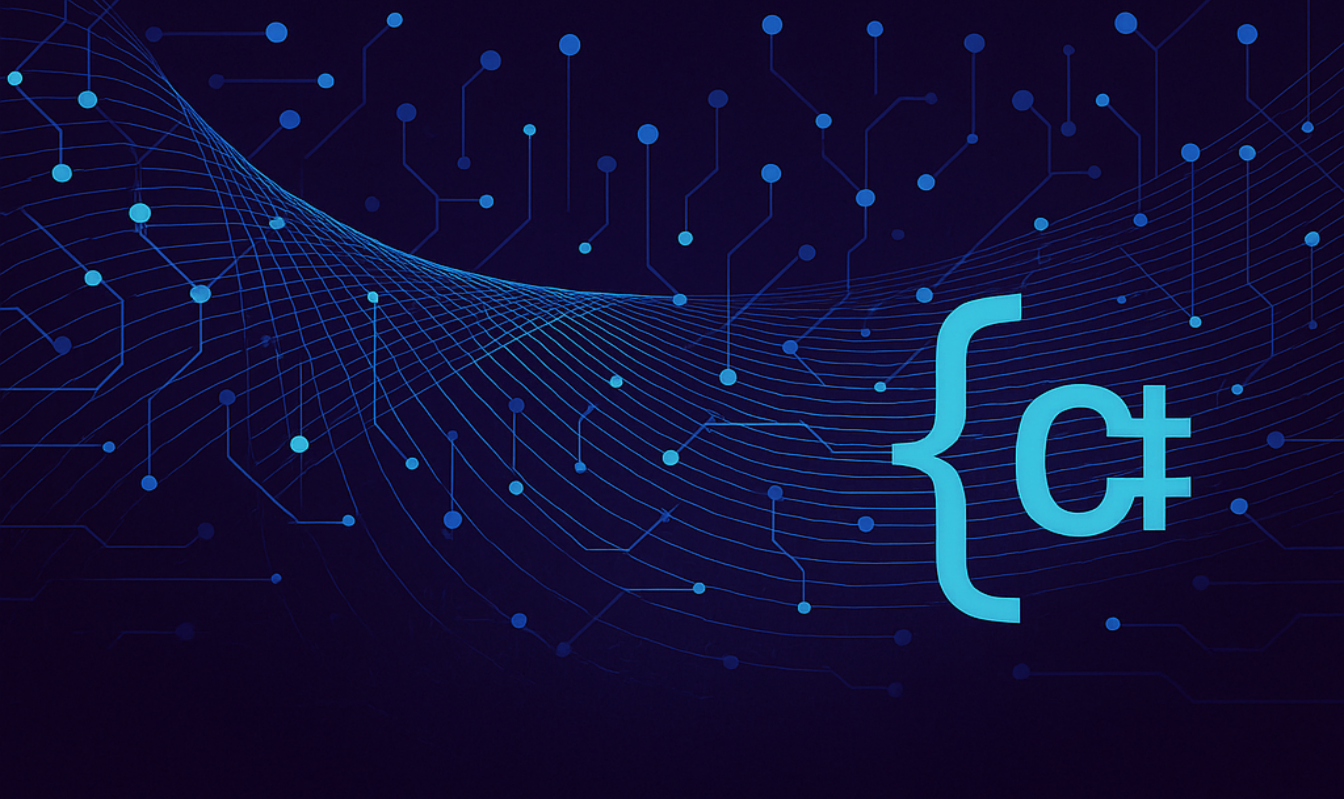
FIODAR SAZANAVETS

MACHINE LEARNING FOR C# DEVELOPERS

MADE EASY

Build smart applications with ML.NET



Machine Learning for C# Developers Made Easy

Build smart applications with ML.NET

Fiodar Sazanavets

This book is available at
https://leanpub.com/machine-learning-for-csharp-developers-made-easy

This version was published on 2025-11-10

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

I would like to say thank you to Luis Quintanilla, a program manager from Microsoft and a highly knowledgeable machine learning expert. He was responsible for overseeing the development of the tools we speak about in the book. He also gave useful tips that made this book possible.

Contents

1. ML.NET: C# developer's gateway to machine learning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

1.1 What is ML.NET?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

1.2 Why is machine learning in demand?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

1.3 What makes ML.NET powerful?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

1.3.1 The main benefits of choosing ML.NET

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

1.3.2 What makes ML.NET easier to use than alternatives

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

1.4 Where can ML.NET be used?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

1.4.1 Building products and services

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

1.4.2 Automating internal tasks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

1.5 Organizations that use ML.NET

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

1.5.1 How ML.NET helps Power BI identify key influencers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

1.5.2 Other organizations that rely on ML.NET

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

1.6 How can ML.NET build a useful model?

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

1.6.1 Selecting a task

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

1.6.2 Preparing the data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

1.6.3 Training and evaluating the model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

1.6.4 Using trained models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

1.7 Installing ML.NET

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

1.7.1 Minimal system requirements

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

1.7.2 Installing ML.NET via the command line

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

1.7.3 Updating the tool

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

1.7.4 Installing Model Builder in Visual Studio

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

2. Machine learning fundamentals

This chapter covers:

- The process of training a machine learning model
- Different types of machine learning
- Using ML.NET to perform supervised machine learning tasks
- Using ML.NET to perform unsupervised machine learning tasks

Although we don't need to be machine learning (ML) or data science experts to use ML.NET, there are still some basic concepts that we need to understand. Otherwise, we won't even hen we instantiate MLContextknow how to get ML.NET to solve a specific type of problem that we want it to solve.

What we don't necessarily need to know are the complex algorithms and mathematical formulae that are involved in machine learning. These things are important and machine learning is impossible without them, however, we can let the underlying pipeline deal with these concepts.

It's analogous to driving a car. To be a good driver, you don't need to know how the internal combustion engine of the car works, but you aren't getting anywhere unless you know where all the controls are and how to use them.

2.1 Types of machine learning

Broadly speaking, there are three types of machine paradigms, which can be outlined as follows:

- **Supervised Learning:** Supervised learning is a type of machine learning where the model learns from labeled training data. In this approach, the training dataset consists of input features and their corresponding target labels. For example, the features may represent various characteristics of a house and the label may represent the price of the house that the model is being trained to predict. The model's goal is to learn the mapping between the input features and the target labels, enabling it to make predictions on new, unseen data.

- **Unsupervised Learning:** Unsupervised learning involves training a model on unlabeled data. In this type of learning, the model's objective is to discover patterns, structures, or relationships within the data without any predefined labels. Unsupervised learning is often used for tasks such as clustering (i.e., arranging items with similar characteristics into distinct clusters), anomaly detection, dimensionality reduction, and data visualization.
- **Reinforcement Learning:** Reinforcement learning is a type of machine learning where an agent learns how to interact with an environment to maximize its cumulative reward. For example, an agent can be taught to play a video game and its "cumulative reward" is the score that the agent earns while playing it. The agent receives feedback through rewards or penalties based on its actions. Through trial and error, the agent explores the environment, learns optimal strategies, and takes action to maximize its long-term reward. Reinforcement learning has been successfully applied to various domains, including the above mentioned game playing, robotics, autonomous driving, and recommendation systems. Some notable algorithms in reinforcement learning include Q-learning, deep Q-networks (DQN), policy gradients, and actor-critic methods.

