Microsoft

Introducing Machine Learning

Professional

Dino Esposito
Francesco Esposito

Microsoft

Introducing Machine Learning

Dino Esposito
Francesco Esposito

Introducing Machine Learning

Published with the authorization of Microsoft Corporation by:

Pearson Education, Inc.

ISBN-13: 978-0-13-556566-7

ISBN-10: 0-13-556566-9

Library of Congress Control Number: 2019954810

ScoutAutomatedPrintCode

Editor-in-Chief
Brett Bartow

Executive Editor
Loretta Yates

Development Editor
Mark Renfrow

Assistant Sponsoring Editor
Charvi Arora

Managing Editor
Sandra Schroeder

Senior Project Editor
Tonya Simpson

Copy Editor
Chuck Hutchinson

Indexer
Cheryl Ann Lenser

Proofreader
Abigail Manheim

Technical Editor
Cesar De la Torre Llorente

Editorial Assistant
Cindy Teeters

Cover Designer
Twist Creative, Seattle

Cover Image
IMOGI graphics

Compositor
codeMantra

Dedications

To Michela and her dreams

To my loved ones, to whom I couldn't help but dedicate a book

This page intentionally left blank

Contents at a Glance

Contents

Chapter 19 Sentiment Analysis: An End-to-End Solution 309

PART V FINAL THOUGHTS

Chapter 20 AI Cloud Services for the Real World 327

This page intentionally left blank

Acknowledgments

Writing a book with your son is a special experience even when it's the umpteenth book you write. For this one, I just put down in (hopefully clear) words Francesco's thoughts, vision, and his deep, and largely unexplained, understanding of machine learning. I definitely learned a lot from writing as much as I hope you will learn from reading.

If I learned a lot, well, that was mostly because of two people.

It's not the first time I have done some writing under the technical supervision of Cesar De la Torre Llorente, and it's always been a heavenly experience. I love his pragmatism and accuracy in devising, before designing, software products. He's currently principal program manager on the .NET product group at Microsoft and is in charge of the development of ML.NET. This is not specifically a book on ML.NET, but if the parts of the book that illustrate the .NET way to machine learning are accurate, well, that's because of the great help we received from Cesar.

There's an aspect in renewable energy that is little known: you need intelligent software to make it happen. At least on a functional level, it is vital to make accurate production, outage, fault, and price forecasts. Now, I don't think there are many people on this planet with a decade's worth of experience in this area that only recently was appointed the label *artificial intelligence*. Tiago Santos has been our guide in the random forest of machine learning and real-world artificial intelligence. "AI is just software" is now our shared motto.

If I've been able to give my career yet another turn (from Windows to web development and from software architecture to machine learning), it's also because two other people keep my creativity constantly stimulated. Giorgio Garcia-Agreda of Crionet made real my dreams as a tennis fanatic come true, up to singing "Easy like Sunday morning" in front of the tennis bigwigs. Simone Massaro of BaxEnergy discovered a fascinating new space where my renewable energy as a thinker can be freely expressed, sometimes even in front of top managers.

Any book is the result of teamwork, and it is our pleasure to call out the names of those who ultimately made it possible: Loretta Yates, as the acquisition editor; Charvi Arora, the managing editor, and Tonya Simpson, production editor.

—Dino

I finished high school one year early, and all I wanted was some money to practice as a professional investor. I had the wrong parents, though, and that approach didn't work. So, I asked my dad how to make money. "That's your problem," he said. "I can only teach you all I know." So, he taught me how to do things right and forgot to teach me how to do it wrong. As a result, today, we make the same mistakes in software. At this point, with some money in my hands, I was blissfully neglecting college when, on a hot summer afternoon, my dad told me, "Be honest: if you don't want to train your brain further, resign from college." As a result, a month later, I was back in class with a radically different mindset. I love mathematics, and I can't live doing anything different from it.

Then I met Gianfranco—friend, business partner, father, grandfather. He's a real professional investor, and he too taught me how to do things right and forgot to teach me how to do it wrong. As a result, today, we make the same mistakes in finance.

At school, at work, in the stock market, I study and try things. Sometimes they work, sometimes they don't, and whenever they don't, I learn something. It's the stick and carrot principle: the essence of learning for humans and machines. This book stems from my obsession for mathematical rigor and my dad's obsession for clarity. We used the stick on ourselves during the writing to ensure that carrots would be available during the reading.

This book is for you, Mom, because you'd love me anyway, regardless of triumph, disaster, or other impostors. This is for you, Maicol, because you'd love me even more if I stopped making noise on Sunday mornings.

This is for you, Alessandro, because you remind me when it's time to stop, and for you, Antonino, because you remind me of when I was too much of a smartass to be nice. This is for you, Sara, because you always give me a place to go the day before Christmas. And for you, Giorgio, because I'll always be a junior in front of you. This is for you, Grandma Concetta and Grandpa Salvatore, for the sausages, and for you, Grandma Leda, for your being as lively as any of us youngsters.

This is for you, Tiago, because we met only once to date, but enough to learn how much I have to learn from you.

This is also for all those I couldn't mention, including any of my present and future loves, so very complicated that would deserve a book of its own! And this is for me too, to help understand what I want to be.

—Francesco

About the Authors

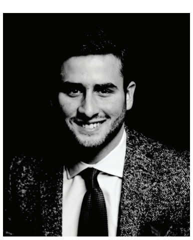
DINO ESPOSITO

If I look back, I count more 20 books authored and 1000+ articles in a 25-year-long career. I've been writing the "Cutting Edge" column for *MSDN Magazine* month after month for 22 consecutive years. It is commonly recognized that such books and articles have helped the professional growth of thousands of .NET and ASP.NET developers and software architects worldwide.

After I escaped a dreadful COBOL project, in 1992 I started as a C developer, and since then, I have witnessed MFC and ATL, COM and DCOM, the debut of .NET, the rise and fall of Silverlight, and the ups and downs of various architectural patterns. In 1995 I led a team of five dreamers who actually deployed things that today we would call Google Photos and Shuttershock—desktop applications capable of dealing with photos stored in a virtual place that nobody had called the cloud yet. Since 2003 I have written Microsoft Press books about ASP.NET and also authored the best-seller *Microsoft .NET: Architecting Applications for the Enterprise*. I have a few successful Pluralsight courses on .NET architecture, ASP.NET MVC UI, and, recently, ML.NET. As architect of most of the backoffice applications that keep the professional tennis world tour running, I've been focusing on renewable energy, IoT, and artificial intelligence for the past two years as the corporate digital strategist at BaxEnergy.

