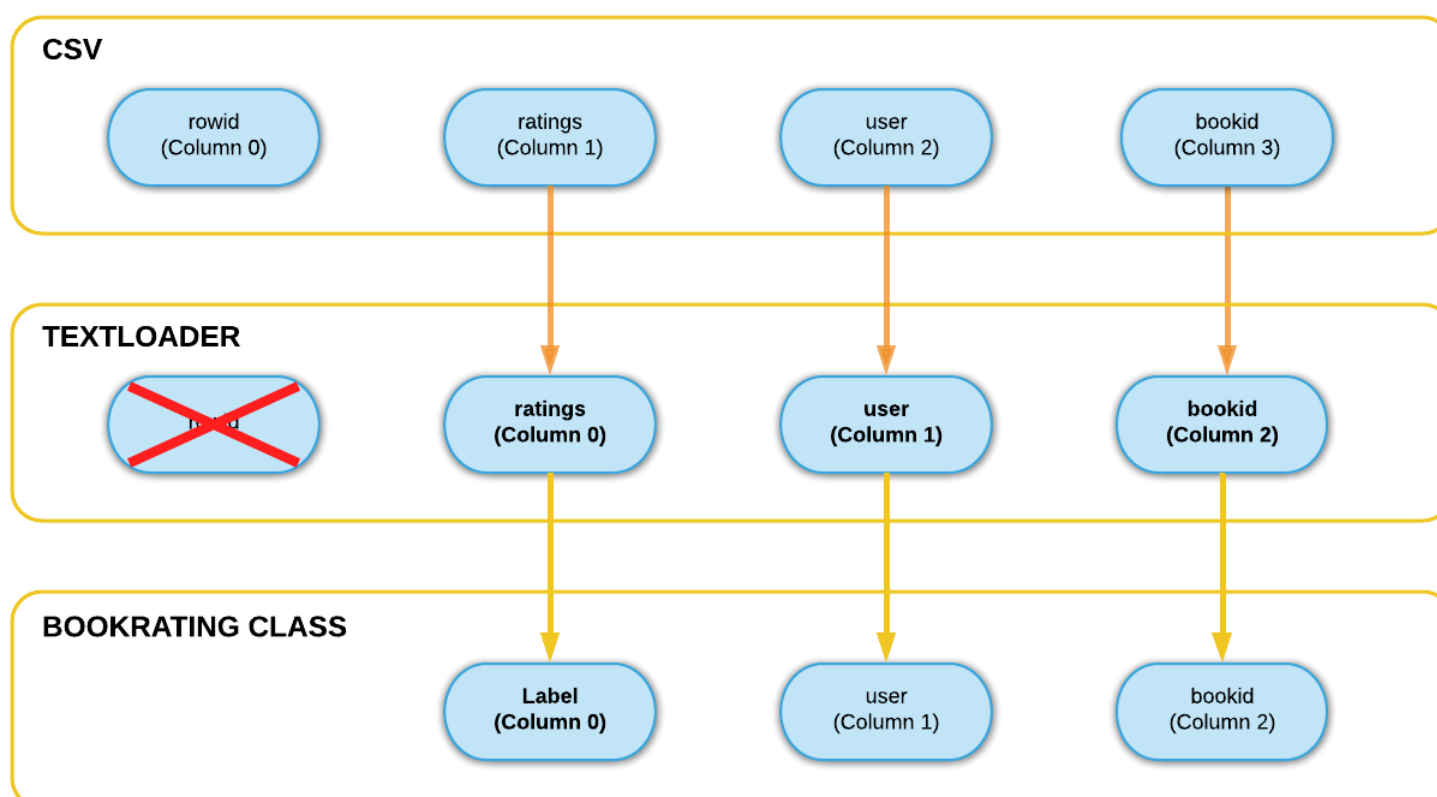
1  using BookRecommenderShared.Configuration;
2  using Microsoft.ML.Data;
3
4  namespace BookRecommenderShared.Models
5  {
6      public class BookRating
7      {
8          /*
9           * A machine learning label is the actual data field to be calculated.
10          * The ordinal property on the column attribute defines the column number in the dataset.
11          */
12          [Column(ordinal: "0", name: Labels.Label)]
13          public float Label;
14          [Column(ordinal: "1")]
15          public float user;
16          [Column(ordinal: "2")]
17          public float bookid;
18      }
19  }
```

<script src="https://gist.github.com/tiebird/d9739d791ed9786cf335ba82ac07ba18.js"></script>



Some of you may already notice the column numbers do not match with the column summary of the csv above. The primary key was not defined in the class because we are not going to use this key to train our model. Training data can hog memory very fast, so always be careful and only select data you really need. Check the diagram for a better understanding how the data will transform.

Data transformation



Let us finish our models by adding the class **BookRatingPrediction** to the **Models** folder.

```
BookRatingPrediction.cs

1  using Microsoft.ML.Data;
2
3  namespace BookRecommenderShared.Models
4  {
5      public class BookRatingPrediction
6      {
7          /*
8           * A machine learning label is the actual data field to be calculated.
9           */
10         [ColumnName("Label")]
11         public float Label;
12         /*
13          * Score defines the predicted value by the model
14          */
15         [ColumnName("Score")]
16         public float Score;
17     }
18 }
```

`<script src="https://gist.github.com/tiebird/0c8cc3107973c32965c0d46844a5bbf4.js"></script>`

Create a folder named **Configuration** in the root of the shared project and add a new class named **Labels**.

