Essential Computational Thinking

Computer Science from Scratch

Ricky J. Sethi
Essential Computational Thinking

Computer Science from Scratch

Ricky J. Sethi
For Rohan, Megan, Surinder, Harindar, and especially the team of the two Harindars.
Acknowledgments

I would like to thank my current and former students for the feedback and help in crafting a clearer, more concise book than my meandering initial drafts. In particular, thanks to Kabir Chug and Bryce “Texas” Shurts for assisting with the reviews and administration of the text. I’d also like to acknowledge my Fitchburg State University colleagues for their feedback and encouragement, and particularly Kevin B. Austin, Natasha Kurtonina, Nadimpalli V.R. Mahadev, and John Russo for going the extra mile.

I’m grateful to all of the reviewers, especially Dmitry Zinoviev of Suffolk University, Frank J. Kachurak of Pennsylvania State University, and Siva Jasthi of Metropolitan State University, who so generously donated their time and expertise to help produce a much better and more correct book with their detailed and insightful feedback.

This book depends on quite a few open source resources and I’d like to acknowledge some that were indispensable to its development like LaTeXTemplates.com, www.pexels.com, www.pixabay.com, venngage.com, htmltidy.net, repl.it, and generatedata.com.

And last, but certainly not least, I’d like to thank my family, especially my mother for the constant encouragement and positive reinforcement; my father, for reading endless drafts and commenting thoughtfully and proudly on each one; my son, for letting me bounce ideas off him and making life a joy, in general; and my beautiful wife, for being a constant support and always helping edit drafts in a topic that’s not only outside her area of expertise but also includes occasional sentences full of equations, the bane of her existence.

I should add that any errors which remain, and I’m sure there will be quite a few, are my sole responsibility. The book resource page, available at http://research.sethi.org/ricky/book/, will also contain the reams of errata as they’re discovered.

Ricky J. Sethi
http://research.sethi.org/ricky/
January 2020
Preface

Why a book on CS0 from scratch?

Most of us write the books that we would have wanted to read and this is no different. As such, it follows a normal CS0 breadth-first approach but lays particular emphasis on computational thinking and related topics in data science.

My hope is this book will help build a theoretical and practical foundation for learning computer science from the ground up. It’s not quite from first principles as this is not a rigorous text. But, following William of Occam’s Razor, it is placed on a firm mathematical footing that starts with the fewest assumptions.

And so we delve into defining elementary ideas like data and information; along the way, we quantify these ideas and eke out some of their links to fundamental physics in an effort to show the connection between computer science and the universe itself. In fact, all the laws of physics can be represented as computable functions, and we can, in a very real sense, think of computer science as the most basic representation of the physical universe.

The eminent physicist John A. Wheeler went so far as to say, “...the universe is made of information; matter and energy are only incidental.” I view the role of this book, at its theoretical core, to be to help explore, elucidate, and communicate this deep and perhaps surprising relationship between physics, information, and computation.

Target Audience

This book would be most appropriate for highly-motivated undergraduates who have a significant technical bent and a genuine interest in computer science. It is also appropriate for non-CS graduate students or non-CS professionals who are interested in learning more about computer science and programming for either professional or personal development.

So what kind of a background do you need to appreciate these various connections and learn some of the more advanced material? Although you should be quantitatively inclined, you don’t need any specific background in mathematics. Whatever math we need, we’ll derive or explain along the way. Things might look heinous at times but all the tools you need should be in here.
Book Organization

Computer science is a diverse field and, in Part 1, we explore some of its theoretical bases in a connected but wide-ranging manner with an emphasis on breadth, not depth. Some chapters might still pack quite a punch but, if students are interested in computer science, they should get an overview of the field in large measure, not necessarily in its entirety. Not only is computer science an enormously broad area but this book, to paraphrase Thoreau, is limited by the narrowness of my experience and vision to those areas that caught my eye.

For that matter, this book is not a comprehensive introduction to a particular programming language, either. By necessity, we are constrained to only meeting those parts of Python or Java that support our understanding of fundamental computational ideas. In general, computer scientists tend to be a language agnostic bunch who pick the most appropriate language for a particular problem and that’s the approach in this book, as well. But we should still get a sense of the large variety of topics that underly our study of computer science, from the abstract mathematical bases to the physical underpinnings.

Once the theoretical foundation is laid in Part 1, we shift gears to a purely pragmatic exploration of computing principles. So, in Part 2, we learn the basics of computation and how to use our greatest tool: programming. These chapters should be short and approachable. Students will hopefully find them digestible in brief sittings so they can immediately apply the ideas they’ve learned to start writing substantial programs, or computational solutions. Computer science is an inherently empirical science and few sciences can offer this unique opportunity to let students actually implement and execute theoretical ideas they have developed.

In Part 3, we’ll explore some relatively sophisticated computational ideas. We’ll both meet powerful programming concepts as well as investigate the limits of computational problem solving. We’ll also increase the tools in our toolkit by learning about object-oriented programming, machine learning, data science, and some of the underlying principles of software engineering used in industry. In addition, online supplementary chapters will address topics such as P vs. NP, Big O notation, GUI development, etc.

How to use this book

Depending upon student or instructor interests, Part 1 can be considered completely optional and skipped entirely. The most important aspects of Part 1 are summarized in Ch. 3 and, if you don’t want to emphasize the theoretical underpinnings, you can safely choose to start directly with Part 2.

Similarly, in Part 3, you can pick and choose any chapters that interest you as each chapter in Part 3 is independent of the other chapters. For a typical semester-length course, I spend about 1-2 weeks on Part 1, cover all of Part 2, and pick topics of interest from Part 3 depending on the particular course.

You can also access supplementary information for the text at http://research.sethi.org/ricky/book/, including complementary videos and auto-graded problem sets.

Nota Bene: This book uses standard bibliographical notation as is common in computer science literature so references are indicated by a number in brackets, like [XX]. It also uses bold and italics to define phrases and emphasize concepts inline. Finally, it contains call out boxes like IDEA boxes, NOTE boxes, etc., which are important points that should be highlighted separately or extend a concept without breaking the main narrative.1

Okay, that’s enough build-up... strap yourself in and let’s get started on our adventure!

1The same applies to footnotes, of course.
Contents

Theory: What Is Computer Science?

0 On the Road to Computation ................................................. 3
  0.1 What Is Knowledge? ....................................................... 3
    0.1.1 Step 1: Gather the Facts ......................................... 3
    0.1.2 Step 2: Contextualize the Data ................................... 4
    0.1.3 Step 3: Take Some Action! ....................................... 5
  0.2 Declarative vs Imperative Knowledge ............................. 6
  0.3 The Key to Science ...................................................... 7
  0.4 Computer Science: The Study of Computation .................. 8
  0.5 A Review of Functions .................................................. 8
    0.5.1 Graphical Representation ........................................ 9
    0.5.2 Relational Representation ....................................... 10
    0.5.3 Functional Representation ...................................... 11
  0.6 Computable Functions .................................................. 12
    0.6.1 Computation and Algorithms ................................... 13
  0.7 Talking in Tongues: Programming Languages .................... 14
    0.7.1 Going to Church-Turing ......................................... 16