You can get in touch with me through https://youbiquitous.net or twitter.com/despos, or you can connect to my LinkedIn network.

FRANCESCO ESPOSITO



I was 12 or so in the early days of the Windows Phone launch, and I absolutely wanted one of those devices in my hands. I could have asked Dad or Mom to buy it, but I didn't know how they would react. As a normal teenager, I had exactly zero chance of having someone buy it for me. So, I found out I was quite good at making sense of programming languages and impressed some folks at Microsoft enough to have a device to test. A Windows Phone was only the beginning; then came my insane passion for iOS and, later, the shortcuts of C#.

The current part of my life began when I graduated from high school, one year earlier than expected. By the way, only 0.006 percent of students do that in Italy. I felt as powerful as a semi-god and enrolled in mathematics. I failed my first exams, and the shock put me at work day and night on ASP.NET as a self punishment. I founded my small software company, Youbiquitous, and began living on my own money. In 2017, my innate love for mathematics was resurrected and put me back on track with studies and led me to take the plunge in financial investments and machine learning.

This book, then, is the natural consequence of the end of my childhood. I wanted to give something back to my dad and help him make sense of the deep mathematics behind neural networks and algorithms. By the way, I have a dream: developing a supertheory of intelligence that would mathematically explore why the artificial intelligence of today works and where we can go further.

You can get in touch with me at https://youbiquitous.net.

Introduction

We need men who can dream of things that never were, and ask why not.

—John F. Kennedy, Speech to the Irish Parliament, June 1963

There are two views of artificial intelligence that people face today, and they are nonexclusive. One is the view pushed and pursued by the vast majority of media; the other is the view pushed and pursued by the IT community. In both camps, there are some true experts and some true pundits.

The view pushed by media focuses on the impact that artificial intelligence as a whole, in known and yet-to-know forms, may possibly have on our lives in some unfathomable future. The view pushed by the IT community (where software and data science experts belong) presents machine learning as the foundation of a new generation of software services that are just more intelligent than current services.

In the middle ground between the mass of people that the media reach and the much smaller IT community sits the patrol of cloud giants. They're the ones who conduct research and move the state of the art one step further every day, releasing new services for everyone to potentially add intelligence to new and existing applications.

At the base of the artificial intelligence pyramid sit managers and executives. On one hand, they're eager to apply to business those stunning services they hear from the tech news to edge out their competitors. On the other hand, they face the staggering bills of the projects they embarked on with the best of hopes.

- Artificial intelligence is not a magic wand.
- Artificial intelligence is not a service to pay per use. Worse yet, it's neither a capital nor operating expenditure.
- Artificial intelligence is just software.

Any business decision about artificial intelligence is better if made through the lens of software development: set requirements, get a reliable partner, put a budget on the table, work, start again in full respect of agility.

Is it that easy, then?

While artificial intelligence is about software development, it's not exactly the same as building an e-commerce website or a booking platform.

- Don't embark on artificial intelligence projects if you don't have a clear idea of the problem to solve, the context of it, and the point(s) to make.
- Don't embark on ambitious and adventurous projects by following the sole example of your closest competitor.
- Don't embark on such projects if you're not ready to lose some good money.

Just address one pain point at a time, build a cross-functional team, and provide full access to data.

Who Should Read This Book?

In the preparation of this book, we received a lot of feedback about the structure and elaborated on it quite a few times. We radically changed the table of contents at least three times. The hard part is that we devised this book to be unique and innovative, pursuing an idea of machine learning and software development a bit far away from the reality we see. Hopefully, our vision is the vision of machine learning that comes from the near future!

We see machine learning bounded within the fences of data science, as an artifact to be delivered to developers to embed it into some web service or desktop application. This is waterfall—no more no less. Where is all the agile that companies and enterprises constantly talk about? Agile ML means that data scientists and developers work together, and business analysts and domain experts join the team. And data stakeholders—whether it's IT or DevOps or whatever else—also join to facilitate data access and manipulation. This is agile teamwork—no more, no less.

We see the (business) need of a convergence of skills—from data science to software development and from software development to data science. This entry-level book is good for both sides of the pipeline. It talks to developers and shows ML.NET in action (over Python and along with Python) before getting into the analysis of the mechanics of machine learning algorithms. It also talks to data scientists who need to learn more about software needs.

This book is ideal if you're a software developer willing to add data science and machine learning skills to your arsenal. It's also ideal if you're a data scientist willing to learn more about software. Both categories, though, need to learn more and more about the other.

This is the bet of this book. We've classified it as "introductory" because it expands in width instead of going deep. It provides .NET examples because we think that, while the Python ecosystem is rich and thriving, there's no reason not to look around for platforms that allow you to do some machine learning closer to the bare metal of software applications, software services, and microservices—where ultimately any learning pipeline (including TensorFlow, PyTorch, handcrafted Python code) ends up being used.

Who Should Not Read This Book?

This is an introductory-level book specifically devised to give a broad but clear and accurate overview of machine learning using the ML.NET platform for experimenting. If you're looking for tons of Python examples, this book is not ideal. If you're looking for how-to examples to copy and paste in your solutions, whether Python or ML.NET, we're not sure this book is ideal. If you're looking for the nitty-gritty details of the mathematics behind algorithms or for an annotated overview of some implementations of algorithms, again, this book may not be ideal. (We do include some mathematics, but we still only scratch the surface.)

Organization of This Book

This book is divided into five sections. Part I, "Laying the Groundwork of Machine Learning," provides a quick overview of the foundation of artificial intelligence, intelligent software, and the basic steps of any machine learning project within end-to-end solutions. Part II, "Machine Learning in .NET," focuses on the ML.NET library and outlines its core parts, such as tasks for data processing, training, and evaluation in the context of common problems such as regression and classification. Part III, "Fundamentals of Shallow Learning," touches on the mathematical details of families of algorithms commonly trained to solve real-life problems: regressors, decision trees, ensemble methods, Bayesian classifiers, support vector machines, K-means, online gradients. Part IV, "Fundamentals of Deep Learning," is dedicated to neural networks that may come into play when none of the previous algorithms are found suitable. Finally, Part V, "Final Thoughts," is about the business vision of artificial intelligence in general and machine learning in particular, and it provides a cursory review of the runtime services for data processing and computation made available by cloud platforms, specifically the Azure platform.