```
Labels.cs

1  namespace BookRecommenderShared.Configuration
2  {
3      public static class Labels
4      {
5          public const string Label = "Label";
6          public const string User = "user";
7          public const string BookId = "bookid";
8
9          public const string UserEncoded = "userEncoded";
10         public const string BookIdEncoded = "bookIdEncoded";
11     }
12 }
```

`<script src="https://gist.github.com/tiebird/f79fd51679d5be9047a9c7e8136b5324.js"></script>`

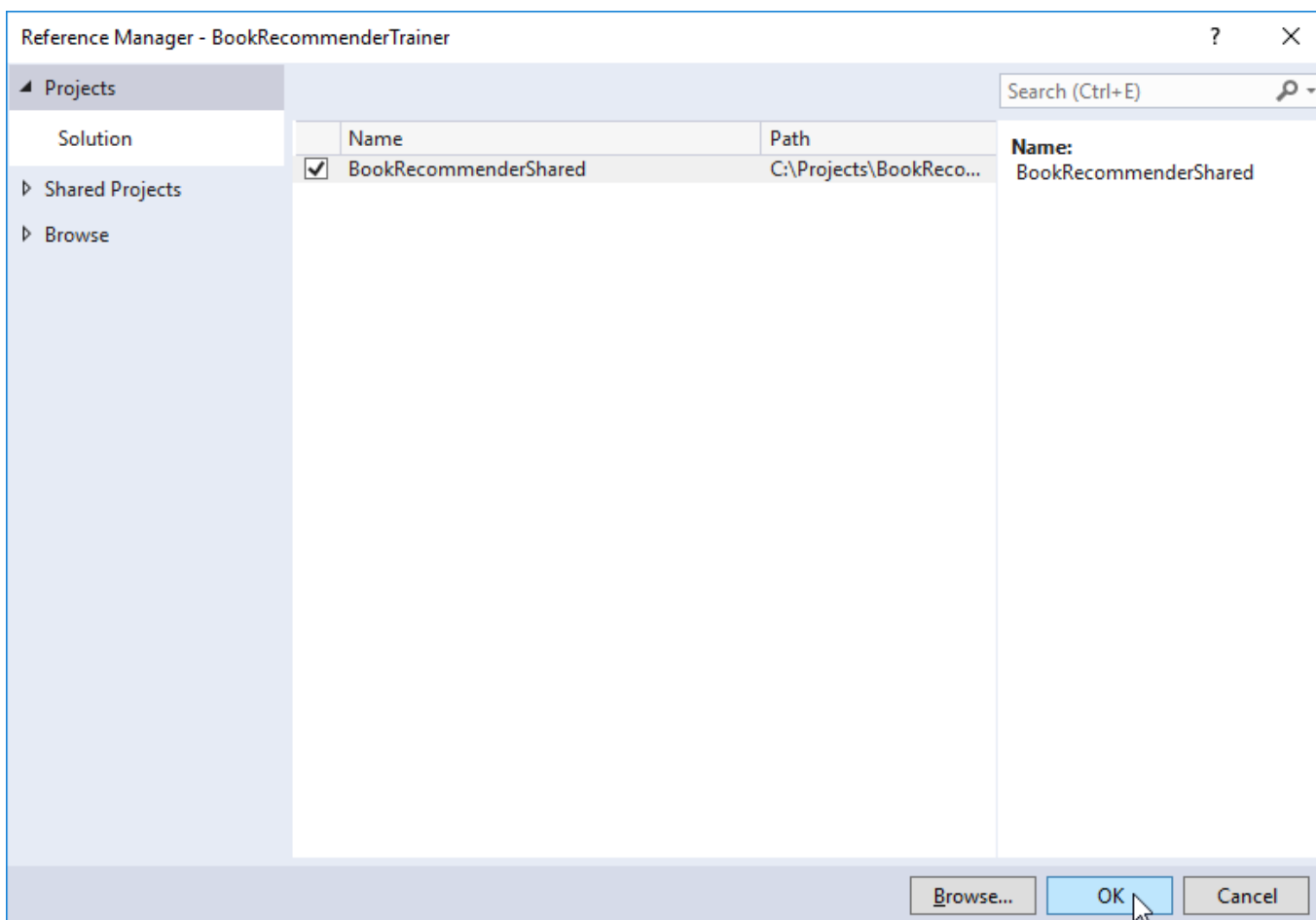
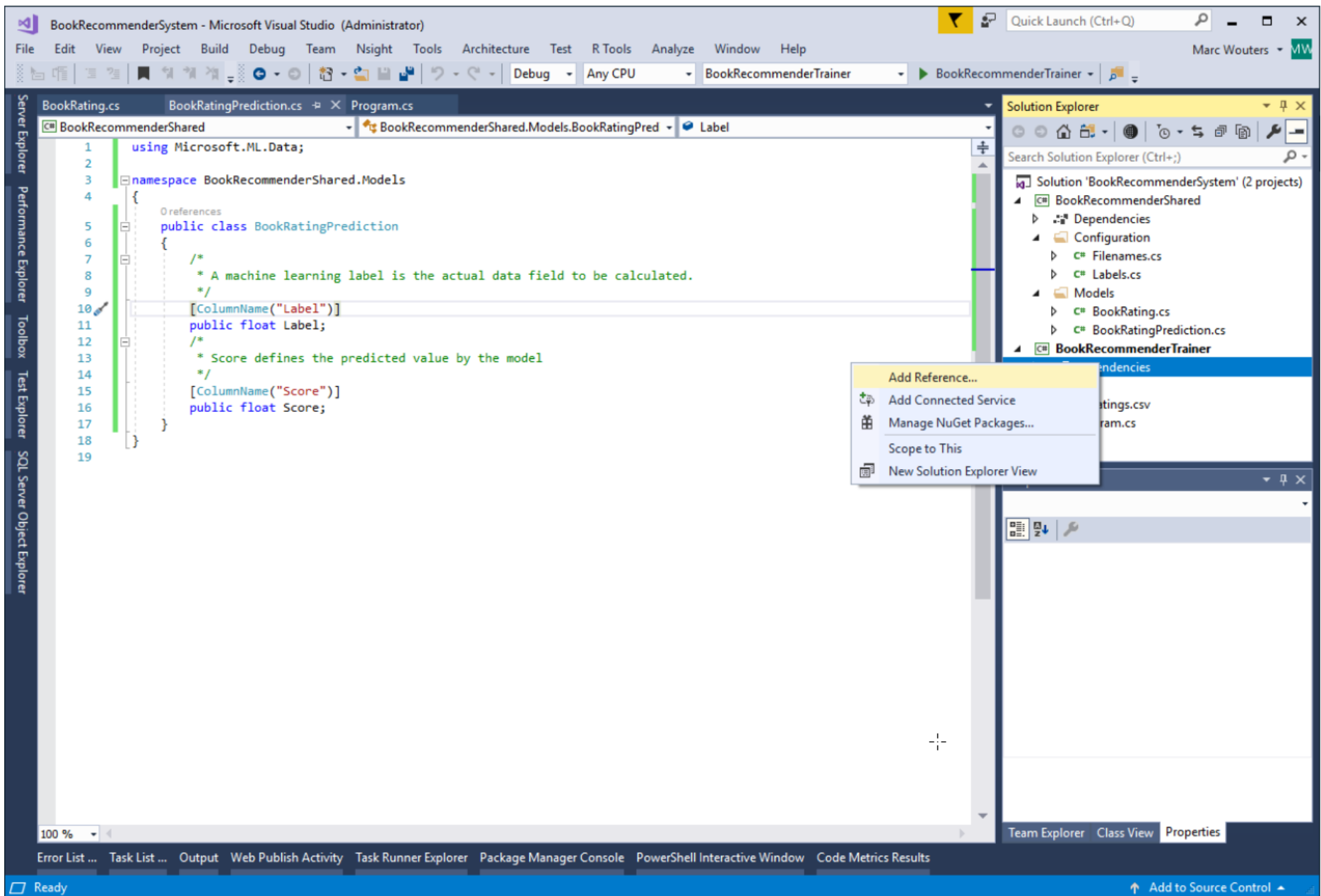
We need a few more shared properties so create another class called **FileNames** to the **Configuration** folder.