1 Computational Thinking and Information Theory ................. 21
  1.1 What Is Thinking? ....................................................... 21
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2</td>
<td>Deductive vs Inductive Thinking</td>
<td>22</td>
</tr>
<tr>
<td>1.3</td>
<td>Thinking About Probabilities</td>
<td>23</td>
</tr>
<tr>
<td>1.3.1</td>
<td>Calculating the Probabilities</td>
<td>24</td>
</tr>
<tr>
<td>1.3.2</td>
<td>Samples and Populations in Statistics</td>
<td>24</td>
</tr>
<tr>
<td>1.3.3</td>
<td>Conditional Probabilities and Sample Spaces</td>
<td>25</td>
</tr>
<tr>
<td>1.4</td>
<td>Logical Thinking</td>
<td>25</td>
</tr>
<tr>
<td>1.4.1</td>
<td>Origins of Formal Logic</td>
<td>25</td>
</tr>
<tr>
<td>1.4.2</td>
<td>Propositional Logic and Argumentation</td>
<td>26</td>
</tr>
<tr>
<td>1.4.3</td>
<td>Declarative vs Imperative Statements</td>
<td>28</td>
</tr>
<tr>
<td>1.5</td>
<td>Computational Thinking and Computational Solutions</td>
<td>29</td>
</tr>
<tr>
<td>1.5.1</td>
<td>Computational Thinking Overview</td>
<td>30</td>
</tr>
<tr>
<td>1.6</td>
<td>Two Fundamental Models of Programming</td>
<td>32</td>
</tr>
<tr>
<td>1.6.1</td>
<td>Declarative vs Imperative Programming Languages</td>
<td>33</td>
</tr>
<tr>
<td>1.7</td>
<td>Pseudocode</td>
<td>34</td>
</tr>
<tr>
<td>1.7.1</td>
<td>Pseudocode Expanded: Now with Twice the Pseudol</td>
<td>35</td>
</tr>
<tr>
<td>1.8</td>
<td>Functional and Imperative Models of Computation</td>
<td>37</td>
</tr>
<tr>
<td>1.8.1</td>
<td>Computation in General</td>
<td>38</td>
</tr>
<tr>
<td>1.9</td>
<td>Information Theory</td>
<td>38</td>
</tr>
<tr>
<td>1.9.1</td>
<td>The Birth of Information Theory</td>
<td>39</td>
</tr>
<tr>
<td>1.9.2</td>
<td>First Rule of Communication Club: There Is No Meaning</td>
<td>39</td>
</tr>
<tr>
<td>1.9.3</td>
<td>Shannon Information</td>
<td>42</td>
</tr>
<tr>
<td>1.9.4</td>
<td>Information, Uncertainty, and... Surprise!</td>
<td>44</td>
</tr>
<tr>
<td>1.9.5</td>
<td>Data → Information → Knowledge Revisited</td>
<td>45</td>
</tr>
<tr>
<td>1.10</td>
<td>Shannon’s Information Entropy</td>
<td>46</td>
</tr>
<tr>
<td>1.10.1</td>
<td>Entropy in Physics</td>
<td>47</td>
</tr>
<tr>
<td>1.10.2</td>
<td>Information Entropy and Physical Entropy</td>
<td>50</td>
</tr>
<tr>
<td>1.10.3</td>
<td>Demons Arise!</td>
<td>52</td>
</tr>
<tr>
<td>1.10.4</td>
<td>Connecting Information Entropy and Thermodynamic Entropy</td>
<td>54</td>
</tr>
<tr>
<td>1.10.5</td>
<td>The Universe Itself Is Informational</td>
<td>56</td>
</tr>
<tr>
<td>2</td>
<td>Computational Problem Solving</td>
<td>59</td>
</tr>
<tr>
<td>2.1</td>
<td>What Is a Model?</td>
<td>59</td>
</tr>
<tr>
<td>2.1.1</td>
<td>Models and Abstractions</td>
<td>60</td>
</tr>
<tr>
<td>2.2</td>
<td>Data Representations</td>
<td>62</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Data Structures</td>
<td>63</td>
</tr>
<tr>
<td>2.3</td>
<td>Number Representations</td>
<td>64</td>
</tr>
<tr>
<td>2.3.1</td>
<td>Positional Number Representations</td>
<td>64</td>
</tr>
<tr>
<td>2.3.2</td>
<td>Computational Representations</td>
<td>65</td>
</tr>
<tr>
<td>2.3.3</td>
<td>Physical Representations</td>
<td>65</td>
</tr>
<tr>
<td>2.4</td>
<td>Digital Representations</td>
<td>67</td>
</tr>
<tr>
<td>2.5</td>
<td>Boolean Algebra</td>
<td>68</td>
</tr>
<tr>
<td>2.5.1</td>
<td>Boolean Algebra and Digital Circuits</td>
<td>69</td>
</tr>
</tbody>
</table>
2.5.2 Digital Logic Circuits ........................................ 71
2.5.3 Theory of Computation ......................................... 71
2.6 What Is Information Processing? ............................. 72
2.7 What Is Computer Information Systems (CIS)? .......... 73
  2.7.1 What Is Software Engineering? .......................... 74
2.8 Programming Languages ........................................ 74
  2.8.1 Programming Language Generations ..................... 76
2.9 Computational Thinking Defined ............................. 76
  2.9.1 Computational Thinking Skills ............................ 78
  2.9.2 Algorithmic Expression: Computational Problem Solving ......................................................... 78
  2.9.3 Strategies for Computational Problem Solving .......... 80
2.10 Problem Space and System State ........................... 81
  2.10.1 Turing Machine Example ................................. 83
2.11 Computational Thinking in Action ........................... 85

II Basics: Algorithmic Expression

3 Computational Thinking and Structured Programming .......... 91
  3.1 Review of Computation ........................................ 91
    3.1.1 Modern Digital Computers .............................. 91
    3.1.2 Evolution of Computers .................................. 92
    3.1.3 What is a Computer Program? ........................... 95
  3.2 Computational Thinking Basics ............................... 96
  3.3 Minimal Instruction Set ....................................... 96
  3.4 Getting Started with Python .................................. 98
    3.4.1 What is Python and Where Do I Get It? ............... 98
    3.4.2 "Hello World" in Python ................................ 99
    3.4.3 Error, Error... Does Not Compute! .................... 100
  3.5 Syntax, Semantic, or Logic Errors .......................... 101
    3.5.1 Debugging ................................................. 102
  3.6 State of a Computational System ............................ 103
  3.7 Natural vs Formal Languages ................................ 104
  3.8 Translating Your Programs .................................. 105
  3.9 Playing with Python ......................................... 107
    3.9.1 Python and Mathematics ................................ 108
  3.10 An Example Using Computational Thinking ................. 109

4 Data Types and Variables ......................................... 111
  4.1 Different Types of Data ...................................... 111
  4.2 Data Type = Values + Operations ............................ 112
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3</td>
<td>Variables and Expressions</td>
<td>114</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Expressions and Statements</td>
<td>118</td>
</tr>
<tr>
<td>4.4</td>
<td>Input/Output</td>
<td>118</td>
</tr>
<tr>
<td>4.5</td>
<td>An Example Using Computational Thinking</td>
<td>120</td>
</tr>
<tr>
<td>5</td>
<td>Control Structures</td>
<td>123</td>
</tr>
<tr>
<td>5.1</td>
<td>Algorithms and Control Structures</td>
<td>123</td>
</tr>
<tr>
<td>5.2</td>
<td>Sequence</td>
<td>123</td>
</tr>
<tr>
<td>5.3</td>
<td>Selection</td>
<td>124</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Chained and Nested Conditionals</td>
<td>126</td>
</tr>
<tr>
<td>5.4</td>
<td>Repetition</td>
<td>127</td>
</tr>
<tr>
<td>5.4.1</td>
<td>While Away the Time</td>
<td>130</td>
</tr>
<tr>
<td>5.5</td>
<td>An Example Using Computational Thinking</td>
<td>131</td>
</tr>
<tr>
<td>6</td>
<td>Data Structures</td>
<td>135</td>
</tr>
<tr>
<td>6.1</td>
<td>Abstract Data Types</td>
<td>135</td>
</tr>
<tr>
<td>6.2</td>
<td>A Non-Technical Abstract Type</td>
<td>136</td>
</tr>
<tr>
<td>6.3</td>
<td>Advantages of ADTs</td>
<td>137</td>
</tr>
<tr>
<td>6.4</td>
<td>Data Structures</td>
<td>138</td>
</tr>
<tr>
<td>6.5</td>
<td>Strings</td>
<td>139</td>
</tr>
<tr>
<td>6.5.1</td>
<td>A First Look at Objects</td>
<td>141</td>
</tr>
<tr>
<td>6.6</td>
<td>Lists and Tuples</td>
<td>141</td>
</tr>
<tr>
<td>6.6.1</td>
<td>Tuples</td>
<td>142</td>
</tr>
<tr>
<td>6.7</td>
<td>An Example Using Computational Thinking</td>
<td>142</td>
</tr>
<tr>
<td>7</td>
<td>Procedural Programming</td>
<td>147</td>
</tr>
<tr>
<td>7.1</td>
<td>Functions Redux</td>
<td>147</td>
</tr>
<tr>
<td>7.2</td>
<td>Functions in Python</td>
<td>150</td>
</tr>
<tr>
<td>7.3</td>
<td>Sub-Routines with Parameters and Values</td>
<td>153</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Procedures with Parameters</td>
<td>153</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Functions with Parameters</td>
<td>153</td>
</tr>
<tr>
<td>7.3.3</td>
<td>Input Values and Variables</td>
<td>154</td>
</tr>
<tr>
<td>7.4</td>
<td>Namespaces and Variable Scope</td>
<td>155</td>
</tr>
<tr>
<td>7.5</td>
<td>Exception Handling</td>
<td>158</td>
</tr>
<tr>
<td>7.5.1</td>
<td>Handling Exceptions in Python</td>
<td>158</td>
</tr>
<tr>
<td>7.6</td>
<td>File I/O</td>
<td>161</td>
</tr>
<tr>
<td>7.7</td>
<td>An Example Using Computational Thinking</td>
<td>164</td>
</tr>
</tbody>
</table>
## Advanced: Data and Computation