Code Samples

All the code illustrated in the book, including possible errata and extensions, can be found at *MicrosoftPressStore.com/IntroMachineLearning/downloads*.

PART I

Laying the Groundwork of Machine Learning

CHAPTER 3

Mapping Problems and Algorithms

When I consider what people generally want in calculating, I found that it always is a number.

—Muḥammad ibn Mūsā al-Khwārizmī

Persian mathematician of eighth century whose name originated the word algorithm

More often than not, the user experience produced by machine learning looks like magic to users. At the end of the day, though, machine learning is only a new flavor of software—a new specialty much like web or database development—and a flavor of software that today is a real breakthrough.

A breakthrough technology is any technology that enables people to do things that weren't possible before. Behind the apparent magic of final effects, however, there is a series of cumbersome tasks and, more than everything else, there's a series of sequential and interconnected decisions along the way that are hard to make and time consuming. In a nutshell, they are critical decisions for the success of the solution.

This chapter has two purposes. First, it identifies the classes of problems that machine learning can realistically address and the algorithms known to be appropriate for each class. Second, it introduces a relatively new approach—automated machine learning or AutoML for short—that can automate the selection of the best machine learning pipeline for a given problem and a given dataset.

In this chapter, we'll describe classes of problems and classes of algorithms. We'll focus on the building blocks of a learning pipeline in the next chapter.

Fundamental Problems

As you saw in Chapter 2, "Intelligent Software," the whole area of machine-based learning can be split into supervised and unsupervised learning. It's an abstract partition of the space of algorithms, and the main discriminant for being supervised or unsupervised is whether or not the initial dataset includes valid answers. Put another way, we can reduce automated learning into the union of two learning approaches—*learning by example* (supervised) and *learning by discovery* (unsupervised).

Under these two forms of learning, we can identify a number of general problems and for each a number of general algorithms. This layout is reflected in the organization of any machine learning software development library you can find out there and use—whether it's based on Python, Java, or .NET.

> **Note** Not coincidentally, most of the topics covered in the following chapters match, to a large extent, the tasks of the newest Microsoft's ML.NET framework (covered in Part II, "Machine Learning in .NET") and algorithm cheat-sheet of scikit-learn—an extremely popular machine learning Python library. (See https://scikit-learn.org.)

Classifying Objects

The classification problem is about identifying the category an object belongs to. In this context, an object is a data item and is fully represented by an array of values (known as *features*). Each value refers to a measurable property that makes sense to consider in the scenario under analysis. It is key to note that classification can predict values only in a discrete, categorical set.

Variations of the Problem

The actual rules that govern the object-to-category mapping process lead to slightly different variations of the classification problem and subsequently different implementation tasks.

Binary Classification. The algorithm has to assign the processed object to one of only two possible categories. An example is deciding whether, based on a battery of tests for a particular disease, a patient should be placed in the "disease" or "no-disease" group.

Multiclass Classification. The algorithm has to assign the processed object to one of many possible categories. Each object can be assigned to one and only one category. For example, classifying the competency of a candidate, it can be any of poor/sufficient/good/great but not any two at the same time.

Multilabel Classification. The algorithm is expected to provide an array of categories (or labels) that the object belongs to. An example is how to classify a blog post. It can be about sports, technology, and perhaps politics at the same time.

Anomaly Detection. The algorithm aims to spot objects in the dataset whose property values are significantly different from the values of the majority of other objects. Those anomalies are also often referred to as *outliers*.

Commonly Used Algorithms

At the highest level of abstraction, classification is the process of predicting the group to which a given data item belongs. In stricter math terms, a classification algorithm is a function that maps input variables to discrete output variables. (See Figure 3-1.)

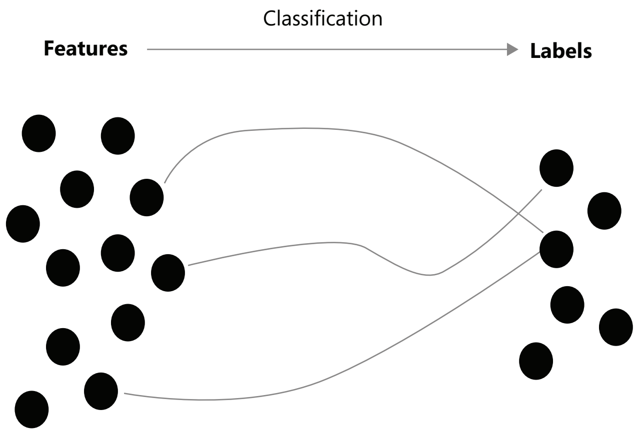


FIGURE 3-1 A graphical representation of a classification function

The classes of algorithms most commonly used for classification problems are as follows:

Decision Tree. A decision tree is a tailor-made binary tree that implements a sequence of rules to be progressively applied to each input object. Each leaf of the tree represents one of the possible output categories. Along the way, the input object is routed downward through the levels of the tree based on rules set at each node. Each rule is based on a possible value of one of the features. In other words, at each step, the key feature value of the input object (say, Age) is checked against the set value (say, 40), and the visit proceeds in the subtree that applies (say, less than or greater than or equal to 40). The number of nodes and the feature/value rules implemented are determined during the training of the algorithm.

Random Forest. This is a more specialized version of the decision tree algorithm. Instead of a single tree, the algorithm uses a forest of simpler trees trained differently and then provides a response that is some average of all the responses obtained.

Support Vector Machine. Conceptually, this algorithm represents the input values as points in an *n*-dimensional space and looks for a sufficiently wide gap between points. In two dimensions, you can imagine the algorithm looking for a curve that cuts the plane in two, leaving as much space as possible along the margin. In three dimensions, you can think of a plane that cuts the space in two.

Naïve Bayes. This algorithm works by computing the probability that a given object, given its values, may fall in one of the predefined categories. The algorithm is based on Bayes' theorem, which describes the likelihood of an event given some related conditions.

Logistic Regression. This algorithm calculates the probability of an object falling in a given category given its properties. The algorithm uses a sigmoid (logistic) function that, for its mathematical nature, lends itself well to be optimized to calculate a probability very close to 1 (or very close to 0). For this reason, the algorithm works well in either/or scenarios, and so it is mostly used in binary classification.