Figure 2.1 summarizes the different types of machine learning.

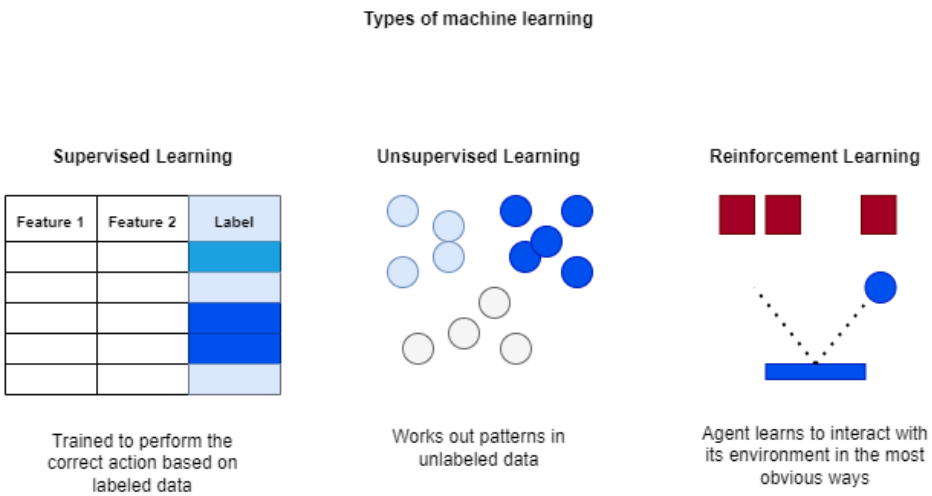


Figure 2.1. Different types of machine learning summarized

The important thing to remember from the ML.NET perspective is that it doesn't support reinforcement learning. At least, not at the time of writing.

However, supervised and unsupervised learning are fully supported and we will have a look at the examples of both later in this chapter.

Let's now examine each type of machine learning in more detail with some examples.

2.2 Supervised machine learning

In supervised learning, the model is trained on labeled data. In this context, a data record is equivalent to a table row. The row contains a special label column that we want to map to the remaining data. All other columns are known as features. For example, if we are building a model to predict house prices, the price of the house will be the label, while things like the number of bedrooms, number of bathrooms, name of the neighborhood, area of the house, etc. would be the features, as seen in figure 2.2.

Bedrooms	Bathrooms	Floor Area Square Feet	Neighborhood	Price
5	2	5382	Binfield	1 000 000
1	1	550	Easthampsted	150 000
2	1	800	Birch Hill	250 000
3	2	2520	Binfield	600 000
2	1	600	Birch Hill	200 000
4	2	3230	Lilly Hill Park	750 000
3	2	2890	Lilly Hill Park	650 000

Figure 2.2. Training data for house price predictions

The model's goal is to learn the mapping between the input features and the target labels, enabling it to make predictions on new data previously not seen by the model.

By leveraging labeled data, supervised machine learning allows the model to learn from existing patterns and make predictions or classifications on new, unseen data based on the learned relationships between features and labels. Some of the best-known examples of supervised learning are regression and classification. We will go over these supervised learning task types supported by ML.NET in detail in chapter 4. However, here is a quick overview of some of them.

2.2.1 Regression

Regression is a type of supervised learning used for predicting continuous numerical values. The target variable in regression represents a real-valued output. The goal is to learn a function that maps the input features to the target variable. Examples of regression problems include:

- **House Price Prediction:** Given features like size, number of bedrooms, location, etc., the model learns to predict the price of a house.
- **Stock Price Prediction:** Using historical stock prices, trading volumes, and other relevant factors, the model learns to predict future stock prices.
- **Medical Outcome Prediction:** Based on patient characteristics, medical history, and diagnostic tests, the model learns to predict the likelihood of a specific medical outcome, such as disease progression or treatment success.

Figure 2.3 summarizes how a regression model is trained.

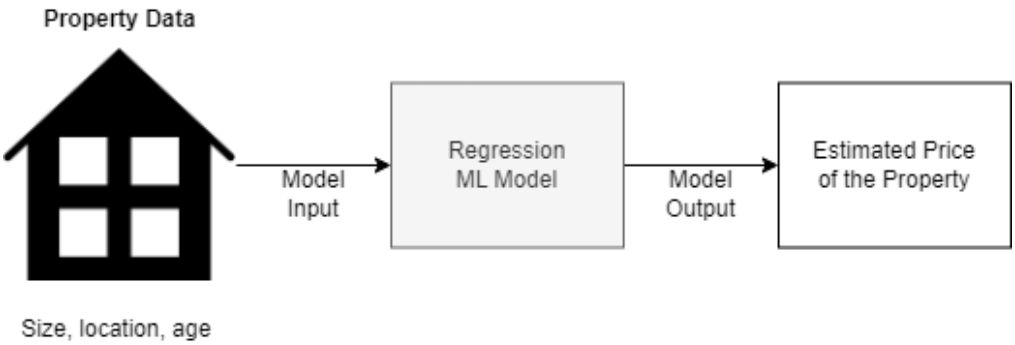


Figure 2.3. House price prediction by using a regression ML model

2.2.2 Classification

Classification is another type of supervised learning used for predicting discrete class labels or categorical outcomes. The target variable in classification represents a category or class label. The goal is to learn a decision boundary that separates different classes based on the input features. Examples of classification problems include:

- **Email spam detection:** Given email content and metadata, the model learns to classify emails as spam or non-spam.
- **Image classification:** Using images and corresponding labels, the model learns to classify images into predefined categories, such as identifying objects in photos.
- **Sentiment analysis:** Based on text data, the model learns to classify text as positive, negative, or neutral sentiment.

There are other examples of supervised learning, but all of them follow a common pattern. In any supervised learning task, the training data is labeled and the goal of ML algorithms is to figure out the relationship between features and the labels.

2.3 Unsupervised machine learning

Unsupervised learning allows the model to explore the data, find hidden structures or patterns, and derive insights without the need for labeled examples. It is particularly useful when working with large, unannotated datasets or when there is no prior knowledge about the underlying patterns.

Evaluation of unsupervised learning models is often challenging since there are no predefined labels to describe what each data point represents. The assessment may involve qualitative analysis, visualization techniques, or domain-specific measures depending on the problem domain.