```
FileNames.cs

1  namespace BookRecommenderShared.Configuration
2  {
3      public static class FileNames
4      {
5          public const string DataFolder = "Data";
6          public const string RatingsDataset = "ratings.csv";
7          public const string BooksDataset = "bookfeatures.csv";
8          public const string TrainedModel = "rec_model.zip";
9      }
10 }
```

`<script src="https://gist.github.com/tiebird/50f69b25f3b43aa4604dd3789c027dd1.js"></script>`

Now, add the shared project as a reference to the **BookRecommenderTrainer** project and try building the solution.



Now the real fun begins. Open the **Program.cs** file from the trainer project. First, we will create a new machine learning context. This context will track all our data, transformations, optimizers and trainers. This will enable us to load data, edit or normalize where necessary and ultimately build a pipeline that will be used for analyzing the data for creation of our machine learning model. Replace the code in **Program.cs** with the code below.

Program.csRaw

```
1 using BookRecommenderShared.Configuration;
2 using Microsoft.ML;
3 using Microsoft.ML.Data;
4 using System;
5 using System.IO;
6
7 namespace BookRecommenderTrainer
8 {
9     class Program
10    {
11        static void Main(string[] args)
12        {
13            var mlContext = new MLContext();
14
15            var reader = mlContext.Data.CreateTextReader(new TextLoader.Arguments()
16            {
17                Separator = ",",
18                HasHeader = true,
19                /*
20                 * Define the columns to read be from the csv.
21                 * The indices are based on the position in the csv file and will receive a new index.
22                 * DataKind is defined as R4 (floating point) instead of integer
23                 * Computers are optimized for processing floating point calculations
24                 * so this will not create performance issues but mitigates some possible issues
25                 */
26                Column = new[]
27                {
28                    new TextLoader.Column(Labels.Label, DataKind.R4, 1), // new index 0
29                    new TextLoader.Column(Labels.User, DataKind.R4, 2), // new index 1
30                    new TextLoader.Column(Labels.BookId, DataKind.R4, 3) // new index 2
31                }
32            });
33
34            var data = reader.Read(Path.Combine(Environment.CurrentDirectory, Filenames.DataFolder, Filenames.RatingsDataset));
35
36            // Split our dataset in 80% training data and 20% testing data to evaluate our model
37            var (trainData, testData) = mlContext.BinaryClassification.TrainTestSplit(data, testFraction: 0.2);
38        }
39    }
40 }
```

`<script src="https://gist.github.com/tiebird/2fae84093101af1654cb35d4492b1a05.js"></script>`