The preceding list is not exhaustive but includes the most-used classes of algorithms battle-tested for classification problems.

> **Important** In the everyday jargon of machine learning, the term *algorithm* commonly refers to an entire family of algorithms that share the same general approach to the solution but may differ on a number of minor and not-so-minor implementation details. If you want to refer to a specific implementation of an algorithm, the term *trainer* (or even the term *estimator*) is more common. The term *pipeline*, instead, refers to the overall combination of data transformations, trainers, and evaluators that form the ultimately deployed machine learning *model*.

Common Problems Addressed by Classification

A number of real-life problems can be modeled as classification problems, whether binary, multiclass, or multilabel. Again, the following list can't and won't be exhaustive, but it is enough to give a clue about where to look when a concrete business issue surfaces:

- Spam and customer churn detection
- Data ranking and sentiment analysis
- Early diagnosis of a disease from medical images
- A recommender system built for customers
- News tagging
- Fraud or fault detection

Spam detection can be seen as a binary classification problem: an email is spam or is not. The same can be said for early diagnosis solutions although in this case the nature of the input data—images instead of records of data—requires a more sophisticated pipeline and probably would be solved using a neural network rather than any of the algorithms described earlier. Customer churn detection and sentiment analysis are multiclass problems, whereas news tagging and recommenders are multilabel problems. Finally, fraud or fault detection can be catalogued as an anomaly detection problem.

Predicting Results

Many would associate artificial intelligence with the ability to make smart predictions about future events. In spite of appearances, prediction is not magic but the result of a few statistical techniques, the most relevant of which is regression analysis. Regression measures the strength of a relationship set between one output variable and a series of input variables.

Regression is a supervised technique and is used to predict a continuous value (as opposed to discrete categorical values of classification).

Variations of the Problem

Regression is about finding a mathematical function that captures the relationship between input and output values. What kind of function? Different formulations of the regression function lead to different variations of the regression problem. Here are some macro areas:

Linear Regression. The algorithm seeks a linear, straight-line function so that all values, present and future, plot around it. The linear regression algorithm is fairly simple and, to a large extent, even unrealistic because, in practice, it means that a single value guides the prediction. Any realistic predictive scenarios, instead, bring in several different input data flows.

Multilinear Regression. In this case, the regression function responsible for the actual prediction is based on a larger number of input parameters. This fits in a much smoother way into the real world because to predict the price of a house, for example, you would use not only square footage but also historical trends, neighborhood, rooms, age, and maybe more factors.

Polynomial Regression. The relationship between the input values and the predicted value is modeled as an *n*th degree polynomial in one of the input values. In this regard, polynomial regression is a special case of multilinear regression and is useful when data scientists have reasons to hypothesize a curvilinear relationship.

Nonlinear Regression. Any techniques that need a nonlinear curve to describe the trend of the output value given a set of input data fall under the umbrella of nonlinear regression.

Commonly Used Algorithms

The solution to a regression problem is finding the curve that best follows the trend of input data. Needless to say, the training phase of the algorithm works on training data, but the deployed model, instead, needs to perform well on similar live data. The curve that predicts the output value based on the input is the curve that minimizes a given error function. The various algorithms define the error function in different ways and measure the error in different ways.

The classes of algorithms most commonly used for regression problems are as follows:

Gradient Descent. The gradient descent algorithm is expected to return the coefficients that minimize an error function. It works iteratively by first assigning default values to the coefficient and then measuring the error. If the error is large, it then looks at the gradient of the function and moves ahead in that direction, determining new values for the coefficients. It repeats the step until some stop condition is met.

Stochastic Dual Coordinate Ascent. This algorithm takes a different approach and essentially solves a dual problem—maximizing the value calculated by the function rather than minimizing the error. It doesn't use the gradient but proceeds along each axis until it finds a maximum and then moves to the next axis.

Regression Decision Tree. This algorithm builds a decision tree, as discussed previously, for classification problems. The main differences are the type of the error function used to decide if the tree is deep enough and the way in which the feature value in each node is chosen (in this case, it is the mean of all values).

Gradient Boosting Machine. This algorithm combines multiple weaker algorithms (e.g., most commonly, a basic decision tree) and builds a unified, stronger learner. Typically, the prediction results from the weighed combination of the output of all the chained weak learners. Extremely popular algorithms in this class are XGBoost and LightGBM.

> **Important** Both regression and classification cover very large areas of real-life problems. And often the actual problems faced can't be solved with any of these algorithms. Instead, they require a deeper learning approach via some neural network.

Common Problems Addressed by Regression

Regression is the task of predicting a continuous value, whether a quantity, a price, or a temperature.

- Price prediction (houses, stocks, taxi fares, energy)
- Production prediction (food, goods, energy, availability of water)
- Income prediction
- Time series forecasting

Time series regression is interesting because it can help understand and, better yet, predict the behavior of sophisticated dynamic systems that periodically report their status. This is fairly common in industrial plants where, even thanks to Internet of Things (IoT) devices, there's plenty of observational data. Time series regression is also commonly used in the forecasts of financial, industrial, and medical systems.

Grouping Objects

In machine learning, clustering refers to the grouping of objects represented as a set of input values. A clustering algorithm will place each object point into a specific group based on the assumption that objects in the same group have similar properties and objects in different groups have quite dissimilar properties.

At first, clustering may look like classification, and in fact, both problems are about deciding the category that a given data item belongs to. There's one key difference between the two, however. A clustering algorithm receives no guidance from the training dataset about the possible target groups. In other words, clustering is a form of unsupervised learning, and the algorithm is left alone to figure out how many groups the available dataset can be split on.

A clustering algorithm processes a dataset and returns an array of subsets. Those subsets receive no labels and no clues about the content from the algorithm itself. Any further analysis is left to the data science team. (See Figure 3-2.)

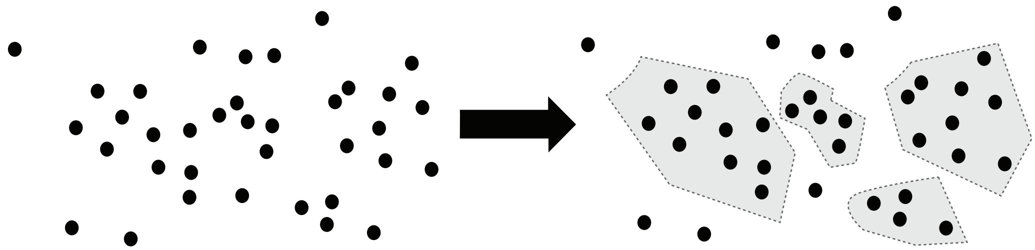


FIGURE 3-2 The final outcome of a clustering algorithm run on a dataset

Commonly Used Algorithms

The essence of clustering is analyzing data and identifying as many relevant clusters of data as it can find. While the idea of a cluster is fairly intuitive—a group of correlated data items—it still needs some formal definition of the concept of correlation to be concretely applied. In the end, a clustering algorithm looks for disjoint areas (not necessarily partitions) of the data space that contain data items with some sort of similarity.