2.3.1 Clustering

Clustering is a common unsupervised learning technique that groups similar instances together based on their intrinsic properties or characteristics. The goal is to identify natural clusters in the data. Examples of clustering problems include the following:

- **Customer segmentation:** Given customer data like age, income, and purchasing behavior, the model learns to group similar customers together for targeted marketing strategies. In this example, clusters can form around categories like recent graduates, high earners, retirees, etc.
- **Urban planning and land use:** Clustering can be applied in geographical information systems (GIS) to group similar regions or urban areas based on population density, land use, or infrastructure. This can help urban planners optimize resource allocation or identify areas needing development.

2.3.2 Anomaly detection

By learning the normal behavior of a system or process, the model can identify instances that deviate significantly from the norm, indicating anomalies or outliers. Here are some scenarios where anomaly detection would be useful:

- **Error detection in logs:** By utilizing anomaly detection, we can automate the process of identifying errors and unusual behavior in our application logs.
- **Fraud detection:** Data from credit card transactions can be analyzed and the transactions that look out of place can be flagged for fraud investigation.
- **Data cleanup:** Anomaly detection can also be used for pre-processing data for other tasks, such as supervised learning. The algorithms will be able to detect outliers and other anomalies that can be removed.

2.3.3 Association rule learning

Association rule learning discovers interesting relationships or associations between items in a dataset. It identifies datasets and the rules to express the relationship without those relationships being explicitly stated beforehand. These may include frequently occurring itemsets or item combinations. Once identifying such relationships, the model would then generate rules to express these relationships. Examples of association rule learning include:

- **Market basket analysis:** By analyzing transaction data, the model can identify items that are frequently purchased together, enabling businesses to make recommendations or optimize product placement.

- **Web clickstream analysis:** By examining user navigation patterns on websites, the model can discover associations between web pages, helping in personalization or content optimization.
- **Genomics:** In genomics, association rule learning can be used to identify genetic variants that are frequently co-occurring or associated with certain traits or diseases.

Figure 2.4 outlines how associated rule learning is applied to identify functional and non-functional DNA sequences.

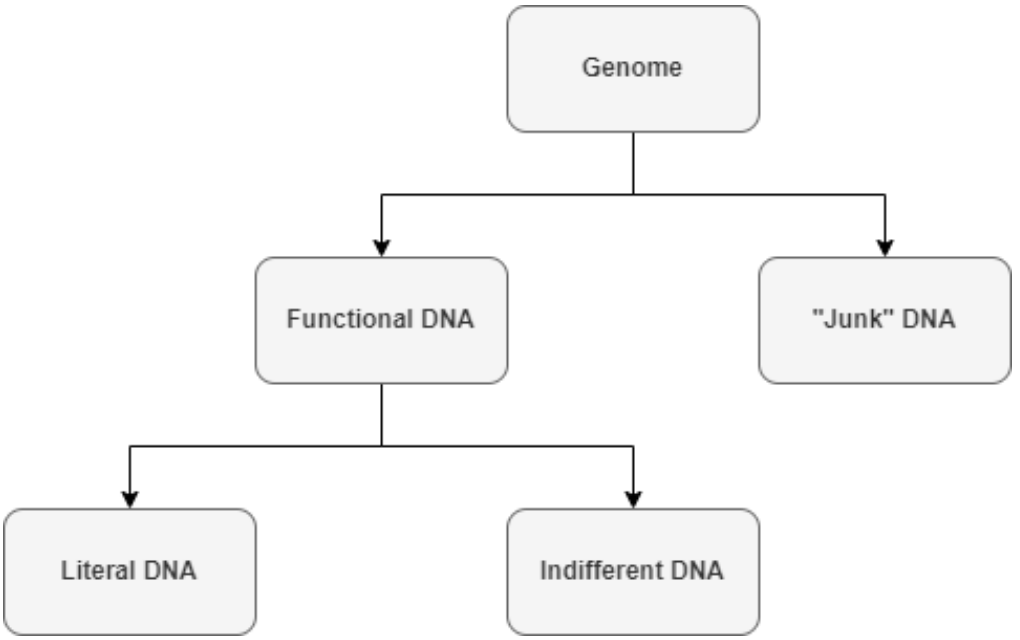


Figure 2.4. Identifying functional and non-functional DNA sequences

Regardless of what specific purpose we are using our ML algorithms for, unsupervised learning allows us to work with semi-raw data that requires very little pre-processing and no labeling at all. The algorithm itself will try to figure out how to process the data.

2.4 Reinforcement machine learning

Reinforcement learning enables the agent to learn optimal policies through interaction with an environment, making it suitable for problems where a

sequence of actions needs to be learned and optimized over time. It is particularly useful in scenarios with complex decision-making and sparse or delayed feedback.

In this context, the "agent" represents an entity that we are training to interact with its environment. For example, a common application of reinforcement learning is training a model to play a video game. In this case, the video game character would be the agent.

The agent receives feedback in the form of rewards or penalties based on its actions and it then changes its state according to this feedback. Through trial and error, the agent explores the environment, learns optimal strategies, and takes action to maximize its long-term reward.

Evaluation of reinforcement learning models often involves assessing the agent's performance in terms of the total amount of reward achieved or other domain-specific metrics.

2.4.1 The reinforcement learning process

- **Agent, Environment, State, and Actions:** In reinforcement learning, the learning process involves an agent and an environment. The agent takes actions in the environment based on its observations or state and receives feedback in the form of rewards or penalties. The goal of the agent is to learn a policy—a mapping from states to actions—that maximizes its cumulative reward over time.
- **Reward Signal:** The agent receives rewards or penalties from the environment based on its actions. The reward signal serves as feedback to guide the agent's learning. The agent's objective is to learn a policy that maximizes the total expected reward it receives over the long term.

Figure 2.5 provides an outline of how reinforcement learning is done.

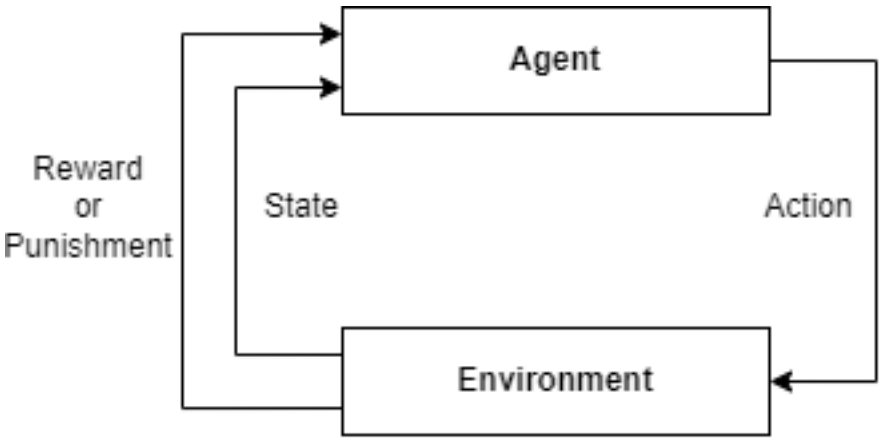


Figure 2.5. The reinforcement machine learning process

2.4.2 Exploitation and exploration

In reinforcement learning, the agent needs to balance exploration and exploitation. Exploration involves trying out different actions to learn about the environment and discover optimal strategies. Exploitation involves leveraging the learned knowledge to take actions that are expected to yield high rewards. The agent must strike a balance between exploring new actions and exploiting the learned information to make optimal decisions.