### 8 Object-Oriented Programming (OOP)

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1 Zen and the Art of Object-Oriented Programming</td>
<td>169</td>
</tr>
<tr>
<td>8.1.1 Installing Java</td>
<td>169</td>
</tr>
<tr>
<td>8.2 The Road to OOP</td>
<td>173</td>
</tr>
<tr>
<td>8.2.1 Designing the Objects</td>
<td>175</td>
</tr>
<tr>
<td>8.3 Solving Problems with Software Objects</td>
<td>178</td>
</tr>
<tr>
<td>8.3.1 A Detailed Example in Java</td>
<td>180</td>
</tr>
<tr>
<td>8.3.2 Solving Computational Problems Using Objects</td>
<td>183</td>
</tr>
<tr>
<td>8.3.3 OOP Fundamentals</td>
<td>185</td>
</tr>
<tr>
<td>8.4 What Is Java?</td>
<td>186</td>
</tr>
<tr>
<td>8.4.1 Class Structure</td>
<td>186</td>
</tr>
<tr>
<td>8.4.2 Object-Oriented Concepts and Computational Thinking Principles</td>
<td>190</td>
</tr>
<tr>
<td>8.4.3 OOP Advantages and Motivations</td>
<td>194</td>
</tr>
<tr>
<td>8.5 Data Structures and I/O in Java</td>
<td>195</td>
</tr>
<tr>
<td>8.5.1 Arrays and ArrayLists</td>
<td>197</td>
</tr>
<tr>
<td>8.5.2 Exception Handling in Java</td>
<td>198</td>
</tr>
<tr>
<td>8.5.3 I/O in Java</td>
<td>200</td>
</tr>
</tbody>
</table>

### 9 Databases and MDM

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1 Mind Your Data</td>
<td>203</td>
</tr>
<tr>
<td>9.1.1 Data Abstraction</td>
<td>204</td>
</tr>
<tr>
<td>9.1.2 Data Models and Master Data Management</td>
<td>206</td>
</tr>
<tr>
<td>9.1.3 Three LEVELS of Data Model Abstractions</td>
<td>208</td>
</tr>
<tr>
<td>9.2 Database Management</td>
<td>208</td>
</tr>
<tr>
<td>9.2.1 DataBase Life Cycle (DBLC) PHASES</td>
<td>210</td>
</tr>
<tr>
<td>9.2.2 LAYERS of Database Abstraction</td>
<td>213</td>
</tr>
<tr>
<td>9.2.3 Client-Server Database Application TIERs</td>
<td>214</td>
</tr>
<tr>
<td>9.3 Relational Database Model</td>
<td>214</td>
</tr>
<tr>
<td>9.3.1 Data Models and Database Models</td>
<td>215</td>
</tr>
<tr>
<td>9.4 Database Modeling and Querying</td>
<td>218</td>
</tr>
<tr>
<td>9.4.1 Databases, Data Warehouses, and Data Lakes</td>
<td>218</td>
</tr>
<tr>
<td>9.4.2 Structured Query Language (SQL)</td>
<td>219</td>
</tr>
<tr>
<td>9.4.3 Embedded SQL</td>
<td>220</td>
</tr>
<tr>
<td>9.5 Normalization</td>
<td>220</td>
</tr>
<tr>
<td>9.5.1 1st Normal Form (1NF)</td>
<td>223</td>
</tr>
<tr>
<td>9.5.2 2NF and 3NF</td>
<td>224</td>
</tr>
<tr>
<td>9.5.3 Denormalization</td>
<td>224</td>
</tr>
</tbody>
</table>
# Machine Learning and Data Science

## 10.1 Computational Thinking and Artificial Intelligence
- What Is Machine Learning? ........................................ 229

## 10.2 Getting Started with Machine Learning
- Movie Night with the Family ..................................... 231
- Building the Generalized Movie Night Model .................. 232
- Selecting a Movie Night Algorithm ............................... 234
- Implementing the Movie Night Decision Trees ................. 235
- Ensemble Methods .................................................. 239
- Miss the Random Forests for the Decision Trees .............. 240
- Generalized Classification Task ................................. 241
- Unsupervised Learning via k-Means ............................ 243

## 10.3 Elements of Machine Learning
- Machine Learning Writ Large .................................... 249
- ML Definition: Tasks .............................................. 250
- ML Definition: Models ............................................ 251
- ML Definition: Performance Measures .......................... 253
- Model Building and Assessment ................................ 253

## 10.4 Data Science and Data Analytics
- Business Intelligence/Business Analytics ........................ 256
- The Jataka Analytics .............................................. 257
- Is Data Science a Science? ........................................ 259

## 10.5 Bayesian Inference and Hypothesis Testing
- Null Hypothesis Significance Testing ............................ 260
- Hypothesis Testing .................................................. 262
- Bayes Theorem .................................................... 263
- Bayesian Hypothesis Testing ...................................... 265
- Bayesian Inference and Statistics ................................ 266
- Supervised Learning with Naive Bayes .......................... 267
- Common Statistical Metrics ...................................... 269

## 10.6 The Entropy Strikes Back
- Random Variables .................................................. 272
- Shannon Information, Shannon Information Entropy, and Information Gain ............................................. 275
- Entropy and Information in Terms of Number of States of Physical Systems ............................................. 277
- Entropy for Classification ......................................... 278
- Entropy for Decision Trees ........................................ 279

## 10.7 Learning in Decision Trees
- Sources of Error in Machine Learning .......................... 282

## 10.8 Machine Learning, Data Science, and Computational Thinking

---

**Bibliography** ................................................................ 285

**Index** .......................................................................... 287
### Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>Collection of sensor temperature values</td>
<td>4</td>
</tr>
<tr>
<td>0.2</td>
<td>Plot of Temperature vs. Time</td>
<td>5</td>
</tr>
<tr>
<td>0.3</td>
<td>Transformation of Data to Information to Knowledge</td>
<td>5</td>
</tr>
<tr>
<td>0.4</td>
<td>A Function is a Correspondence</td>
<td>9</td>
</tr>
<tr>
<td>0.5</td>
<td>One-to-One and Many-to-One Mappings</td>
<td>10</td>
</tr>
<tr>
<td>0.6</td>
<td>Graphical Representations of the two functions</td>
<td>10</td>
</tr>
<tr>
<td>0.7</td>
<td>A function can be represented as a black box</td>
<td>11</td>
</tr>
<tr>
<td>0.8</td>
<td>Some essential ideas in the theory of computation</td>
<td>20</td>
</tr>
<tr>
<td>1.1</td>
<td>Computational Thinking Steps</td>
<td>31</td>
</tr>
<tr>
<td>1.2</td>
<td>Signal degradation image directly from Hartley’s paper</td>
<td>40</td>
</tr>
<tr>
<td>1.3</td>
<td>Maxwell’s Demon at work</td>
<td>53</td>
</tr>
<tr>
<td>1.4</td>
<td>Data to Information to Knowledge</td>
<td>57</td>
</tr>
<tr>
<td>2.1</td>
<td>Accuracy vs Precision</td>
<td>60</td>
</tr>
<tr>
<td>2.2</td>
<td>Values, Data, and Data Representation</td>
<td>63</td>
</tr>
<tr>
<td>2.3</td>
<td>Positional Number Systems</td>
<td>64</td>
</tr>
<tr>
<td>2.4</td>
<td>8-fingered Simpsons</td>
<td>66</td>
</tr>
<tr>
<td>2.5</td>
<td>A simple electrical switch example</td>
<td>67</td>
</tr>
<tr>
<td>2.6</td>
<td>AND and OR circuits</td>
<td>67</td>
</tr>
<tr>
<td>2.7</td>
<td>Shannon and Boolean Circuits</td>
<td>70</td>
</tr>
<tr>
<td>2.8</td>
<td>Full Adder circuit based only on NAND gates</td>
<td>71</td>
</tr>
</tbody>
</table>
## FIGURES