As you can see, we have first created a TextReader that reads our csv file. The data is being read record by record and only the data defined in the column array will be stored in memory. The DataKind for each Column was specified as R4 which translates to a 16-bit floating point. In the introduction we have already mentioned GPU's for machine learning. You can think of GPU's as the masters of floating point (FP16) calculations. Changing your current machine learning algorithm from FP32 to FP16 can improve the performance in major ways, albeit this is not possible with every machine learning algorithm out there.

Our data is stored in memory, split into 2 different sets and ready to go. The next step is to create our machine learning pipeline. This pipeline will transform the data, feed the data to the algorithm and spit out a machine learning model. Now add these lines as stated below to the end of your main method. Do not forget to import the using statement: `"using Microsoft.ML.Trainers;"`

Program.csRaw

```
1 var pipeline = mlContext.Transforms.Conversion.MapValueToKey(Labels.User, Labels.UserEncoded)
2 .Append(mlContext.Transforms.Conversion.MapValueToKey(Labels.BookId, Labels.BookIdEncoded))
3 .Append(new MatrixFactorizationTrainer(mlContext, Labels.UserEncoded, Labels.BookIdEncoded, Labels.Label,
4     advancedSettings: s =>
5     {
6         s.NumIterations = 50;
7         s.K = 500;
8         s.NumThreads = 1;
9     }));
```

`<script src="https://gist.github.com/tiebird/f064250b5d42404d8c25274bc17b400d.js"></script>`

Setting up the pipeline is straightforward thanks to our earlier created mlContext. Firstly, we define some transforms to map our UserIds and our BookIds. This is necessary to vectorize them internally and use them for calculations. In the last step of the pipeline we create a new MatrixFactorizationTrainer. MF is one of the “magic” algorithms used to create recommender systems. The algorithm takes the data and splits it into smaller matrices, by doing this the processing can be parallelized over multiple threads.

Because we want to verify our results with 1 specific case, the configuration states to only use 1 thread, this is of course due to round-off errors. These settings also contain a few other parameters often called hyperparameters. They are used to optimize the process and are often experimented with manually or by using hyperparameters libraries. Feel free to experiment with these values, I have not optimized them for this article.

Time to create and train our model!

Program.cs

Raw

```
1 Console.WriteLine("Creating and training the model");
2 // Fit the data/train the model
3 var model = pipeline.Fit(trainData);
4 Console.WriteLine("Training finished");
```

<script src="https://gist.github.com/tiebird/de80b8e5fecf6ad88de96ffe37f1a563.js"></script>

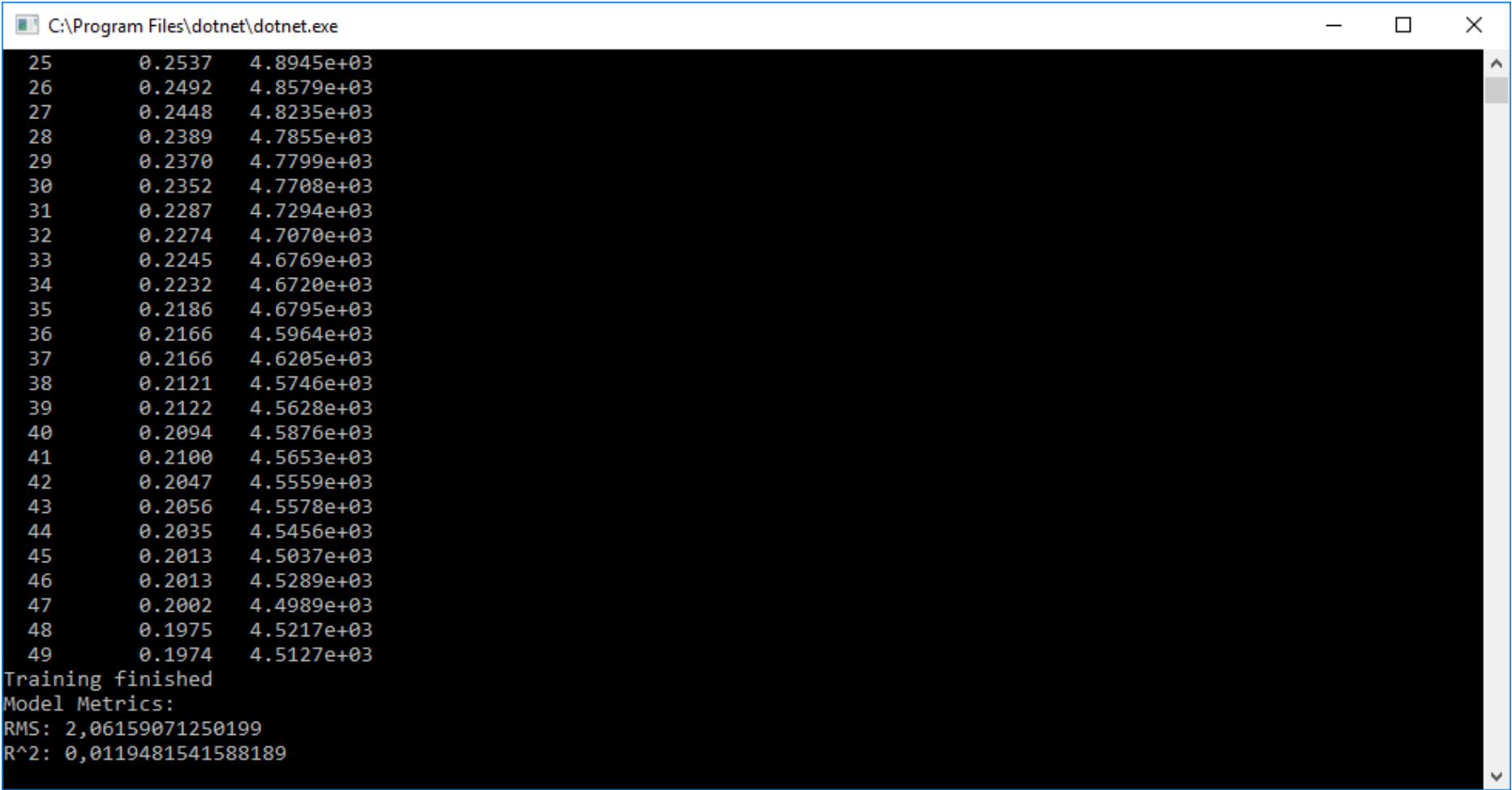
What? That’s it? We have barely typed anything... It could be wise to test our model and evaluate the performance. Add the code below and try running the application.

Program.cs

Raw