For example, let's imagine an AI model that is teaching itself to play a video game, such as a car racing simulator. It understands the concept of the finish position in the race and knows that it needs to try its best to complete the race before any other participants. It doesn't know how to do that yet.

It will start by getting familiar with the driving mechanics. In the process, it will learn how to control the car. Then it will learn that, in order to complete the race, it needs to follow the track. This is the exploration phase.

Once it gets familiar with the controls and other gameplay mechanics, it can start figuring out the most optimal ways of applying them to reach the finish line first. This is the exploitation phase.

2.4.3 Reinforcement learning applications

Some examples of reinforcement learning applications are as follows:

- **Game Playing:** Reinforcement learning has achieved remarkable success in game playing. For example, the AlphaGo algorithm learned to play the game of Go at a world champion level. The agent interacted with the Go board, received rewards for winning or penalties for losing, and learned optimal strategies through self-play and reinforcement learning.
- **Robotics:** Reinforcement learning can be used to train robots to perform tasks. For instance, a robot may learn to navigate a maze, manipulate objects, or perform complex movements by receiving rewards or penalties based on its actions and optimizing its policy through reinforcement learning.
- **Autonomous Vehicles:** Reinforcement learning can be employed in training autonomous vehicles to make decisions in real-world environments. The agent can learn to control the vehicle's acceleration, braking, and steering based on sensor inputs and feedback from the environment to optimize safety, efficiency, or other objectives.

As has been mentioned before, at the time of writing, ML.NET doesn't support reinforcement learning out of the box. However, it would still be useful to have a high-level understanding of what it is, as we have just discussed.

2.5 Using ML.NET for supervised learning

Let's now have a look at how different types of machine learning can be utilized by ML.NET. We will begin by going through the process of supervised learning to solve a simple sentiment analysis classification problem. Then we will apply unsupervised learning to build a clustering model.

Any supervised training process in ML.NET would use a so-called pipeline. Such a pipeline would consist of multiple processing steps. It will start by transforming the original values in the input data into numeric values that a training algorithm can understand. This will happen to both the feature and label columns. Then an appropriate training algorithm will be applied to train the model.

Figure 2.6 provides an overview of how an ML.NET training pipeline works.

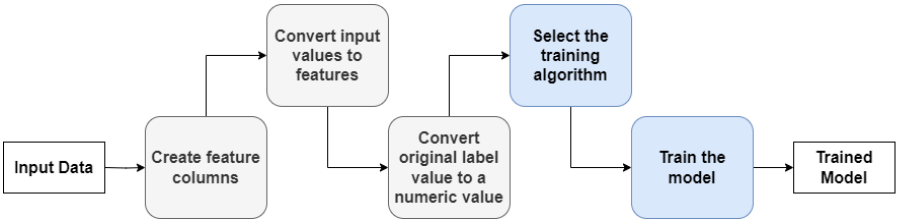


Figure 2.6. ML.NET training pipeline overview

Let's now delve deeper into these concepts and examine how the training pipeline works in ML.NET.

2.5.1 Training a classification model

Let's now examine how easy it is to perform a supervised machine learning task by using ML.NET. The task that we will use as an example is a binary classification problem. We will train our model to be able to recognize sentiment in sentences, in this case, whether the sentence constitutes a positive or negative review. The model will be able to detect whether the sentiment is positive or negative.

If you haven't installed the ML.NET CLI yet, you can do so by following the instructions from Appendix I.

To train our model, we will use a popular open-source dataset available via the following link:

https://github.com/microsoft/ML-Server-Python-Samples/blob/master/microsoftml/
analysis/yelp_labelled.txt

This dataset is represented by a text file called `yelp_labelled.txt`. The file contains tab-delimited data with two columns. The first column is a sentence taken from a product or service review on Yelp.com. The second column is a sentiment label. A value of 1 represents positive sentiment. A value of 0 represents negative sentiment. This is what the data looks like:

```
Wow... Loved this place.    1
Crust is not good.  0
Not tasty and the texture was just nasty.   0
Stopped by during the late May bank holiday off Rick Steve recommendation and
↪  loved it. 1
The selection on the menu was great and so were the prices. 1
Now I am getting angry and I want my damn pho.  0
Honeslty it didn't taste THAT fresh.)   0
The potatoes were like rubber and you could tell they had been made up ahead of
↪  time being kept under a warmer. 0
The fries were great too.   1
A great touch.  1
Service was very prompt.    1
Would not go back.  0
The cashier had no care what so ever on what I had to say it still ended up
↪  being wayyy overpriced. 0
I tried the Cape Cod ravoli, chicken,with cranberry...mmmm! 1
I was disgusted because I was pretty sure that was human hair.  0
I was shocked because no signs indicate cash only.  0
```

Figure 2.7 below demonstrates how this data is seen by the ML training algorithm.

Value	Label
Wow... Loved this place.	1
Crust is not good.	0
Not tasty and the texture was just nasty.	0
Stopped by during the late May bank holiday off Rick Steve recommendation and loved it.	1
The selection on the menu was great and so were the prices.	1
Now I am getting angry and I want my damn pho.	0
Honeslty it didn't taste THAT fresh.)	0
The potatoes were like rubber and you could tell they had been made up ahead of time being kept under a warmer.	0
The fries were great too.	1
A great touch.	1
Service was very prompt.	1

Figure 2.7. Data organized into a feature value and corresponding label

We can use the data to train a classification model. During training, various algorithms will run to try to figure out the relationship between the textual content of the sentence and the sentiment. The sentiment column is known as a "label". The role of the label in supervised learning is to tell the training algorithms what output value should be associated with specific input values.

In this particular case, the type of machine learning task we are performing is known as "classification". This is a process of mapping a specific value (or set of values) to a specific category. In our case, we only have two classes:

- Positive sentiment is represented by the value of one

- Negative sentiment is represented by the value of zero

As we only have two classes, we are dealing with a binary classification problem. If we had more than two classes, then the problem would be of a multiclass classification type.

We are training our model to be able to recognize the sentiment in the sentences that it hasn't seen before. Once the model is trained, we can feed any sentence into it and the model will try to predict whether the sentiment is positive or negative.

If we have the ML.NET command-line interface (CLI) tool installed, the process of training the model is very simple. All we have to do is execute the following command in a terminal of our choice (assuming we are in the same folder that we placed the `yelp_labelled.txt` in):

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

This example uses the `\\` character at the end of each line, which is a common way of splitting a single long command in a terminal into multiple lines in most command line terminals. However, if we run this command in Windows CMD, then we would use the `^` character instead. If we run it in PowerShell, the `'` character is used to split the command into multiple lines. These characters can also be removed and the command can be written in a single line.