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.9</td>
<td>Information Processing Workflow</td>
<td>79</td>
</tr>
<tr>
<td>2.10</td>
<td>Data Processing Workflow for Temperature Problem from Section 0.1</td>
<td>86</td>
</tr>
<tr>
<td>2.11</td>
<td>Difference between CS, CIS, and IT</td>
<td>87</td>
</tr>
<tr>
<td>3.1</td>
<td>The Fetch-Execute Cycle in the CPU</td>
<td>95</td>
</tr>
<tr>
<td>3.2</td>
<td>NetBeans IDE</td>
<td>99</td>
</tr>
<tr>
<td>3.3</td>
<td>Python IDLE Shell and Text Editor</td>
<td>100</td>
</tr>
<tr>
<td>3.4</td>
<td>Python Virtual Machine and Compiler</td>
<td>106</td>
</tr>
<tr>
<td>3.5</td>
<td>Syntax, Semantic, and Logical Errors</td>
<td>110</td>
</tr>
<tr>
<td>4.1</td>
<td>An Informal Model of Fundamental Program Elements</td>
<td>120</td>
</tr>
<tr>
<td>5.1</td>
<td>Syntax of the selection control structure in Python</td>
<td>126</td>
</tr>
<tr>
<td>5.2</td>
<td>Gauss’s Summation Trick</td>
<td>129</td>
</tr>
<tr>
<td>6.1</td>
<td>Abstract Fastener Type</td>
<td>136</td>
</tr>
<tr>
<td>6.2</td>
<td>Numeric Abstract Data Type</td>
<td>137</td>
</tr>
<tr>
<td>6.3</td>
<td>Data Types and Representations</td>
<td>145</td>
</tr>
<tr>
<td>7.1</td>
<td>Coffee Making Metaphor</td>
<td>147</td>
</tr>
<tr>
<td>7.2</td>
<td>Coffee Maker Metaphor</td>
<td>148</td>
</tr>
<tr>
<td>7.3</td>
<td>Coffee Maker Function Metaphor</td>
<td>149</td>
</tr>
<tr>
<td>7.4</td>
<td>Exception Handling</td>
<td>166</td>
</tr>
<tr>
<td>8.1</td>
<td>NetBeans IDE</td>
<td>171</td>
</tr>
<tr>
<td>8.2</td>
<td>Java Application Outline</td>
<td>172</td>
</tr>
<tr>
<td>8.3</td>
<td>Modular Programming Representation</td>
<td>175</td>
</tr>
<tr>
<td>8.4</td>
<td>Class Blueprint and Two Human Objects: Joe and Hannah</td>
<td>178</td>
</tr>
<tr>
<td>8.5</td>
<td>Object Reference Variable vs Primitive Variable</td>
<td>185</td>
</tr>
<tr>
<td>8.6</td>
<td>Java programs, classes, and methods</td>
<td>187</td>
</tr>
<tr>
<td>8.7</td>
<td>Association, Aggregation, and Composition</td>
<td>193</td>
</tr>
<tr>
<td>9.1</td>
<td>The raw notes for the home business transformed into structured data</td>
<td>204</td>
</tr>
<tr>
<td>9.2</td>
<td>The Data → Information → Knowledge data processing workflow</td>
<td>204</td>
</tr>
<tr>
<td>9.3</td>
<td>The idea of abstraction in the context of a map</td>
<td>205</td>
</tr>
<tr>
<td>9.4</td>
<td>The data processing workflow for the Contacts “database”</td>
<td>206</td>
</tr>
<tr>
<td>9.5</td>
<td>DataBase Life Cycle (DBLC)</td>
<td>211</td>
</tr>
<tr>
<td>9.6</td>
<td>ERD for Customers</td>
<td>212</td>
</tr>
<tr>
<td>9.7</td>
<td>Application Development Tiers and Layers of Database Abstraction</td>
<td>215</td>
</tr>
<tr>
<td>9.8</td>
<td>Database Normalization</td>
<td>222</td>
</tr>
<tr>
<td>10.1</td>
<td>Machine Learning Definition</td>
<td>230</td>
</tr>
<tr>
<td>10.2</td>
<td>Plot of Cats and Dogs by Features</td>
<td>231</td>
</tr>
<tr>
<td>FIGURES</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>10.3 Machine Learning Model</td>
<td>232</td>
<td></td>
</tr>
<tr>
<td>10.4 Decision Tree Classification Example</td>
<td>234</td>
<td></td>
</tr>
<tr>
<td>10.5 Movie Night Decision Tree Image</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>10.6 Ensemble Method: Stacking.</td>
<td>240</td>
<td></td>
</tr>
<tr>
<td>10.7 Ensemble Method: Bagging.</td>
<td>240</td>
<td></td>
</tr>
<tr>
<td>10.8 Ensemble Method: Boosting.</td>
<td>241</td>
<td></td>
</tr>
<tr>
<td>10.9 Machine Learning Model Detail</td>
<td>242</td>
<td></td>
</tr>
<tr>
<td>10.10 Supervised learning approach for a general predictive model</td>
<td>244</td>
<td></td>
</tr>
<tr>
<td>10.11 Iris Dataset Scatterplot</td>
<td>246</td>
<td></td>
</tr>
<tr>
<td>10.12 Structured, Un-Structured, and Semi-Structured Data for the Iris Dataset.</td>
<td>248</td>
<td></td>
</tr>
<tr>
<td>10.13 How Artificial Intelligence (AI) and Machine Learning (ML) are related.</td>
<td>249</td>
<td></td>
</tr>
<tr>
<td>10.14 Eight Common ML Approaches</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>10.15 Traditional Computer Science vs Machine Learning paradigms</td>
<td>251</td>
<td></td>
</tr>
<tr>
<td>10.16 Machine Learning Categories</td>
<td>251</td>
<td></td>
</tr>
<tr>
<td>10.17 Elements of Classical Machine Learning</td>
<td>252</td>
<td></td>
</tr>
<tr>
<td>10.18 Splitting the full dataset for supervised learning.</td>
<td>253</td>
<td></td>
</tr>
<tr>
<td>10.19 Training Testing Data</td>
<td>254</td>
<td></td>
</tr>
<tr>
<td>10.20 Hypothesis Testing, Two-Tailed Z Test, and Probability Tree</td>
<td>264</td>
<td></td>
</tr>
<tr>
<td>10.21 Confusion Tables</td>
<td>269</td>
<td></td>
</tr>
<tr>
<td>10.22 Rényi Entropy</td>
<td>278</td>
<td></td>
</tr>
<tr>
<td>10.23 Decision Tree Example with Entropies</td>
<td>281</td>
<td></td>
</tr>
</tbody>
</table>
## Tables

0.1 Organized Sensor Values .......................................................... 4

1.1 List of Variables and Control Structures in Algorithm 1.1 .............. 34

2.1 State Table for PB&J System ...................................................... 82

2.2 State Transition Table for a simplified Turing Machine .................... 84

4.1 Simple symbol table for first two statements .................................. 116

4.2 Final symbol table after the third statement ................................... 117

4.3 Some Python Operators ............................................................ 117

5.1 Boolean and Relational Operators in Python .................................... 125

5.2 ASCII Art for a backslash made out of asterisks .............................. 132

6.1 Two Kinds of Fasteners: Two Fastener Structures ............................ 136

6.2 Two Python Numeric Data Structures ........................................... 137

8.1 Class Blueprint for Human Objects .............................................. 176

8.2 Physical, conceptual, and software objects .................................... 177

9.1 Different levels of abstraction ...................................................... 207

9.2 Conceptual, Logical, and Physical Data Models ................................. 209

9.3 Database models extend data models by adding in constraints. ........... 216

9.4 Customer Table in the DBMS ....................................................... 217
9.5 Orders Table in the DBMS .................................................. 217
9.6 Logical-Physical Relational Terminology .............................. 218
9.7 Table showing the three anomalies .................................... 221
9.8 Emails Repeating Group .................................................. 223
9.9 Multi-Valued Attribute in Email column .............................. 223
9.10 New Email-Student Table ............................................... 223
9.11 2NF and 3NF Issues ...................................................... 224

10.1 Show Preferences ......................................................... 233
10.2 Decision tree dataset for the family movie night .................... 235
Algorithms

0.1 Algorithm for doing dishes ................................................. 14
1.1 Algorithm for calculating average temperature ......................... 35
1.2 Algorithm for calculating average temperature using Procedures .......... 36
2.1 Algorithm for adding $29 + 14 = 43$ using Roman numerals .............. 62
5.1 Algorithm for printing an ASCII art picture of a backslash ................ 132
6.1 Algorithm for computing the Chinese zodiac animal for a particular year .......... 143
7.1 Algorithm for making coffee ............................................ 148
7.2 Algorithm for the morning coffee routine using Functions ................. 150
7.3 Algorithm for computing distance between two Cartesian points .......... 165
## Code Listings