```
1 // Use the test data to evaluate the difference between the real value and the prediction
2 var prediction = model.Transform(testData);
3 var metrics = mlContext.Regression.Evaluate(prediction);
4 Console.WriteLine("Model Metrics:");
5 Console.WriteLine($"RMS: {metrics.Rms}");
6 Console.WriteLine($"R^2: {metrics.RSquared}");
7 Console.ReadLine();
```

<script src="https://gist.github.com/tiebird/d1ae66287c26fe19a58bf8629edd11a.js"></script>



There were no errors thrown and that is always a good thing. We have added the RMS (root mean square) and the R squared. The root mean square defines the average difference between the predicted values and the real values. R squared is a statistical measure to verify how good of a fit our regression model has. For this kind of recommender system, a value between 0.10 and 0.15 is reasonable. Unfortunately, RMS and R squared neither are a good method of evaluating the precision of our model. Validation of recommender systems is often tricky and would take up too much time for this article.

Visualizing a prediction as proof is one of the most common tasks requested by business people. To visualize our result, we will choose a fixed user and bookid. We will be using this case to verify the outcome in a different application by passing only the machine learning model and the fixed test case.


```
Program.cs
1      var predictionFunction = model.CreatePredictionEngine<BookRating, BookRatingPrediction>(mlContext);
2
3      var bookPrediction = predictionFunction.Predict(new BookRating
4      {
5          user = 91,
6          bookid = 10365
7      });
8
9      Console.WriteLine($"Predicted rating: {Math.Round(bookPrediction.Score, 1)}");
10     Console.ReadLine();
11
12     Console.WriteLine("Saving model");
13     using (var stream = new FileStream(
14         Path.Combine(Environment.CurrentDirectory, Filenames.TrainedModel),
15         FileMode.Create,
16         FileAccess.Write,
17         FileShare.Write))
18     {
19         mlContext.Model.Save(model, stream);
20         Console.WriteLine($"The model is saved as {Filenames.TrainedModel}");
21     }
22
23     Console.ReadLine();
```

`<script src="https://gist.github.com/tiebird/15b12841c056f23074d91fabf2d81f3a.js"></script>`

And of course, the output:

```
C:\Program Files\dotnet\dotnet.exe
30      0.2352  4.7708e+03
31      0.2287  4.7294e+03
32      0.2274  4.7070e+03
33      0.2245  4.6769e+03
34      0.2232  4.6720e+03
35      0.2186  4.6795e+03
36      0.2166  4.5964e+03
37      0.2166  4.6205e+03
38      0.2121  4.5746e+03
39      0.2122  4.5628e+03
40      0.2094  4.5876e+03
41      0.2100  4.5653e+03
42      0.2047  4.5559e+03
43      0.2056  4.5578e+03
44      0.2035  4.5456e+03
45      0.2013  4.5037e+03
46      0.2013  4.5289e+03
47      0.2002  4.4989e+03
48      0.1975  4.5217e+03
49      0.1974  4.5127e+03
Training finished
Model Metrics:
RMS: 2,06159071250199
R^2: 0,0119481541588189