This is how simple many supervised learning tasks are with ML.NET. Let's now break down the command to see what it's doing:

- The `mlnet` part is the name of the ML.NET CLI utility. Every ML.NET CLI command starts with it.
- The `classification` part is the name of the task. In our case, it's classification because we are training the model to classify data.
- The `--label-col` parameter represents the zero-based index of the label column. In our case, it's the second column; therefore, we are using the index of 1.

- The `--has-header` parameter tells our model trainer that the data set doesn't have the header row if this flag is set to false. In this case, no column names would be inferred from the headers and the first row will be recognized as a data row.
- Alternatively, if set to true (which is the default value), the model training algorithms would expect the first row to contain headers and no data.
- The `--name` parameter represents the name of the model.

Once we run this command, a .NET project will be created in the folder that we ran the command from with the trained model and the code that will allow us to consume the model. This is what we will have a look at next.

2.5.2 Auto-generated ML.NET project

The auto-generated ML.NET project is a standard .NET project. Therefore, it will have a file with the .csproj extension that will dictate the framework version, dependent libraries, and other types of project configuration. The content of such a file after executing the command above would be similar to the following:

Listing 2.1 ML.NET classification project structure

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net8.0</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.ML" Version="3.0.1" />
  </ItemGroup>
  <ItemGroup Label="DemoMLModel">
    <None Include="DemoMLModel.mlnet">
      <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>
    </None>
  </ItemGroup>
</Project>
```

Depending on the operating system we run the command on and the version of the tool, some of the model-related file references may not be added to the .csproj file. However, the files themselves will still be generated

inside the project folder. The PropertyGroup element at the beginning of the XML contains the standard top-level .NET project configuration, such as the target framework version and output type. Then we have the PackageReference element that references the Microsoft.ML NuGet package. This is the library from the NuGet repository that contains the code of the core ML.NET functionality. The next ItemGroup element arranges the auto-generated files that were created by the CLI tool.

In our case, the following files have been auto-generated by the process:

- `DemoMLModel.mbconfig`, which contains a detailed configuration of the training run.
- `DemoMLModel.training.cs`, which contains the code for retraining the model.
- `DemoMLModel.consumption.cs`, which contains the code for consuming the trained model.
- `DemoMLModel.mlnet`, which is the actual trained model. Please note that, depending on the version of the tool, the extension of this file may be either `.zip` or `.mlnet`. As you may have noted, all the names begin with DemoMLModel. This is because this is the name we specified as the value of the –name parameter in the original command.

Let's now open the C# files that were generated by the training process. First, we will examine the code that allows us to retrain the model.

2.5.3 Retraining the classification model

Sometimes we need to retrain an existing model. For example, we may want to do it if we get hold of a larger dataset than the one that was used to train the original model. Doing so may significantly improve the accuracy of the model.

The code for training and retraining the model can be found inside the `DemoMLModel.training.cs` file. Both training and retraining processes are performed by building a training pipeline with multiple processing steps and inputting the training data into it. The pipeline performs various transformation steps, applies an appropriate training algorithm, and creates the trained model. If we open it, we will find the RetrainModel method, which looks as follows:

Listing 2.2 Method for retraining a pipeline

```
public static ITransformer RetrainModel(MLContext mlContext, IDataView
↪  trainData)
{
    var pipeline = BuildPipeline(mlContext); // Building the pipeline
    var model = pipeline.Fit(trainData); // Training the model

    return model; // Returning the model
}
```

The parameters of this method are an instance of the MLContext class and an implementation of the IDataView interface. The MLContext is the class that provides the context for all operations related to ML.NET, including model training and model consumption. It is the entry point for any ML.NET operation. The IDataView interface represents the input data. The return type, which is an implementation of the ITransformer interface, represents a trained model.

In this method, we are calling the BuildPipeline method to build the training pipeline. Then we pass the data to the Fit method of the pipeline to train the model. Finally, we return the trained model.

Let's now have a look at the BuildPipeline method, which is used for building the training pipeline. The precise content of this method may be different after different model runs. This is because the ML trainers used by the ML.NET CLI tool don't have any preconceived notion of the relationship between the features and the label. The tools have to find these relationships experimentally by running multiple training algorithms and adjusting their parameters.

The final code will be generated based on the best-performing algorithm with the parameter values that return the most accurate results. Because the process of selecting the algorithm and adjusting its parameters is somewhat random, different runs of the tool may produce different code for the training pipeline even when the same input data is used. Here is one example of what this method may look like:

Listing 2.3 Method for building a pipeline

```
public static IEstimator<ITransformer> BuildPipeline(MLContext mlContext)
{
    // Data process configuration with pipeline data transformations
    var pipeline =
        mlContext.Transforms.Text.FeaturizeText(
            inputColumnName:@"col0",outputColumnName:@"col0")
        .Append(mlContext.Transforms.Concatenate(
            @"Features", new []{@"col0"}))
        .Append(
            mlContext.Transforms.Conversion.MapValueToKey(
                outputColumnName:@"col1",inputColumnName:@"col1"))
        .Append(mlContext.Transforms.NormalizeMinMax(
            @"Features", @"Features"))
        .Append(
            mlContext.MulticlassClassification
                .Trainers.OneVersusAll(
                binaryEstimator:
                    mlContext.BinaryClassification
                        .Trainers.LbfgsLogisticRegression(
                            new LbfgsLogisticRegressionBinaryTrainer.Options()
                            {
                                L1Regularization=0.0336147F,
                                L2Regularization=0.8686289F,
                                LabelColumnName=@"col1",
                                FeatureColumnName=@"Features"
        }), labelColumnName:@"col1"))
        .Append(
            mlContext.Transforms.Conversion.MapKeyToValue(
                 outputColumnName:@"PredictedLabel",
                 inputColumnName:@"PredictedLabel"));

    return pipeline;
}
```

Let's go through this example step-by-step. The first step is to featurize the data, which is the process of turning data columns into features. This is done by calling the `mlContext.Transforms.Text.FeaturizeText()` method:

```
var pipeline =
    mlContext.Transforms.Text.FeaturizeText(
        inputColumnName:@"col0",outputColumnName:@"col0")
```

Then we perform column mapping. We are doing so by calling the Append method on the pipeline. The Append method is used by the pipeline to append steps to it. This is what the column mapping step looks like:

```
.Append(mlContext.Transforms.Concatenate(
    @"Features", new []{@"col0"}))
```

The next step in the pipeline is to convert any human-readable label text values to numeric data. All unique label values will be identified, and each will be converted into a unique integer value. This is necessary because machine learning algorithms are math-based; therefore, they work with numeric data. This is what the step looks like:

```
.Append(
    mlContext.Transforms.Conversion.MapValueToKey(
        outputColumnName:@"col1",
        inputColumnName:@"col1"))
```

In our case, this step isn't strictly necessary because our labels are already numeric. This step was added by our training command because the command can't always accurately infer the data type in the column. But labels used for training are often textual rather than numeric; therefore, it would still be useful to know to apply this step if necessary.

The next step in our process is to normalize the input features. This is also done to convert textual values to numeric data. In our case, the algorithm converts sentences into decimal numbers between -1 and 1. This is what this step looks like:

```
    .Append(mlContext.Transforms.NormalizeMinMax(
        @"Features", @"Features"))
```

Then we are configuring the training algorithm, which, in our case, was chosen experimentally when we ran the CLI command to train the model. The algorithm contains some parameters that were selected during training. The values were also applied to these parameters automatically by trial and error. We will examine them in more detail in later chapters. This is what the training algorithm application looks like in the code:

```
.Append(
    mlContext.MulticlassClassification.Trainers.OneVersusAll(
        binaryEstimator:
            mlContext.BinaryClassification
                .Trainers.LbfgsLogisticRegression(
                    new LbfgsLogisticRegressionBinaryTrainer.Options()
                    {
                        L1Regularization=0.0336147F,
                        L2Regularization=0.8686289F,
                        LabelColumnName=@"col1",
                        FeatureColumnName=@"Features"
                    }), labelColumnName:@"col1"))
```

Finally, we configure the output data. In the following code, we are mapping the raw output from the model to the `PredictedLabel` column in our output data object. This is how it's done:

```
.Append(
    mlContext.Transforms.Conversion.MapKeyToValue(
        outputColumnName:@"PredictedLabel",
        inputColumnName:@"PredictedLabel"));
```

Next, we will have a look at the code that allows us to consume a trained model. In our scenario, this is how we can pass any random sentence to the model. The model will then try to predict whether the sentiment in the sentence is positive or negative.

2.5.4 Consuming a trained classification model

All our code for model consumption can be found inside the `DemoMLModel.consumption.cs` file. First, we will examine the class that represents the model input. This is what it looks like:

Listing 2.4 Model input data object

```
public class ModelInput
{
    [LoadColumn(0)]
    [ColumnName(@"col0")]
    public string Col0 { get; set; }

    [LoadColumn(1)]
    [ColumnName(@"col1")]
    public string Col1 { get; set; }
}
```

Here, we have representations of two columns, as we had in our data. As we had no headers in our original dataset, the names given to the columns were `col0` and `col1`. However, we can rename the columns if we want to.

Let's now have a look at the output object. This is how it's defined:

Listing 2.5 Model output data object

```
public class ModelOutput
{
    [ColumnName(@"col0")]
    public float[] Col0 { get; set; }

    [ColumnName(@"col1")]
    public uint Col1 { get; set; }

    [ColumnName(@"Features")]
    public float[] Features { get; set; }

    [ColumnName(@"PredictedLabel")]
    public float PredictedLabel { get; set; }

    [ColumnName(@"Score")]
    public float[] Score { get; set; }
}
```

In this object, we have two columns that represent transformed input data. We use numeric data type because the ML engine doesn't understand the human-readable text and string data type. This data gets converted into numeric data before it's processed.

But the most important properties in this object are PredictedLabel and Score. The PredictedLabel property shows which class the input data belongs to according to the model. The Score property contains the numeric values showing the likelihood of input data belonging to each of the possible classes.

Once we define the shapes of our input and output objects, we can have the code in place that would allow us to consume the model. The first thing we want to do is define the model path. This is done by the following expression:

```
private static string MLNetModelPath =
    Path.GetFullPath("DemoMLModel.mlnet");
```

Next, we need some code in place to load the prediction engine. This is represented by the PredictEngine property defined as follows:

```
public static readonly Lazy<PredictionEngine
    <ModelInput, ModelOutput>> PredictEngine
        = new Lazy<PredictionEngine<ModelInput, ModelOutput>>(
            () => CreatePredictEngine(), true);
```

This property loads the prediction engine in lazy mode, which means that it's only ever loaded when we actually need it and not when the object is instantiated. The prediction engine is loaded via the following method:

```
private static PredictionEngine
        <ModelInput, ModelOutput> CreatePredictEngine()
{
    // Initializing MLContext that is required for model consumption
    var mlContext = new MLContext();

    // Loading the model from the specified file path
    ITransformer mlModel =
        mlContext.Model.Load(MLNetModelPath, out var _);

    // Creating the prediction engine from the model
    return mlContext.Model
        .CreatePredictionEngine
            <ModelInput, ModelOutput>(mlModel);
}
```

In this method, we load the model from the path specified. Then we create a prediction engine from the model. The prediction engine is the logical implementation of the model that can be executed as any other code.

Finally, we have the following method that allows us to make a prediction based on the input data:

```
public static ModelOutput Predict(ModelInput input)
{
    // Retrieving prediction engine value
    var predEngine = PredictEngine.Value;

    // Using the prediction engine to make the prediction
    return predEngine.Predict(input);
}
```

The method obtains the prediction engine. Once the engine is loaded, it calls the Predict method on the engine to make the prediction and return the output.

This covers the basics of using ML.NET for supervised learning tasks. We learned how to train the model and generate the wrapper code by executing the ML.NET CLI utility. We also learned how this code works so we can modify it if necessary.

As we can see from this example, ML.NET is capable of automating several steps of the machine learning process. It was able to evaluate multiple training algorithms and select the best-performing one, so we didn't have to do it by hand. It was able to tune the algorithm by automatically applying appropriate parameters to it. All we had to do was prepare the data and specify the task.

We will have a more detailed look in the next chapter at how the best training algorithm is selected and how the quality and accuracy of the trained model are assessed.

2.6 Using ML.NET for unsupervised learning

If we want to do unsupervised learning in ML.NET, we cannot rely on the mlnet CLI utility. This is because, at the time of writing, the utility doesn't have an appropriate command to do it. The only way to do it is via the ML.NET NuGet libraries; therefore, we will have to build the training pipeline ourselves. Let's have a look at an example of how to do it.

For our scenario, we will be using a modified open-source dataset showing property prices in Lisbon, Portugal. The original dataset can be obtained via the link below:

https://www.openml.org/search?type=data&status=active&id=43660

The original dataset doesn't adhere to the format that can be easily understood by the ML model, so we need to turn it into a standard CSV where different columns are separated by commas, and there are no rows other than those that contain the headers and the data.

If you want to follow along with the exercises in this chapter, you'll need to modify the dataset first as I have done. Here are the steps you will need to follow:

- Remove the multi-line header with the descriptive metadata, as this data was unstructured and would confuse the training algorithms.
- Remove all textual columns. This is done to make the model faster and to only leave numerical data for the model that it can work with without conversion. The easiest way to do it is by opening the file in Excel and using GUI to delete the columns.
- Finally, add the header row for better readability based on the header descriptions from the original metadata.

After these modifications, we have a CSV file with content that looks like this:

```
1  Bedrooms,Bathrooms,Net Area,Gross Area,Parking Spots,Latitude,Longitude,Price
   ↪  per Square Meter,Full Price
2  3,1,76,152,0,38.7792,-9.1186,2463,198000
3  5,3,190,380,0,38.7056,-9.1784,3125,1270000
4  1,1,26,52,0,38.7058,-9.1639,4005,140000
5  5,4,185,370,0,38.7466,-9.164,3412,995000
6  7,1,150,300,0,38.7323,-9.1287,3277,570000
7  3,2,95,190,0,38.6965,-9.2099,3542,425000
8  3,1,76,152,0,38.7544,-9.1129,2881,130000
```

The goal of our unsupervised learning task is to identify patterns in the data and split the data into three distinct categories based on their similarities. For example, in the world of real estate investment where such data could be used, we could use such a task to automatically split the data into high-end, cheap, and mid-market properties.

We can perform the following task in any .NET project type. To keep it simple, we may want to do it inside a basic Console App project. All we need to do is add the Microsoft.ML NuGet package to the project.

Before we can start our task, we need to define input and output data objects. The input data object will be defined as follows:

Listing 2.6 Input data object

```
internal class Input

{
    [LoadColumn(0)]
    public float NumberOfBedrooms;

    [LoadColumn(1)]
    public float NumberOfBathrooms;

    [LoadColumn(2)]
    public float NetArea;

    [LoadColumn(3)]
    public float GrossArea;

    [LoadColumn(4)]
    public float ParkingSpots;

    [LoadColumn(5)]
    public float Latitude;

    [LoadColumn(6)]
    public float Longitude;

    [LoadColumn(7)]
    public float PricePerSqMeter;

    [LoadColumn(8)]
    public float FullPrice;
}
```

It is a class where each property represents a column from the CSV file that we use as the input. Our output data class will look as follows:

Listing 2.7 Output data object

```
internal class Output
{
    [ColumnName("PredictedClusterId")]
    public uint PredictedClusterId;

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

The class contains the arbitrarily assigned cluster id of an item, which is the identifier of the group that the item most likely belongs in. The cluster id represents a unique numeric identifier. The first cluster will be given the value of 1, the second cluster will be given the value of 2, and so on. However, the order of clusters will be arbitrarily decided by the training algorithm and can be different in different model runs.

The Distances property represents the distances from each of the available clusters. The lower the distance is to a particular cluster, the more likely the item is to belong in this cluster. This is a mathematical measure of how similar the items are to each other and not the geographic distance between the properties. Once we have defined the shapes of our input and output objects, we can start training our model. The first thing we will do is define all global variables and load the data. The example below assumes that the input data is represented by the lisbon_house_prices.csv file that has been placed in the Data folder inside the project folder and that this file, along with its folder, has been configured to be copied into the output location during the build. The following code demonstrates how we can define the input file path, define the model path, initialize MLContext, and finally load the data:

Listing 2.8 Preparing model inputs and outputs

```
string dataPath =
    Path.Combine(
        Environment.CurrentDirectory,
           "Data", "lisbon_house_prices.csv"); // Path of the input file

string modelPath =
    Path.Combine(
        Environment.CurrentDirectory,
            "Data", "ClusteringModelDemo.mlnet"); // Path of the trained model
            ↪  file

MLContext mlContext = new MLContext(seed: 0); // Instantiating MLContext

IDataView dataView =
    mlContext.Data.LoadFromTextFile<Input>(
        dataPath,
        hasHeader: false,
        separatorChar: ','); // Loading training data into the view
```

When we instantiate `MLContext`, we are passing the seed value of 0. The training process uses a random number generator and this parameter defines the number that we start from.

When we start, we don't have the trained model yet. The `modelPath` variable represents the full path of the file that we will save the model to during the training process. Then we will build the pipeline to train our model. This is done via the following code:

Listing 2.9 Building the pipeline

```
var pipeline = mlContext.Transforms.Concatenate("Features",
    "NumberOfBedrooms",
    "NumberOfBathrooms",
    "NetArea",
    "GrossArea",
    "ParkingSpots",
    "Latitude",
    "Longitude",
    "PricePerSqM",
    "FullPrice")

.Append(mlContext.Clustering.Trainers.KMeans(
    "Features", numberOfClusters: 3));
```

In this code, we are first concatenating all columns from the input data into a single column called "Features". If we look at the string values inside the

Concatenate method call, we can see that they all correspond to the column names from our input file. Then, we are calling the Append method to apply the training algorithm, which, in our case, is KMeans. We will talk about this algorithm in more detail in subsequent chapters.

Please note that one of the parameters of this method, numberOfClusters, is set to the value of 3. This will split the data into three distinct clusters, but it's an arbitrary choice in this case. We can define any number of clusters we want. We then train our model. We do so by passing the data into the Fit method of the pipeline object, as the following code demonstrates:

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

Once the model has been trained, we can save it into the file located at the path we previously specified. This is done by executing the following code:

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

Next, once the model has been trained and saved to the disk, we can evaluate it by inputting an arbitrary item into it and recording the prediction. First, we will need to create a prediction engine based on the model we trained earlier. This is done by the following code, where we are passing the model object into the CreatePredictionEngine method:

```
var predictor =
    mlContext.Model.CreatePredictionEngine
        <Input, Output>(model);
```

Next, we will create an input object and populate its features with some arbitrary values, as shown below:

Listing 2.10 Creating input object for a prediction

```
var house = new Input
{
    NumberOfBedrooms = 5f,
    NumberOfBathrooms = 3f,
    NetArea = 225f,
    GrossArea = 450f,
    ParkingSpots = 2f,
    Latitude = 38.7451f,
    Longitude = -9.179f,
    PricePerSqM = 4700f,
    FullPrice = 2115000f
};
```

After this, we can pass this object into the Predict method of the prediction engine to determine which cluster such an object would belong in:

```
var prediction = predictor.Predict(house);
```

Finally, we can print the identifier of the predicted cluster and the distance values into the console via the following statements:

```
Console.WriteLine($"Cluster: {
    prediction.PredictedClusterId}");
Console.WriteLine($"Distances: {
    string.Join(" ", prediction.Distances ??
Array.Empty<float>())}");
```

The `Distances` property of the output object represents how close the item is predicted to be to the center of each cluster. The closer the item is to the center of a particular cluster—the more likely it is to belong to this cluster. The distances help us to identify items that aren't strongly identified with a particular cluster, so we can re-examine them. The expected output should look similar to this:

```
Cluster: 1
Distances: 4.4521163E+12 4.457347E+12 4.460883E+12
```

This is an unsupervised learning task known as clustering. It is unsupervised because there is no label column. We aren't telling the training pipeline what each data item represents. It's entirely up to the training pipeline to figure out the relationships between the data items.