0.1 Canonical “Hello World!” program in Java ........................................ 15  
0.2 Canonical “Hello World!” program in Python .................................... 16  
3.1 “Hello World!” program for the Python shell .................................... 99  
3.2 “Hello World!” program for the Python Text Editor ........................... 100  
3.3 First attempt at computing the average temperature .......................... 109  
4.1 Reserved words or keywords in Python ........................................... 112  
4.2 Operators and Operands in Python ................................................. 116  
4.3 Compute area of circle for crayon of given length .............................. 121  
5.1 Final algorithm for normal students’ approach to summing numbers ...... 130  
5.2 Print an ASCII art picture of a backslash made out of asterisks ............ 133  
6.1 Compute Chinese zodiac sign for a given year .................................... 143  
7.1 Compute Cartesian Distance ......................................................... 165  
8.1 “Hello World!” program in Java for NetBeans ................................ 171  
8.2 Java application to create a BankAccount for Joe and make some deposits. 181  
9.1 Example of SQL statements .......................................................... 219  
9.2 Example of Embedded SQL in Java ............................................... 221  
10.1 Decision Tree Classifier for the movie night dataset ......................... 237  
10.2 k-Means clustering on the Iris Dataset .......................................... 245  
10.3 Gaussian Naive Bayes classifier on the Iris Dataset .......................... 268
Advanced: Data and Computation

8 Object-Oriented Programming (OOP) 169
8.1 Zen and the Art of Object-Oriented Programming
8.2 The Road to OOP
8.3 Solving Problems with Software Objects
8.4 What Is Java?
8.5 Data Structures and I/O in Java

9 Databases and MDM ..................... 203
9.1 Mind Your Data
9.2 Database Management
9.3 Relational Database Model
9.4 Database Modeling and Querying
9.5 Normalization

10 Machine Learning and Data Science 227
10.1 Computational Thinking and Artificial Intelligence
10.2 Getting Started with Machine Learning
10.3 Elements of Machine Learning
10.4 Data Science and Data Analytics
10.5 Bayesian Inference and Hypothesis Testing
10.6 The Entropy Strikes Back
10.7 Learning in Decision Trees
10.8 Machine Learning, Data Science, and Computational Thinking

Bibliography ............................... 285

Index ........................................ 287
10. Machine Learning and Data Science

**Man is the best computer we can put aboard a spacecraft and the only one that can be mass produced with unskilled labor.**
– Wernher von Braun, rocket engineer (1912-1977)

10.1 Computational Thinking and Artificial Intelligence

Computational thinking can be thought of as an extension of thinking critically or reasoning with evidence. In a way, it systematizes critical thinking and evidence-based reasoning. This often involves logical thinking and argumentation, where we mean the classical, formal sense of argumentation as opposed to bickering. We could say, as we saw in Section 1.2, that this kind of logical thinking builds upon three fundamental ways to approach a problem: ad hoc thinking, deductive thinking, or inductive thinking.

Thinking, then, seems to be at the very core of being human and being intelligent enough to solve problems in the world. As such, people began to wonder if we could model thinking and intelligence. In the 17th century, as we saw in Section 2.5.1, Leibniz postulated the *Characteristica Universalis*, a new language to represent human thought where each letter represented some concept and these could then be combined and manipulated according to a set of logical rules via the *Calculus Ratiocinator* to compute all mathematical and scientific knowledge. Somewhat more recently, Turing started to think about machines, hypothetical ones which came to be called Turing Machines as we saw in Section 0.7.1, that could solve any computable function, and which seemed to lay some of the foundation for an eventual *Calculus Ratiocinator* of sorts.

It was at this point that neurophysiologist Warren S. McCulloch and logician Walter H. Pitts entered the scene. McCulloch, known for drinking whiskey and eating ice cream till 4am every day\(^1\), was

---

\(^1\)See http://nautil.us/issue/21/information/the-man-who-tried-to-redeem-the-world-with-logic
interested in modeling a theoretical foundation for the human brain in the style of Leibniz’s logical
calculus, and using this simple, logical foundation to build complex neural activity in the same way
that Russell and Whitehead’s *Principia Mathematica* showed all of mathematics could be built from a
simple logical foundation.

So when he read Turing’s 1936 paper entitled, “On Computable Numbers, With an Application
to the Entscheidungsproblem,” McCulloch made the connection that the brain could act like a Turing
Machine. He further reasoned that chains of neurons could be linked together by rules of logic to
create thoughts just as Russell and Whitehead had built all of mathematics by chaining together
simple propositions. McCulloch and Pitts built this computational theory of the brain, using Turing’s
mathematical definition of computation combined with logic, in their seminal 1943 paper, “A Logical
Calculus of the Ideas Immanent in Nervous Activity.”

At the time, neurons were known to be the building blocks of human thought and intelligence in
the brain and people had been studying them for a while. Neurons themselves were thought to be
relatively simple cells that did some processing on multiple inputs and produced a single output spike,
the action potential. McCulloch and Pitts had showed that these spikes could be combined to do logical
and arithmetical operations.

Meanwhile, several biologists had been studying the squid’s giant neuron in the early 20th century,
and biophysicists Alan Hodgkin and Andrew Huxley soon created a circuit model of neurons as a set of
nonlinear differential equations to calculate how conductances in cell membranes varied with time and
voltage. People like the mathematician Norbert Wiener started to see the connection between biological
systems like neurons and neural networks and control systems. Wiener, who was at MIT, had taken
Pitts under his wing. Soon, Pitts also met von Neumann, who had separately been having conversations
with Shannon and Turing about intelligence and machines that could also simulate the kind of thinking
encoded in neural networks.2

In fact, Turing gave a talk to the London Mathematical Society in 1947 in which he said, quite
presciently as we’ll see in just a bit: “What we want is a machine that can learn from experience.” He
went on to say,

“It would be like a pupil who had learnt much from his master, but had added much more
by his own work. When this happens I feel that one is obliged to regard the machine as
showing intelligence.”

Turing even went on to propose a test, now called the Turing Test, to gauge if a machine could
be considered as intelligent as a human. This test was based on a Victorian parlour game called
the Imitation Game3 and posited that machines should be considered as intelligent as humans when
objective observers couldn’t distinguish between a human and machine participant.

The idea of intelligent machines soon garnered quite a bit of interest. In 1955, computer scientist
John McCarthy coined the term “Artificial Intelligence” for it and organized the famous Dartmouth
conference in 1956, which was attended by such luminaries as Claude Shannon, Marvin Minsky, Allen
Newell, and Herbert A. Simon and marked the birth of the field of Artificial Intelligence (AI).

Artificial intelligence, in the computational sense, is often described in textbooks like Russell &
Norvig in terms of a rational agent or intelligent agent, which is any entity, virtual or physical, that can

---

2 As an interesting side-note, when von Neumann published his historic paper, “First Draft of a Report on the EDVAC,”
in 1945, it described the first proposed modern computer as a stored-program binary computing machine. In the paper, he
modelled the proposed machine after McCulloch and Pitts’ neural networks but replaced the neurons with vacuum tubes as
logic gates. The published paper only had one citation: McCulloch and Pitts’ 1943 paper.

3 See https://hsm.stackexchange.com/questions/5890/did-turing-invent-the-imitation-game-did-he-name-it
take sensory input from its environment and interpret it, put it into context with its existing knowledge, and then take actions which would maximize its chances of achieving its goals.

The real problem is that, like for life itself, there is no good generalizable definition of intelligence. Although we’ve made quite a few advances in neurobiology, we still don’t have any exact idea of what humans do when they do things that we deem to be intelligent. All we know is that humans do things which we cannot yet automate with computational systems.

Over the years, the popularity of AI waxed and waned and, with it, the definition of what constituted intelligence. The original goal of mimicking human intelligence soon came to be known as General AI (also known as Strong AI or Hard AI). The more modest, and achievable, goal of simply having “machines that learn,” as Turing himself had initially suggested, in a specialized context or in niche applications was then given the corresponding term of Narrow AI (also known as Weak AI or Soft AI). Over time, this Narrow AI, where you simply have a “machine that learns,” has also come to be called Machine Learning.

### 10.1.1 What Is Machine Learning?

One of its earliest definitions was proposed by the computer scientist Arthur Samuel, who coined the term in 1959 and defined it as a “field of study that gives computers the ability to learn without being explicitly programmed.” In some sense, every useful program learns something. For example, an implementation of Newton’s method learns the roots of a polynomial.

As we saw in Section 0.2, we tend to learn new things by memorization and generalization. We use memorization to accumulate individual facts and generalization to deduce new facts from old facts. The most widely used definition of machine learning uses these ideas and follows Turing’s intuition as elaborated by computer scientist Tom Mitchell:

A computer program is said to learn from experience “E”, with respect to some class of tasks “T” and performance measure “P”, if its performance at tasks in “T” as measured by “P” improves with experience “E”

We can visualize this in Figure 10.1 where we add in the idea that once its performance is deemed to be sufficient, it outputs a final prediction, the result of its learning.

### 10.2 Getting Started with Machine Learning

Machine learning is thus the process of using past experiences to improve future performance on a particular task. Algorithms accomplish this by automatically learning to make useful inferences from implicit patterns in data. For example, linear regression learns a curve that is a model of a collection of examples. This model that’s created can then be used to make predictions on new data points.