This fact leads straight to another noticeable difference between clustering and regression or classification. You'll never deploy a clustering model in production and never run it on live data to get a label or a prediction. Instead, you may use the clustering step to make sense of the available data and plan some further supervised learning pipeline.

Clustering algorithms adopt one of the following approaches: partition-based, density-based, or hierarchy-based. Here are the most popular algorithms:

K-Means. This partition-based algorithm sets a fixed number of clusters (according to some preliminary data analysis) and randomly defines their data center. Next, it goes through the entire dataset and calculates the distance between each point and each of the data centers. The point finds its place in the cluster whose center is the nearest. The algorithm proceeds iteratively and recalculates the data center at each step.

Mean-Shift. This partition-based algorithm defines a circular sliding window (with arbitrary radius) and initially centers it at a random point. At each step, the algorithm shifts the center point of the window to the mean of the points within the radius. The method converges when no better center point is found. The process is repeated until all points fall in a window and overlapping windows are resolved, keeping only the window with the most points.

DBSCAN. This density-based algorithm starts from the first unvisited point in the dataset and includes all points located within a given range in a new cluster. If too few points are found, the point is marked as an outlier for the current iteration. Otherwise, all points within a given range of each point currently in the cluster are recursively added to the cluster. Iterations continue until there's at least one point not included in any cluster or their number is so small that it's OK to ignore them.

Agglomerative Hierarchical Clustering. This hierarchy-based algorithm initially treats each point as a cluster and proceeds iteratively, combining clusters that are close enough to a given distance metric. Technically, the algorithm would end when all the points fit in a single cluster, which would be the same as the original dataset. Needless to say, you can set a maximum number of iterations or use any other logic to decide when to stop merging clusters.

K-Means is by far the simplest and fastest algorithm, but, in some way, it violates the core principle of clustering because it sets a fixed number of groups. So, in the end, it's halfway between classification and clustering. In general, clustering algorithms have a linear complexity, with the notable exception of hierarchy-based methods. Not all algorithms, however, produce the same quality of clustering regardless of the distribution of the dataset. DBSCAN, for example, doesn't perform as well as others when the clusters are of varying density, but it's more efficient than, say, partition-based methods in the detection of outliers.

Common Problems Addressed by Clustering

Clustering is the method of many crucial business tasks in a number of different fields, including marketing, biology, insurance, and in general wherever screening of population, habits, numbers, media content, or text is relevant.

- Tagging digital content (videos, music, images, blog posts)
- Regrouping books and news based on author, topics, and other valuable information
- Discovering customer segments for marketing purposes
- Identifying suspicious fraudulent finance or insurance operations
- Performing geographical analysis for city planning or energy power plant planning

It is remarkable to consider that clustering solutions are often used in combination with a classification system. Clustering may be first used to find a reasonable number of categories for the data expected in production, and then a classification method could be employed on the identified clusters. In this case, categories will be manually labeled, looking at the content of identified clusters. In addition, the clustering method might be periodically rerun on a larger and updated dataset to see whether a better categorization of the content is possible.

More Complex Problems

Classification, regression, and clustering algorithms are sometimes referred to as *shallow learning*, in contrast to *deep learning*. Admittedly, the distinction between shallow learning and deep learning is a bit sketchy and cursory; yet, it marks the point of separating problems that can be solved with a relatively straight algorithm from those that require the introduction of some flavor of neural networks (more or less deep in terms of constituent layers) or the pipelining of multiple straight algorithms. Typically, these problems revolve around the area of cognition such as computer vision, creative work, and speech synthesis.

Image Classification

Image processing began in the late 1960s when a group of NASA scientists had the problem of converting analogic signals to digital images. The core of image processing is the simple application of mathematical functions to a matrix of pixels. A much more enhanced form of image processing is computer vision.

Computer vision isn't limited to processing data points but attempts to recognize patterns of pixels and how they match to forms (objects, animals, persons) in the real world. Computer vision is the branch of machine learning devoted to the emulation of the human eye, capable of capturing images and recognizing and classifying them based on properties such as size, color, and luminosity.

In the realm of computer vision, image classification is one of the most interesting sectors, especially for its applications to sensitive fields such as health care and security. Image classification is the process of taking a picture (or a video frame), analyzing it, and producing a response in the form of a categorical value (it's a dog) or a set of probabilistic values (70 percent, it's a dog; 20 percent, it's a wolf; 10 percent, it's a fox). In much the same way, an image classifier can guess mood, attitude, or even pain.

Even though many existing cloud services can recognize and classify images (even video frames), the problem of image classification can hardly be tackled outside a specific business context. In other words, you can hardly take a generic public cloud cognitive service and use it to process medical images (of a certain type) or monitor the live stream of a public camera. You need specific training for the algorithm tailor-made for the scenario you're facing.

An image classifier is typically a convolutional multilayer neural network. In such a software environment, each processing node receives input from the previous layers and passes processed data to the next. Depending on the number (and type) of layers, the resulting algorithm proves able (or not so able) to do certain things.

Object Detection

A side aspect of computer vision, tightly related to image classification, is object detection. With image classification, you can rely on a class of algorithms capable of looking at live streams of pictures and recognize elements in it. In other words, image classification can tell you what is in the processed picture. Object detection goes one step further and operates a sort of multiclass classification of the picture, telling about all the forms recognized and also about their relative position.

Object detection is very hot in technologies like self-driving cars and robotics. Advanced forms of object detection can also identify bounding boxes for the form to find and even draw precise boundaries around it. Object detection algorithms typically belong to either of two classes—classification-based or regression-based.

In this context, classification and regression don't refer to the straight shallow learning algorithms covered earlier in the chapter but relate to the learning approach taken by the neural network to come to a conclusion.