Summary

- There are three types of machine learning: supervised learning, unsupervised learning, and reinforcement learning.
- Supervised learning is the process of associating a set of values with a specific label value.
- Unsupervised learning doesn't use labeled data and its goal is to find patterns and structures in the data without knowing anything about it beforehand.
- Reinforcement learning is about teaching an agent to interact with its environment in a way that obtains the maximum reward for itself.
- In the language of machine learning, features are equivalent to data columns that are different from the label column.
- ML.NET supports supervised and unsupervised learning out of the box, but it doesn't currently do end-to-end reinforcement learning.
- One of the biggest benefits of using ML.NET is that it automates multiple stages of the machine learning process that are usually time-consuming and require specialist data science knowledge.

3. Selecting a problem for ML to solve

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

3.1 Selecting an ML task

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

3.1.1 Deciding between supervised and unsupervised training

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

3.1.2 Choosing a supervised learning task

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

3.1.3 Choosing an unsupervised learning task

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

3.2 Supervised training tasks available in ML.NET

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

3.2.1 Binary and multiclass classification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

3.2.2 Image classification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

3.2.3 Regression

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

3.2.4 Recommendation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

3.2.5 Forecasting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

3.2.6 Other supervised learning tasks supported by ML.NET

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

3.3 Unsupervised training tasks available in ML.NET

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

3.3.1 The usage of clustering in ML.NET

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

3.3.2 The usage of anomaly detection in ML.NET

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

3.4 Evaluating the training results

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

3.4.1 Quality metrics for supervised learning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

3.4.2 Accuracy metrics for unsupervised learning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

3.4.3 Accuracy metrics in ML.NET

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

4. Training a virtual assistant

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

4.1 Setting up our project

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 4.1 CLI interface placeholder

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

4.2 Classifying inputs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

4.2.1 Training the classification model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

4.2.2 Reviewing the classification pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 4.2 Multiclass classification pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

4.2.3 Making our model consumable

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Class library

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 4.3 Tech support classification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

4.2.4 Interacting with the model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 4.4 Application start-up code with classification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

4.3 Estimating numeric values

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

4.3.1 Training the regression model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

4.3.2 Reviewing the regression pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 4.5 Regression training pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

4.3.3 Adding the regression capabilities to the virtual assistant

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 4.6 Tech support classification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 4.7 Application start-up code with a regression task

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

4.4 Adding recommendation engine

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

4.4.1 Training the recommendation model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

4.4.2 Reviewing the recommendation pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 4.8 Recommendation pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

4.4.3 Adding the recommendation capabilities to the virtual assistant

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 4.9 Recommendation logic

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 4.10 Application startup code with recommendation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

4.5 Teaching our assistant to perform forecasting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

4.5.1 Training the forecasting model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

4.5.2 Reviewing the forecasting pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 4.11 Forecasting training pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

4.5.3 Enabling forecasting in the virtual assistant app

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 4.12 Forecasting consumption logic

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 4.13 Complete application startup logic

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

5. Detecting patterns with unsupervised learning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

5.1 Detecting patterns in the data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 5.1 Input data item

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 5.2 Output data item

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 5.3 data transfer object with input data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

5.1.1 Adding clustering code

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 5.4 Cluster data storage

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

5.1.2 Adding the model consumption code

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 5.5 Cluster prediction class

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

5.1.3 Adding the model to the virtual assistant

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 5.6 Adding the predictor to the virtual assistant

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 5.7 Application start-up code with clustering

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

5.1.4 Testing clustering functionality

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

5.2 Detecting anomalies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 5.8 Anomaly detection input data item

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 5.9 Anomaly detection output

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

5.2.1 Adding anomaly detection code

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 5.10 The entry method for anomaly detection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

5.2.2 Adding anomaly detector to the virtual assistant

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 5.11 Anomaly detection consumption logic

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 5.12 Complete application entry point logic

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

5.2.3 Testing our anomaly detection logic

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

5.3 Removing noise in the data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

5.4 Project: building our own clustering models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

5.4.1 Building a real estate categorization model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 5.13 Input data object for clustering model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

5.4.2 Building a code smells categorization model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

6. Building an intelligent chatbot

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

6.1 Introduction to natural language processing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

6.2 Applying sentence similarity

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

6.2.1 Training a sentence similarity model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 6.1 Input data for sentence similarity

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 6.2 Output data for sentence similarity

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 6.3 Training pipeline for sentence similarity

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

6.2.2 Consuming the sentence similarity model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 6.4 Prediction engine for sentence similarity

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

6.2.3 Adding sentence similarity checker to the virtual assistant

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 6.5 Sentence Comparer helper class

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

6.2.4 Pre-processing sentence similarity data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

6.2.5 extracting features from text tokens

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

6.2.6 Determining the similarity between two sentences

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

6.3 Applying text classification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

6.4 Teaching the chatbot to answer questions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 6.6 Chatbot input processor

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

6.4.1 Hallucinations in AI and where they come from

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

6.5 Named entity recognition technique

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

6.5.1 Training a NER model with ML.NET

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 6.7 NER input class

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 6.8 NER output class

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 6.9 NER label class

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 6.10 seeding label data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 6.11 training data processor

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 6.12 Application entry point

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

6.5.2 Using the NER model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

6.5.3 How NER works

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

6.5.4 Practical applications of NER

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

6.5.5 An example of a NER training model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

6.6 Project: refining chatbot capabilities

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

6.6.1 Enhancing the training data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

6.6.2 Guarding against hallucinations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 6.13 Text classification output class

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

6.6.3 Incorporating NER in our chatbot

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

7. Computer vision and making sense of images

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

7.1 Training an image classification model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

7.1.1 A brief overview of TensorFlow

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

7.1.2 Training pipeline for image classification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 7.1 Image classification pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 7.2 Loading image data from folders

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 7.3 Image classification input object

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 7.4 Image classification output object

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

7.1.3 Consuming image classification model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 7.5 Console interface for image

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

7.2 Training an object detection model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

7.2.1 Training pipeline for object detection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 7.6 Object detection training pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 7.7 Object detection training process

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 7.8 Object detection input model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 7.9 Object detection output model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

7.2.2 Consuming an object detection model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 7.10 Using a trained object detection model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

7.3 Integrating with TensorFlow for image classification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

7.3.1 Adding TensorFlow integration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 7.11 TensorFlow Inception settings

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 7.12 Input data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Listing 7.13 Prediction data structure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

7.4 How computer vision works

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

7.4.1 Image classification: recognizing what's in an image

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

7.4.2 Object detection: locating and identifying objects

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

7.5 Project: building an intelligent shopping system

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Adding models to any .NET apps with Model Builder

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

8.1 Introduction to Model Builder

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

8.2 Choosing a model to train

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

8.3 Optimizing the build

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

8.4 Evaluating model results

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

8.5 Project: building a complex ML app

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/machine-learning-for-csharp-developers-made-easy.