**Example 10.1 The Grumpy Cat:** Suppose you have a jumbled up collection of pictures, where each picture is of a dog or a cat. Perhaps you gathered the photos from your grandparents’ Facebook feed which seems to be dominated by such photos and now they want them sorted. Your task, in this case, would involve separating the pictures into two categories: one pile of dog photos and one pile of cat photos. A program could learn to perform this task for you by modeling pictures that have already been sorted.

---

4 You could argue that biology is the only science that cannot define the object of its study. I once read something like that in an introductory Biochemistry textbook but a more precise perspective is given by Carol E. Cleland and Christopher F. Chyba in their paper, “Defining ‘Life’,” where they conclude there is no broadly accepted definition of life.
Chapter 10. Machine Learning and Data Science

Figure 10.1: Following the ideas of Turing and Mitchell, one definition of Machine Learning can be: a Machine, or a computer program, is said to Learn if its Performance at a Task improves with Experience. The Experience is the input Data and the Performance is quantified by a Performance Measure. Once its performance is sufficient, it outputs a final prediction, the result of its learning.

As such, you might sort through the first hundred or so pictures and provide these sorted photos to the program so it could learn about dog and cat photos. Once it learns how to separate the photos, you can then then evaluate its performance by calculating the percentage of correctly classified pictures and, if that performance is satisfactory, set it loose on the rest of the unsorted photos. This is the machine learning task in a nutshell.

So why machine learning? The reason is that it is often too difficult to design a set of rules “by hand”. Machine learning is about automatically extracting relevant information from data and applying it to analyze new data. For these machine learning algorithms, the experience is captured in the data that’s used as input for the algorithm. Performance of the algorithm improves with experience as the algorithm learns. Learning, in this case, is guided by a quantitative assessment that’s often called an objective function or a loss function and is associated with a notion of some loss that should be minimized or, alternatively, a gain that should be maximized.⁵

Machine learning is thus the study of algorithms that:

- improve their performance, P
- at some task, T
- with experience, E

The business problem⁶ determines the tasks, T; so once a business problem is defined, we can figure out the kinds of tasks involved and classify algorithms by the kinds of tasks they can solve. Once you know the kinds of tasks and the associated algorithms, you pick the most appropriate algorithm based on its learning style, which will be dictated by the kinds of data and resources (labelled, unlabelled, etc.) that are available to help solve the task.

⁵ As a side note, most optimization problems seek to minimize a loss function, which is analogous to optimizing the objective function by either minimizing the loss or maximizing the gain. In regards to terminology, a loss function is a part of a cost function which is a type of an objective function. One of the most popular optimization algorithms is Gradient Descent (GD), and its cousin Stochastic Gradient Descent (SGD), which minimizes the cost function. GD finds a global minimum only for convex loss functions whereas SGD, which is similar to Simulated Annealing, is much more likely to find a global optimum in general.

⁶ As usual, when we use the term business here we mean a generic organization of any sort, not just a corporation or commercial enterprise.
10.2 Getting Started with Machine Learning

10.2.1 Movie Night with the Family

An example might help illustrate the overall process. Let’s start with the task of choosing a show to watch for the next family movie night. But before we actually embark on figuring out that complex task, as a first approximation, let’s start with something simpler, take my wife, for example.7

7Our movie nights never last long and, coupled with our laziness adherence to tradition, we’ve resorted to just keeping the name and watching TV shows instead.

There are, of course, a plethora of jokes that could be made here but, as the saying goes, it takes a smart husband to have the last word and not use it. Besides which, I am of the same mind as Winston Churchill in knowing for a fact that, “My most brilliant achievement was my ability to persuade my wife to marry me.”

Figure 10.2: Plot of Cats (red) and Dogs (blue) by Feature 1 (ear size) and Feature 2 (whisker size).
Having known my wife for a number of years, I have a pretty good sense of the kinds of shows and movies she likes or dislikes. For example, I know she likes *Outlander* based on all the times she forced me to watch it, only to have me actually become a fan of the show. Of course, she wasn’t quite as enthusiastic when we watched *Man From Earth* and *TinTin* but I know she was happy for me.

After all these shows and movies that we’ve seen together, I’ve formulated a model in my head of my wife’s thought processes in regards to whether she’ll like a certain show or not. In addition to the shows themselves, I know whether she likes watching them on Hulu on TiVo or on Netflix on the FireTV box. All of these factors contribute to my overall mental model of my wife.

Following the machine learning model in Figure 10.1, picking a show my wife enjoys would be the task and my mental model would be the machine that learns; the data I’ve gathered over the years about shows my wife enjoys would be the experience; realizing she didn’t like *Man From Earth* but did enjoy *Supergirl* when I suggested it helps me gauge the accuracy of my suggestions, which would be the performance measure. After fine tuning my mental model over the years, I should then be able to use that model to reliably predict if she would like a potential new show.

In the next approximation, I might create a similar mental model for the whole family and use this more generalized model to predict whether the family, as a whole, would like a certain show on a certain device, as shown in Figure 10.3.

### 10.2.2 Building the Generalized Movie Night Model

So our general task, as seen in Figure 10.3, is to pick a TV show the whole family will enjoy. Now that we know the task at hand, we need to work on the experience (the data), the machine that learns (the model), and the performance measure (the accuracy). Instead of relying upon my own mental model, let’s take a more structured approach and use a machine learning model instead.

There are many parts to finding the full machine learning solution but the first part of our Machine Learning Life Cycle involves:

- Figuring out the task specifications
- Finding the available data and resources
- Deciding on the appropriate algorithm

We’ve already figured out the task so let’s start to find the data and resources. To keep things simple, assume we have two kids, two parents, and two grandparents. If your particular family structure

---

9Who knew a show about time travel that never discussed the physics of time travel could turn out to be actually good?

10In general, we have many options for both the model and the performance measure but accuracy is generally an intuitive measure so we’ll use that as a stand-in for some of the more informative measures we’ll meet in places like Section 10.5.7.
is lacking in any of these member numbers, please feel free to substitute your hamster, cat, dog, or
other favourite pet for the missing role. Similarly, if you have a surfeit of family members beyond the
requisite two in each category, please feel free to delegate the extra family members to the role of pets
(this can be a fun, if somewhat emotionally devastating, family activity in its own right, of course).

We can then survey each family member about all the shows anyone in the family has seen and
record each family member’s ratings for each show. We can gather all of this data for the whole family
in Table 10.1.

In general, the rows of our dataset are called samples or instances or observations and the columns
of our dataset are called features or variables. We’ll use these features to learn about the problem
and one of these columns, or features, will be designated to be the label or target variable. The target
variable is our dependent variable and our features are the independent variables.

This kind of task, in which we have a column with a label indicating to which category, or class, the
sample belongs, is called classification and it falls under the rubric of supervised learning because we
have example data with the label of the class already determined in the dataset we previously gathered.
This is the same kind of task we dealt with above in Example 10.1 about the Grumpy Cat where we
had to classify whether a picture was a dog picture or a cat picture.

Classification depends on having one column, or feature, that indicates to which class the sample
belongs, this is the target variable. That label, or target variable, in our case is the UserRating column,
which can have one of two values: either a ThumbsUp or a ThumbsDown, to indicate whether that person
liked or disliked that show.

Our model will use some combination of the other features, or columns, to make its predictions;
the columns it uses to learn what prediction to make are called predictor variables and, for us, they’ll
be things like the Genre, Runtime, etc.

We’ll also want to make sure our dataset is in good shape; according to the well-tested aphorism,
“Garbage in, garbage out,” if our data is junky our machine will learn junk. So pre-processing the
dataset is very important and this kind of data cleaning or data munging usually involves things like:

- Finding missing or incorrect values; e.g., did everyone fill in all the columns for each show?
- Transforming categorical data, like text or boolean, into numeric data as some algorithms
don’t work well with non-numeric data; e.g., we’ll probably want to change the ThumbsUp and
ThumbsDown ratings to 1’s and 0’s since most algorithms probably don’t understand the concept
of a thumb, let alone whether its state is up or down.
- If an algorithm compares values of different features directly, you have to take care of differences
in scale between different features or columns; e.g., the Year’s numeric value is much higher
than just about any UserAge (for most humans in this century, at least) so we should probably
do things like normalize the values in both columns otherwise the value of a Year will always
dominate in any simple sum of values of Year and UserAge.
In addition, if we have a lot of columns or features, we might need to reduce the dimensionality of our dataset. We might do feature engineering by combining different columns or adding in new ones or do feature selection and remove the least informative ones. It’s not a terribly big problem here but imagine Netflix or Amazon and the crazy number of features they might track in trying to figure out if someone enjoys a certain show or movie. We’ll likely have to do some feature selection when we decide on a particular algorithm so let’s turn our attention to selecting an appropriate algorithm next.