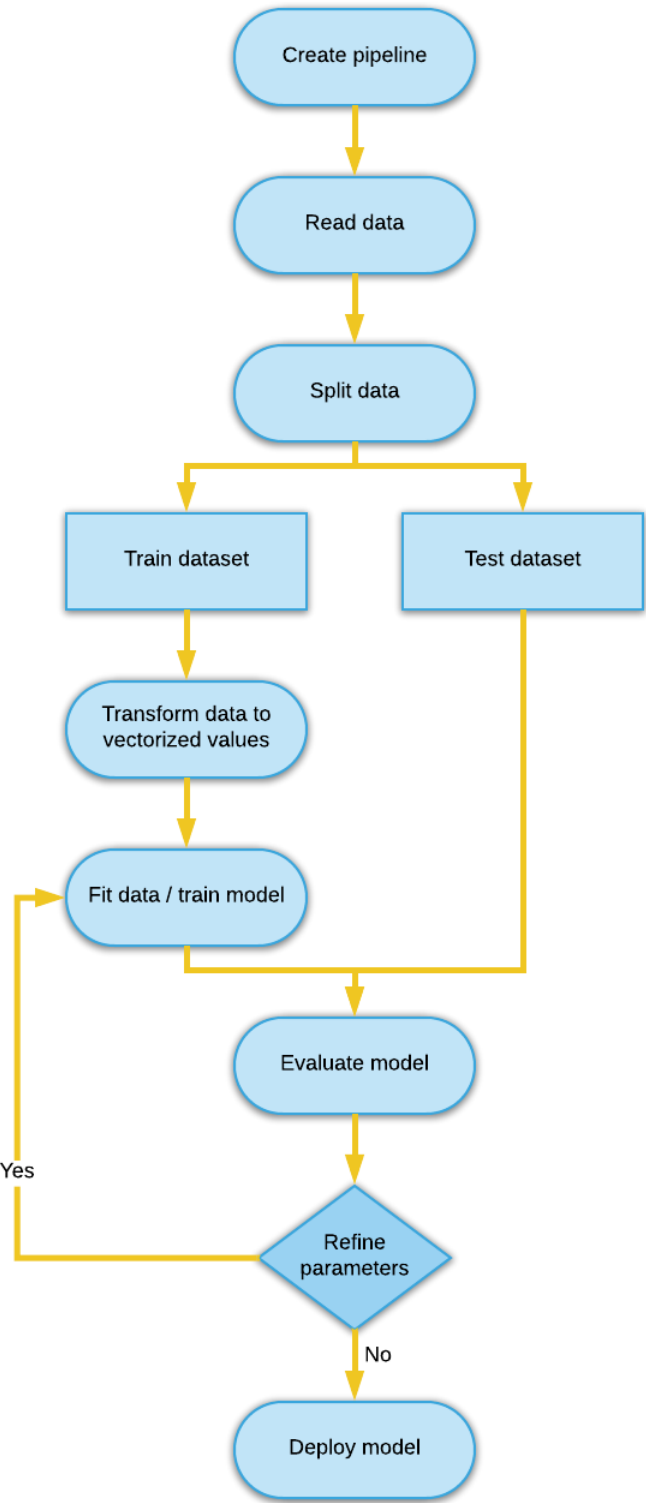
Predicted rating: 4,7

Saving model
The model is saved as rec_model.zip
```

To recapitulate our steps:

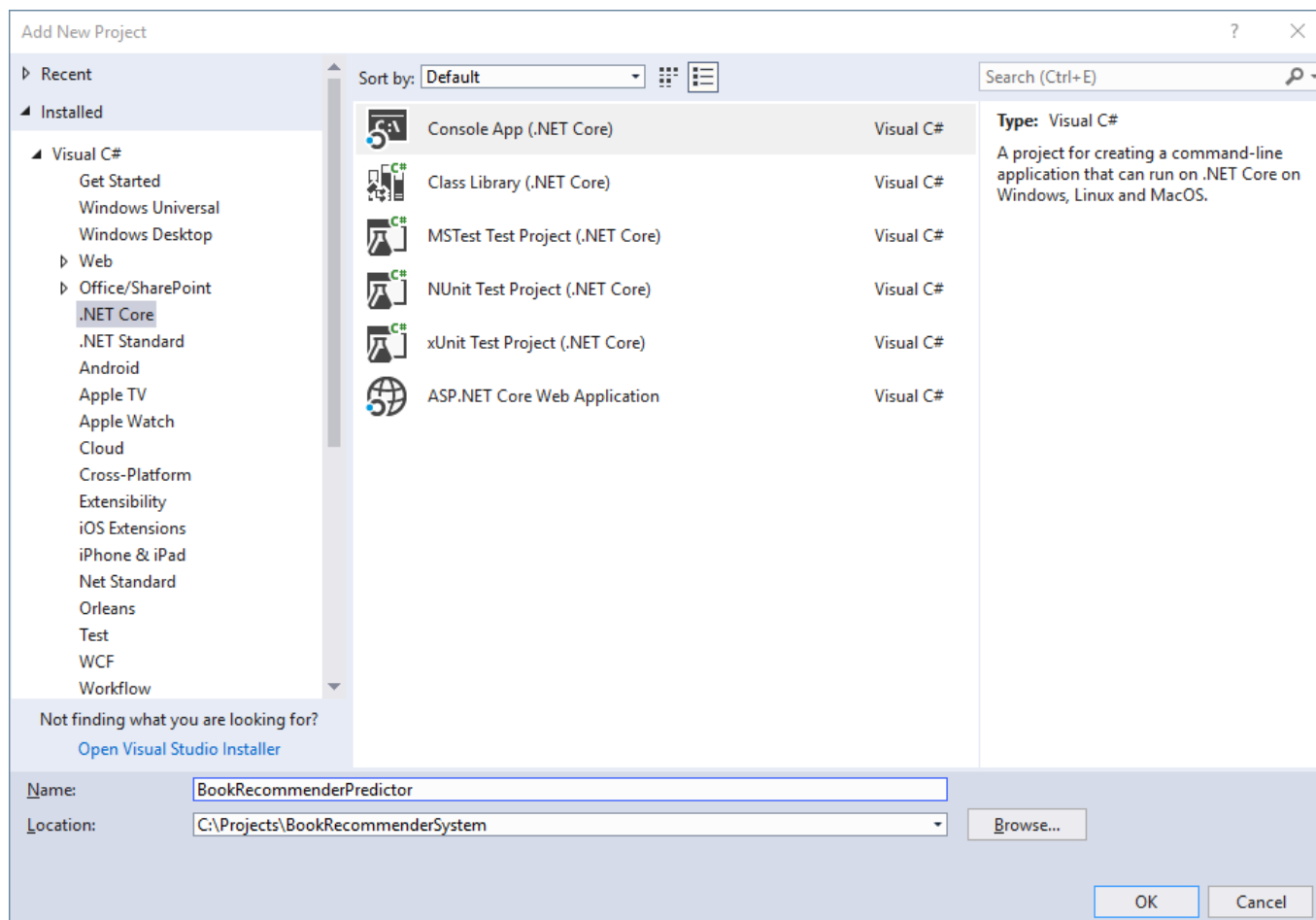
- We have created a machine learning pipeline;
- Read our data in to memory;
- Split our data into a training and a testing dataset;
- Transformed our values for calculation;
- Added a MatrixFactorizationTrainer to the pipeline;
- Trained our model;
- Evaluated the resulting model on our testing dataset;
- Visualized the prediction;
- Exported the model to an archive for easier import.

ML.Net Machine learning process



Value predictor

After creating our model on our research site, we want to deploy the model and validate everything before letting our algorithm loose on the world. Due to lack of an environment, we will add another .Net Core console project to the solution and call it **BookRecommenderPredictor**.