Text Analytics

Text analytics consists of parsing and tokenizing text, looking for patterns and trends. It is about learning relationships between named entities, performing lexical analysis, calculating and evaluating the frequency of words, and identifying sentence boundaries and lemmas. In a way, it's a statistical exercise of data mining and predictive analysis applied to text with the ultimate goal of taking software to interact with humans using the same natural language.

A typical application of text analytics is summarizing, indexing, and tagging the content of large digital free text databases and documents such as the comments (and complaints) left by customers of a public service. Text analytics often goes under the more expressive name of *natural language processing* (NLP) and is currently explored in more ambitious scenarios such as processing a live stream, performing speech recognition, and using recognized text for further parsing and information retrieval. Natural language processing applications are commonly built on top of neural networks in which the input text passes through multiple layers to be progressively parsed and tokenized until the networks produce a set of probabilistic intents.

There are quite a few applications of NLP available in the industry, buried in the folds of enterprise frameworks used in answering machine applications and call centers. However, if you just want to explore the power of the raw NLP, research a few of the existing test platforms, such as https://knowledge-studio-demo.ng.bluemix.net. The tool parses text, an excerpt of a police car accident report, and automatically extracts relevant facts, such as age of the involved people, characteristics of involved vehicles, location, and time.

Automated Machine Learning

Machine learning is a large field and is growing larger every day. As you'll see in much more detail in the next chapter, building an intelligent solution for a real-life business problem requires a *workflow* that essentially consists of a combination of different steps: data transformations, training algorithms, evaluation metrics, and, last but not least, domain knowledge, knowledge base, trial-and-error attitude, and imagination.

In this context, while the human ability to sort things out probably remains unparalleled, the community is seriously investigating the possibility of using automated, wizard-style tools to prepare a sketchy plan that could possibly represent the foundation of a true solution in a matter of minutes instead of days.

This is just the essence of the automated machine learning (AutoML) approach and consists of a framework that looks at your data and declared intent and intelligently suggests the steps to take that it determines most appropriate.

Aspects of an AutoML Platform

The typical end-to-end pipeline of any machine learning solution applied to a real-world problem most likely includes a number of steps, as outlined here:

- Preliminary analysis and cleaning of available data
- Identification of the properties (features) of the data that look most promising and relevant to solve the actual problem

- Selection of the algorithm
- Configuration of the parameters of the algorithm
- Definition of an appropriate validation model to measure the performance of the algorithm and indirectly the quality of the data it is set to use

Machine learning may not be for the faint-hearted, and even when one has a strong domain knowledge, the risk of feeling like a nonexpert newbie is fairly high.

Hence, AutoML is emerging as a solution to get people started quickly on machine learning projects and sometimes even effectively. AutoML offers the clear advantage of being fast and producing working solutions. The debatable point is not how objectively good the solution is that you can get out of an AutoML wizard, but the trade-off between what you get from AutoML and what you might be able to design by hand, especially if your team is not made up of domain and machine learning super-experts.

Note To some extent, the debate about the alleged superficiality of AutoML solutions recalls past debates about the use of high-level programming languages over Assembly and the use of system-managed memory over memory cells directly allocated by the programmer. Our frank opinion is that AutoML frameworks are excellent at doing their job on simple problems. They can't do much for complex problems, however. But unfortunately, as of today, most real-world problems are quite complex.

Common Features

An AutoML framework is made of two distinct parts: a public list of supported learning scenarios and an invisible runtime service that returns a deliverable model based on some input parameters. A learning scenario is essentially an expert subsystem designed to solve specific classes of problems using data in one of a few predefined formats. The runtime is a learning pipeline in which a set of predefined data transformations are performed on selected input given the learning objective; target features are selected; and the trainer is selected, configured, trained, and tested.

An AutoML framework will perform any of the following tasks in an automated way after the user has indicated the physical source of data (tabular files, relational databases, cloud-based data warehouses) and the learning objective:

- Preprocessing and loading of data from different formats including detection of missing and skewed values
- Understanding of the type of each dataset column to figure out whether the column is, say, a Boolean, a discrete number, a categorical value, or free text
- Application of built-in forms of feature engineering and selection, namely the addition or transformation of data columns in a way that makes particular sense for the learning objective
- Detection of the type of work required by the learning objective (binary classification, regression, anomaly detection) and selection of a range of most appropriate training algorithms

- Configuration of the hyperparameters of the selected training algorithms
- Training of the model, application of appropriate evaluation metrics, and testing of the model

In addition, an AutoML framework is also often capable of visualizing data and results in a fancy way that is also helpful to better understand the underpinnings of the problem at hand.

There are a couple of popular AutoML frameworks: one is from Google and one, the newest, from Microsoft. Let's first briefly examine the Google Cloud AutoML platform, and then we'll go for a deeper live demonstration of the Microsoft AutoML framework as integrated in Visual Studio 2019.

Google Cloud AutoML

The Google Cloud AutoML platform is located at https://cloud.google.com/automl. It comes as a suite of machine learning systems specifically designed to simplify as much as possible the building of models tailor-made for specific needs. The platform works much like a UI wizard and guides the user through the steps of selecting the scenario, data, and parameters and then does the apparent magic of returning a deployable artifact out of nowhere. Internally, the Google Cloud AutoML platform relies on Google's transfer learning technology, which allows the building of neural networks as the composition of predefined existing networks.

Google Cloud AutoML supports a few learning scenarios such as computer vision, object detection in videos and still images, and natural language processing and translation. As you can see, it's a group of pretty advanced and sophisticated scenarios. It also supports a simpler one, called AutoML Tables, that works on tabular datasets and tests multiple model types at the same time (regression, feedforward neural network, decision tree, ensemble methods).

Microsoft AutoML Model Builder

An AutoML framework is also integrated in Visual Studio 2019 and comes packaged with ML.NET—the newest Microsoft .NET-based library for machine learning. The AutoML Model Builder framework has both a visual, wizard-style interface in Visual Studio (more on this in a moment) and a command-line interface (CLI) for use from within command-based environments such as PowerShell. A quick but effective summary of AutoML CLI can be found at https://bit.ly/2FaK7SP.

In Microsoft's AutoML framework, developers choose a task, provide the data source, and indicate a maximum training duration. Needless to say, the selected maximum duration is a discriminant for the quality of the final model. The shorter time you choose, the less reliable the final model can be.