### 10.2.3 Selecting a Movie Night Algorithm

One of our goals is to increase stakeholder buy-in and, since the kids are also pretty smart, we want them to understand how we arrived at our final prediction rather than just giving them an answer without any justification. One of the easiest algorithms to visualize and understand is the Decision Tree.

A decision tree example is shown in Figure 10.4. As you can see, this particular decision tree deals with our regular movie snacks: a bunch of fruits and vegetables. Anticipating picky eaters, we’ll use the decision tree to classify each snack as either lettuce, carrots, or oranges. The goal of a decision tree is to take heterogeneous collections of items and separate them so that, at the end, they’re all in homogenous subsets; in other words, take a jumbled up collection of cat and dog photos and, in the end, have two subsets made up entirely of only cat or only dog photos.

Some of the terminology associated with a decision tree is root node, decision node, and leaf node. The root node is at the very top and contains the full jumbled mess of stuff in it. The decision nodes start to separate one kind of thing from other kinds of things. Finally, the leaf nodes only have one kind, or class, of entities; in the case of the tree in Figure 10.4, each leaf node has only one class of

---

11 Getting buy-in from the kids, as all parents know, is the secret to a calm, restful night.

12 This is, of course, pure fiction; regular movie night snacks in our house consist of candies and popcorn like any other sane household. But, my father is a dentist and since this will be in print, let’s together pretend to have healthy inclinations.
Table 10.2: Decision tree dataset for the family movie night. In the above dataset, 1 stands for True and 0 stands for False in the Cartoon column. In the Label column, the 1 indicates Watch and 0 indicates Don’t Watch, corresponding to ThumbsUp and ThumbsDown in Table 10.1.

fruit or vegetable.
A decision tree typically uses a binary tree to model the data using selection conditions at each decision node. It looks like an upside down tree as it normally takes the form of a series of cascading questions. Each of these questions is cast as a selection that tests if a certain feature satisfies a certain condition; e.g., in the sample tree, one of our decision nodes asks, “Is the shape of the fruit or vegetable round?” For our dataset, a selection question might well be along the lines of, “Is the Genre or Format of the show a cartoon?”

Therefore, in order to create a decision tree, we’ll need to make a table of the data: each row, or instance, should be a show we’ve asked the family about. Each column, or feature, should represent some characteristic of the show like Format or Runtime. The values of those features should be amenable to being used for asking a yes or no question so we’ll have to transform our initial dataset in Table 10.1 to meet this format requirement. We’ll also take this opportunity to do some feature selection and keep only the columns we think will be most helpful, or, as we’ll see in Section 10.7, most informative in the sense of Shannon Information, in making our final predictions. The cleaned and munged version of our original dataset is now shown in Table 10.2, where we also add a final column for the Label, or target variable, to indicate whether we should or should not watch that show, analogous to the UserRating column in our original dataset.

10.2.4 Implementing the Movie Night Decision Trees
Decision trees are easy to understand, interpret, and visualize. They have many advantages like the ability to automatically do feature selection or the ability to handle categorical and numerical data with little data preparation. They’re unaffected by non-linear relationships but they are susceptible to overfitting with overly-complex trees. They can also be unstable and have high variance where small variations in data can cause very different trees to be generated. However, this can be mitigated by using bagging or boosting ensemble approaches as we’ll see in Section 10.2.5. Random Forests, which utilize a bagging approach, can also help with finding a more global solution as greedy methods alone are unable to guarantee global optimization. But they might create biased trees so a balanced dataset is very important in such cases.

We’ll see some of these more technical details about decision trees in Section 10.7 but let’s
summarize some of those details here, as well. We won’t spend too long on these details for now, as it’ll be more helpful to first see the actual code to implement the full decision tree algorithm. In brief, though, decision trees try to create homogeneous subsets that have low impurity or low entropy. There are many ways to grow these trees. Things like the criteria to use for splitting nodes, the number of splits per node, when to stop splitting, how to prune the trees, etc. The different values we use for these parameters will result in different possible structures of the trees, as we’ll see in gruesome detail for our fruit and vegetable tree in Section 10.7.

For now, let’s implement our initial decision tree model! We will use Python to implement all our initial models in this chapter; not only is Python very popular in machine learning and data science but the vast array of specialized libraries make the analytic task almost trivial, almost as much as some Business Intelligence tools like Qlik, PowerBI, etc.

One of the most popular libraries is scikit-learn, which allows us to create a wide variety of models with very little code. There are some vocabulary differences in scikit-learn’s implementation: machine learning algorithms are called estimators and the library itself is often referred to as sklearn. We’ll use sklearn to train, test, and evaluate our models all while only using the default parameters and hyperparameters.

Let’s start by loading the required libraries in Python. Any libraries that are not already installed on your system can be installed by using pip or pip3. The only exception is graphviz, which also needs its binaries installed on your system separately.

```python
### Load Required Libraries
from sklearn.tree import DecisionTreeClassifier  # Decision Tree Classifier
from sklearn.model_selection import train_test_split  # train_test_split function
from sklearn import metrics  # scikit-learn metrics for accuracy
from sklearn.tree import export_graphviz  # Generates graphviz of DT model
from sklearn.metrics import confusion_matrix  # Get Confusion Matrix
from sklearn.metrics import accuracy_score  # Get Accuracy
from sklearn.metrics import classification_report  # Get Classification Report
from sklearn.metrics import average_precision_score  # Get Avg Precision
from IPython.display import Image  # Alternate to matplotlib
import pandas as pd  # DataFrames
import numpy as np  # Numeric Python Library
import graphviz  # Graphs in DOT language
import pydotplus  # Python interface to DOT language
```

Once we’ve gotten a bunch of those libraries out of the way, we can start loading the dataset, printing summary statistics, and extracting our three feature columns (cartoon, year, and runtime) and the target class column (label). We will first have to separately save the movie night dataset from Table 10.2 into a file called ml-dataset.csv so that we can then load it and process it with a decision tree classifier in our program; you can call that file something different but please do change the code below to reflect that change, as well:

```python
### Load the dataset and print summary statistics
df = pd.read_csv("ml-dataset.csv")
print("Summary Statistics:")
print(df.describe())

### Split dataset into feature columns (data) and target variable (label)
feature_cols = ['cartoon', 'year', 'runtime']
```
X = df[feature_cols]  # The Features; Predictor Variables
y = df.label       # The Labels; Target Variable

This creates a numpy dataframe object called X which contains all the instances in our dataset. It also creates a dataframe object called y which contains the class labels for each instance. We’ll use both of these in Listing 10.1 where we’ll create and use our main decision tree classifier method for the movie night dataset from Table 10.2:

```python
### Split dataset into training set (70%) and test set (30%)
indices = range(X.shape[0])
(X_train, X_test, y_train, y_test, indices_train, indices_test) = \
    train_test_split(X, y, indices, test_size=0.3, random_state=1)

### Create the Decision Tree classifier default model
clf = DecisionTreeClassifier()

# Train the model then Predict with it:
trained_model = clf.fit(X_train, y_train)  # Train Decision Tree Classifier
y_pred = trained_model.predict(X_test)    # Predict the response for test dataset

# Model Accuracy determines how often the classifier is correct
print(metrics.accuracy_score(y_test, y_pred))
print(metrics.average_precision_score(y_test, y_pred))

### Get actual/predicted shows and statistics:
shows=df['show']
[print(shows[i]) for i in indices_test]
actual = np.array(y_test)
predicted = np.array(y_pred)
print('Confusion Matrix:', confusion_matrix(actual, predicted))
print('Accuracy Score:', accuracy_score(actual, predicted))
print('Report: ', classification_report(actual, predicted))
```

Listing 10.1: Decision Tree Classifier for the movie night dataset.

We start by splitting the dataset into training and testing datasets on Lines 1 - 4. Next, we create the decision tree classification algorithm object, clf. We then train the algorithm on the training data using its fit() function on Line 10 to create our tentative model, trained_model. We learn by training on the data and checking the performance so a trained model is also often called the model we learned and this process is also called learning the model and the model is called the learned model. We can then use the trained model on Line 11 to predict the class label for each sample, each show, as either Watch (0) or Don’t Watch (1). That’s it, our main classification task is done!

Next, we start to assess how well our trained model did. On Lines 13 - 15, we calculate and print the accuracy and precision scores to help evaluate our algorithm’s performance. On Lines 18 - 19, we get the list of shows that were labelled as Watch (1) and then, in Lines 21 - 25, we calculate a few more evaluation metrics to get a sense of how well our decision tree algorithm did.