If we want to use our generated machine learning model, we need to copy the model file from the trainer to the output folder of our predictor.

Copy the **rec_model.zip** file from:

[project_location]\BookRecommenderSystem\BookRecommenderTrainer\bin\Debug\netcoreapp2.2

to:

[project_location]\BookRecommenderSystem\BookRecommenderPredictor\bin\Debug\netcoreapp2.2

Add both ML NuGet packages to our new project. As before, reference the **BookRecommenderShared** project. Open the **Program.cs** file of the **BookRecommenderPredictor** and replace the contents of the file. Do not forget to change the startup project to our new project afterwards.

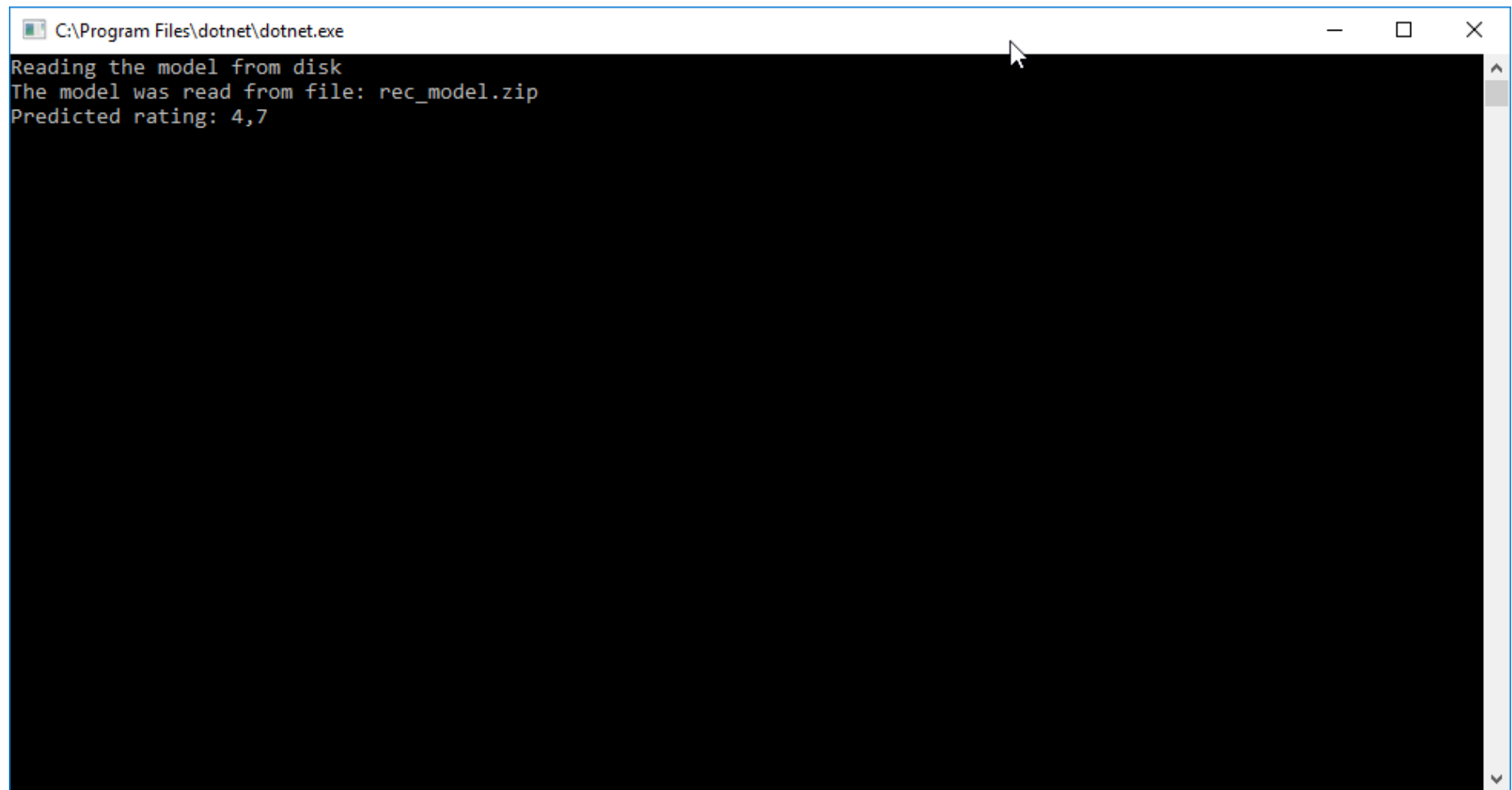
Program.cs

Raw