> **Note** Compared to Google Cloud AutoML, the Microsoft AutoML solution currently focuses on simpler tasks and is available also on premise and then for shorter training cycles. The Google platform, instead, is cloud-based and suitable for longer and more realistic training cycles available through a paid subscription.

The AutoML Model Builder in Action

In Visual Studio 2019, after you install the latest version of the ML.NET Model Builder extension, you gain the ability to add a machine learning item to an existing project. When you do that, you're sent to a wizard like the one shown in Figure 3-3.

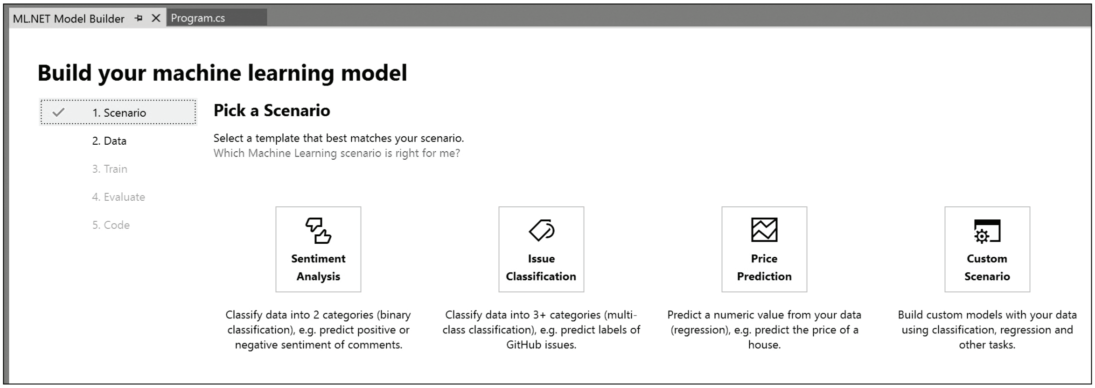


FIGURE 3-3 The main page of the Model Builder Visual Studio extension

As you can see, the wizard is articulated in five steps that broadly match the main steps of any machine learning pipeline. The first step of the builder is choosing the learning scenario—namely, the broad category of the problem for which you'd like to build a machine learning solution. In the version of the builder used for the test, the choice is not very large: Sentiment Analysis, Issue Classification, Price Prediction, and Custom Scenario. As an example, let's go for Price Prediction.

Exploring the Price Prediction Scenario

After you pick the scenario, the wizard asks you to load some data into the system. For the price prediction scenario, you can choose from a plain file or a SQL Server table. In the example shown in Figure 3-4, the loaded file is a CSV file. One key input to provide is the name of the column you want the final model to predict. In this case, the CSV file contains about one million rows, representing a taxi ride that really took place. The column to predict is the fare amount.

Training the Model

The third step is about the selection of the ideal trainer—the algorithm that is the most appropriate for the learning scenario and the data. This is where the power (and from a certain angle also the weakness) of the automated machine learning framework emerges. Some hard-coded logic, specific to the chosen scenario, tries a few training algorithms based on the allotted training time. Figure 3-5 shows an estimation of the training time necessary for a certain amount of data.

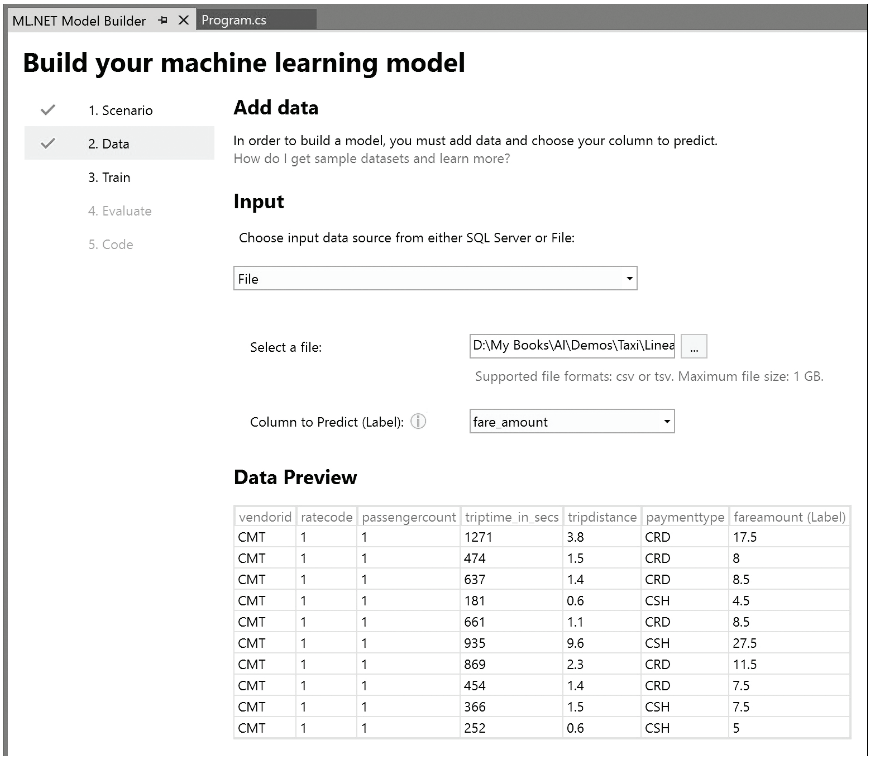


FIGURE 3-4 Loading data into the model

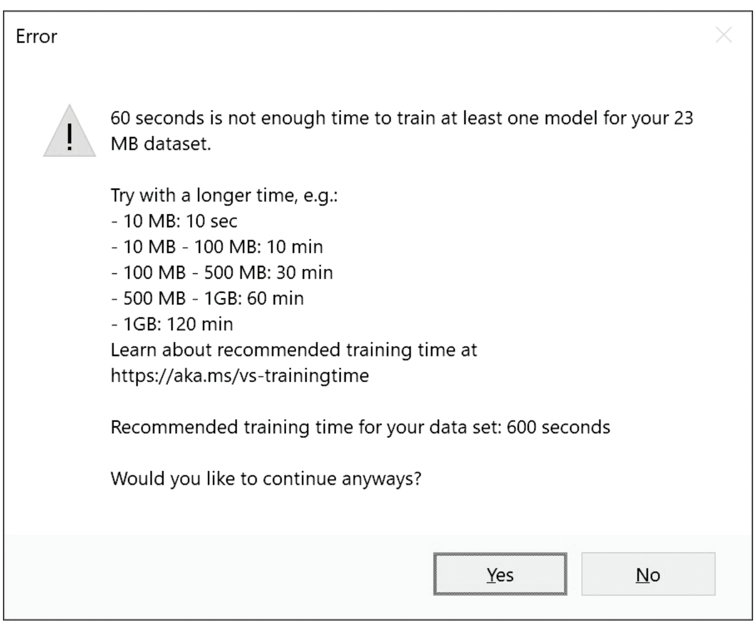


FIGURE 3-5 Estimating the training time

During the training phase, the system tries several different algorithms and uses an apt metric to evaluate its performance. (See Figure 3-6.)