```
1  using BookRecommenderShared.Configuration;
2  using BookRecommenderShared.Models;
3  using Microsoft.ML;
4  using Microsoft.ML.Core.Data;
5  using System;
6  using System.IO;
7
8  namespace BookRecommenderPredictor
9  {
10     class Program
11     {
12         static void Main(string[] args)
13         {
14             var mlContext = new MLContext();
15
16             ITransformer loadedModel;
17
18             Console.WriteLine("Reading the model from disk");
19             // Load the model into memory
20             using (var file = File.OpenRead(Path.Combine(Environment.CurrentDirectory, Filenames.TrainedModel)))
21             {
22                 loadedModel = mlContext.Model.Load(file);
23                 Console.WriteLine($"The model was read from file: {Filenames.TrainedModel}");
24             }
25
26             // Create the prediction function from the loaded model
27             var predictionFunction = loadedModel.CreatePredictionEngine<BookRating, BookRatingPrediction>(mlContext);
28
29             // Predict the outcome
30             var bookPrediction = predictionFunction.Predict(new BookRating
31             {
32                 user = 91,
33                 bookid = 10365
34             });
35
36             Console.WriteLine($"Predicted rating: {Math.Round(bookPrediction.Score, 1)}");
37             Console.ReadLine();
38         }
39     }
40 }
```

`<script src="https://gist.github.com/tiebird/d8794fa0aae5709f5056d872f32aa132.js"></script>`

Build and run the application. Evaluate the output and compare with the image below.



You probably already noticed, we received the same results in our predictor as in our trainer. With this, we can conclude the export and import of the machine learning model was successful.

You can imagine it is not hard to roll out this model. To scale this, we could extract all books that we have not read from a database, calculate the predicted ratings and sort them in descending order. The first items would be books I probably would enjoy reading assuming the ratings are high enough.

If you have problems running the project or maybe some instructions were not clear enough, please feel free to download the samples from Github: https://github.com/tiebird/Ordina-ML.NET_Recommender_System

Next steps

To further your journey into the wonderful world of machine learning, I would recommend to try out the samples on the documentation page of [ML.NET](#). Just be careful to match the NuGet package versions to those in the tutorial. ML.NET is a rather young product that releases new versions on almost a monthly basis, often with breaking changes. The tutorials introduce you to common problems like binary classification, regression, clustering and how to solve them.

People wanting a deeper understanding of machine learning, can follow the [Microsoft Professional Program for Artificial Intelligence Track](#). You will not only learn the basics of machine learning but also the ethics, math and finally the knowledge to build your own shallow and deep machine learning models. All of this can be audited for free. There is a possibility to receive the certificates in exchange for small fees.

From the author

Machine learning has been a real journey for me, it made me more curious than ever about the future. It made me learn math and basic neuroscience as well as explore ethics and philosophy. I have a feeling my journey is far from over, but I am looking forward to exciting times ahead.

I also would like to thank Ordina for guiding me and giving me the chance to share my experiences with so many fantastic people. A lot of respect for the Techorama crew who gave us fantastic experiences in the past and will so going forward. Thank you, our reader, for taking your time to read this. I hope you were inspired and love to see you at Techorama 2019 or at one of the many AI meetups region Antwerp!



Marc is a Software Engineer at Ordina Belgium.

At Ordina he is a full stack .Net developer working with both front-end and back-end technologies.

Passionate about finding the right "cure" he is always looking for new ways to develop his skills.

Marc is constantly searching to improve quality and efficiency for both enterprise applications and himself.