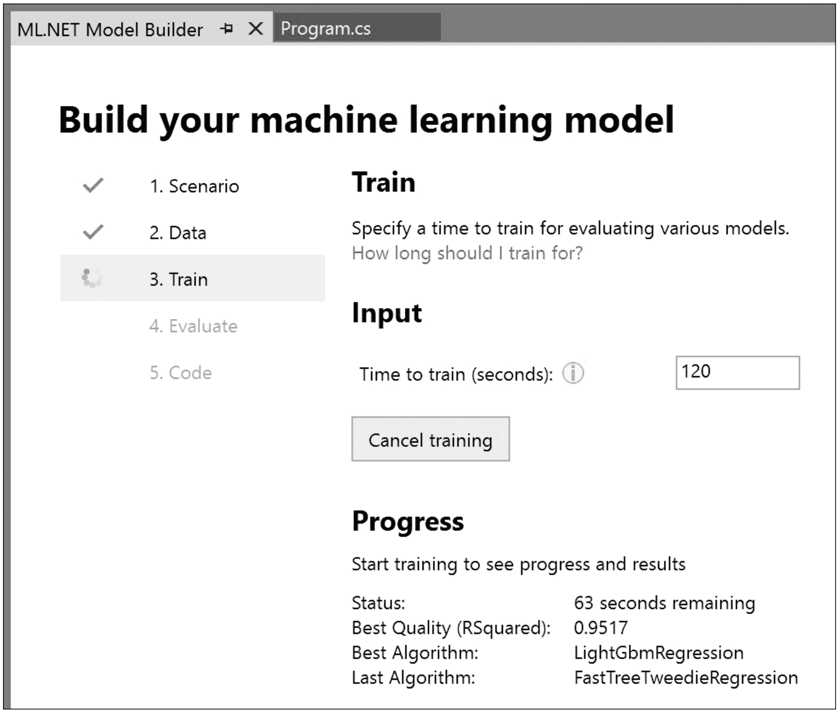


FIGURE 3-6 AutoML tries different algorithms and uses some metrics to evaluate the quality.

Evaluating the Results

At the end of the training, the AutoML system has data about a few algorithms it has tried with different hyperparameters. The metrics for evaluating the performance depend on the tasks and the algorithm. Price prediction is essentially a regression task for which the R-squared measure is the most commonly used. (We'll cover the math behind regression and R-squared in Chapter 11, "How to Make Simple Predictions: Linear Regression.") The theoretic ideal value of the R-squared metrics is 1; therefore, any value close enough to 1 is more than acceptable. Consider that in training, a resulting metric with a value of 1 (or very close to 1) is often the sign of overfitting—the model fits too much to the training data and potentially might not work effectively on live data once in production.

The AutoML process then suggests the use of the *LightGbmRegression* algorithm. If you want, you can just take the ZIP file with the final model ready for deployment. But what about looking into the actual set of data transformation and the actual code to possibly modify for further improvements?

The AutoML also offers the option to add the C# files to the current project for you to further edit them and retrain the model on a different dataset, for example. (See Figure 3-7.)

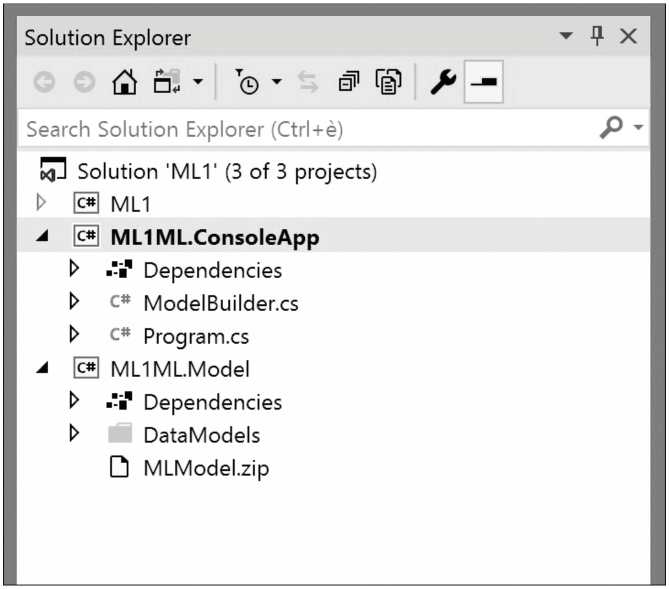


FIGURE 3-7 Autogenerated projects added by the Model Builder

As you can see, the figure contains two projects. One is a console application that contains a *ModelBuilder.cs* file packed with the code used to build the model. The other project is a class library and contains a sample client application seen as the foundation for using the model. This project also contains the actual model as a ZIP file.

Summary

Machine learning is ultimately intelligent software, but it is not the magic wand that movies and literature (and recently also sales/marketing departments) love to depict. More importantly, machine learning is not a physical black box you can pick from the shelves of a drugstore, bring home, mount, and use.

In the real world, you can't just "load data into the machine" and have the machine, in some way, just use it. In the real world, there are a few classes of approaches (mostly derived from statistics) such as regression, classification, and clustering and a bunch of concrete training algorithms. However, when to use which?

Determining which to use is a matter of experience and know-how, but it is also a matter of knowing data and how things actually work in the actual business domain. Does that mean that only experts can do machine learning? Yes, for the most part, that is just the point. However, nobody is born an expert, and everyone needs to get started in some way. This is the reason why automated tools for machine learning are emerging. In this chapter, we briefly looked at the Google Cloud AutoML and Visual Studio ML.NET Model Builder.

With the next chapter, we complete the preliminary path of machine learning, discussing the concept of a pipeline—namely, the sequence of steps that ultimately lead to the production of a deliverable model.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T