Here are some of the results for our classifier:
Accuracy: 0.6666666666666666
Average PrecisionScore: 0.6666666666666666

Shows to watch:
HeMan
Password Plus
Daniel Tiger

We can also visualize the decision tree itself by using the graphviz binaries and graphviz python interface as shown below; the resulting image is shown in Figure 10.5.

```python
### Visualize the result with graphviz library and binary:
from sklearn.externals.six import StringIO # Get StringIO functions
dot_data = StringIO()
export_graphviz(trained_model, out_file=dot_data, filled=True, rounded=True,
special_characters=True, feature_names = feature_cols,
class_names=['Watch','DON'T Watch'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('ml-results.png')
Image(graph.create_png())
```

Figure 10.5: Decision tree built from the movie night example python code showing the nodes and splitting on the cartoon and year features.

We could do some additional processing like creating a ROC curve and computing the AUC, as well; we’ll see what the ROC, AUC, and some of these more funky measures are in Section 10.5.7:

```python
### Make ROC Curve
from sklearn.metrics import roc_auc_score # Get ROC and AUC score
```
10.2 Getting Started with Machine Learning

```python
from sklearn.metrics import roc_curve  # Get ROC data

# y_score = trained_model.fit(X_train, y_train).predict_proba(X_test)
y_score = trained_model.predict_proba(X_test)[:, 1]

print("ROC AUC Score:")
print(roc_auc_score(actual, y_score))

(fpr, tpr, thre) = roc_curve(actual, y_score)
print("False Positive Rate, True Positive Rate, and Threshold:")
print(fpr, tpr, thre)

# Create ROC Curve
import matplotlib.pyplot as plt

plt.plot(fpr, tpr, label='ROC curve')
plt.plot([0, 1], [0, 1], 'k--', label='Random guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.xlim([-0.02, 1])
plt.ylim([0, 1.02])
plt.legend(loc="lower right")

# Save ROC Curve
plt.savefig('ml-roc_curve.png', dpi=200)
```

We can even use automated machine learning (autoML) tools like tPot and auto-sklearn to automate the following tasks:

- Feature Selection
- Feature Processing
- Feature Construction
- Model Selection
- Parameter and Hyperparameter Optimization

These tools are very powerful, as can be seen in this quote from the tPot documentation where we see it evaluates multiple machine learning models and can even do data cleaning and dimensionality reduction (which we’ll learn about in just a bit) all with just a handful of lines of code!

“AutoML algorithms aren’t as simple as fitting one model on the dataset; they are considering multiple machine learning algorithms (random forests, linear models, SVMs, etc.) in a pipeline with multiple preprocessing steps (missing value imputation, scaling, PCA, feature selection, etc.), the hyperparameters for all of the models and preprocessing steps, as well as multiple ways to ensemble or stack the algorithms within the pipeline.”

Although our decision tree did pretty well, we can expand our analysis to be more accurate by using not just one decision tree but lots of trees in what’s termed a random forest, one of a powerful class of machine learning algorithms called ensemble methods which combine multiple models.

10.2.5 Ensemble Methods

In general, combining models tends to be surprisingly effective. There are three main approaches to creating these ensemble methods: stacking, bagging, and boosting.
Stacking, as seen in Figure 10.6, feeds the same original data to several different algorithms and the output of each of those intermediate algorithms is passed to a final algorithm which makes the final decision.

Bootstrap AGGREGATING (Bagging) uses the same algorithm on several different subsets of the original data that are produced by bootstrapping, random sampling with replacement from the original data. The result of each component is then averaged in the end, as seen in Figure 10.7. The random forests we saw before used Bagging on Decision Trees.

Boosting uses several different algorithms in a linear, sequential way rather than in a parallel way as in Bagging. Similar to Bagging, though, each subsequent algorithm is fed a subset of the original data that is produced by random sampling with replacement from the original data but, different from Bagging, the mis-classified data in the previous step is weighted more heavily in a subsequent step. As a result, the different algorithms are trained sequentially with each subsequent algorithm concentrating on data points that were mis-classified by the previous algorithm, as seen in Figure 10.8.\(^\text{13}\)

10.2.6 Miss the Random Forests for the Decision Trees

Although decision trees themselves are pretty good, as you can tell, there are many different divisions you can get depending on the particular hyperparameter choices you might make. Since ensemble

\(^{13}\)These ensemble methods, in general, increase the accuracy and, for bagging, reduce the variance as they combine several estimates from different models; for boosting and stacking, they will reduce the bias compared to the components, even if the variance can also be reduced.
methods tend to be very accurate, one strategy we can employ is to combine multiple single decision trees where each individual decision tree is based on a random sample of the training data. This kind of **random forest** of individual decision trees is usually more accurate than any one decision tree itself.

The random forest approach is very fast and works well with un-balanced and sparse data and does both classification and regression. Random forests are, however, prone to overfitting and the number of trees hyperparameter often needs to be tuned separately in order to control it to some extent. In addition, for regression problems, they are limited to predicting values in the range of the training data only. We’ll learn more about classification, regression, etc., in Section 10.3.

One reason a random forest tends to be more effective is that it doesn’t do any pruning as in a single decision tree and the fully grown tree gives us a higher resolution feature space that is split into more regions that are also smaller and more granular. In addition, since each individual decision tree in a random forest is learned from a random sample of the dataset and, at each node, a random set of features are used for splitting, the diversity amongst the trees is increased substantially. Random forests are more capable of handling noise, outliers, and overfitting because of its randomness in picking observations and variables for the model and its voting approach.

### 10.2.7 Generalized Classification Task

Our final process for the movie night algorithm involved all the parts outlined in Figure 10.9 and is what we use for most approaches to machine learning. Some machine learning approaches (like the supervised learning with decision trees we saw in Section 10.2.4) might utilize all these steps while others (like the unsupervised learning via the k-Means clustering algorithm in Section 10.2.8) might skip some steps like the training, etc.

In large measure, though, most machine learning approaches follow a similar stratagem. Let’s go through the process we will usually follow in making our predictions by seeing how it applies to determining what to watch on Family Movie Night:

1. **Our task** of selecting a show to watch for the next Family Movie Night started by identifying our most important stakeholders. This choice of stakeholders, along with the **kind** of task we’re addressing, dictated the **choice** of algorithm. In this case, we chose a **decision tree classification** algorithm to make the decision-making process understandable. Our intention is to use this **supervised learning algorithm** as our **model** to make the final prediction of a show to watch.

2. Next, we need to know what kinds of shows each group likes: parents, kids, and grandparents.
Figure 10.9: Machine Learning Model details for the initial task of recommending shows:

1. Pick an algorithm relevant for both the stakeholders’ needs and the task
2. Train the algorithm based on the kinds of shows they liked in the past
3. Did that group care about the setting, the particular device, or the app?
4. Did that group like the last suggestion for that setting?
5. Now we can finally predict a show for tomorrow night!

The different groups might like different kinds of shows; maybe they like a show based on the genre (e.g., cartoon) or runtime (e.g., 50mins), etc. We gather all the data about those shows into our initial dataset, with each show as a row and all the details of the different characteristics of the shows, like genre, runtime, etc., as columns. We then split the dataset into both a training dataset and a testing dataset, as well as an optional validation dataset.

- The columns are the features, or attributes, for our model and will be used to tune the parameters, the internal configuration variables of our model that will be estimated from our training data. The features are the input to our model and the number of features determine the dimensionality of our model. We’ll use the training dataset to set these parameters for our model, and then evaluate our model’s performance using the appropriate performance measures or statistical metrics on the results.

3. Then, we need to see if tomorrow will be a school night and also determine which device we’ll use to watch the show.
   - On school nights, we tend to favour what the kids want; on holidays, we don’t mind seeing what the grandparents might want; on weekends, the wife wins.
   - We will also check if people like one kind of device or app over another: e.g., everyone
hates the ads on Hulu and would rather watch it on FireTV or TiVo but TiVo has a more limited selection

- These are our hyperparameters: they have nothing to do with the show itself but are part of our model (who gets more weight depends on the night) or architecture (the device and app we’ll use to watch the show).

- We can use the things we watched on similar nights over the last couple of months on all the devices to figure out who was watching what to get the preferences for those nights and those devices/apps.

- We would adjust these hyperparameters with our validation dataset.

4. Next, we need to use another part of the recent History (on the device we finally selected) to see which genre won on which night (did cartoons win on weekday nights?). We’ll use this from our testing dataset and we’ll evaluate the results with our performance measures or statistical metrics to decide on our final model.

5. Finally, we can get a list of the new shows that are available tomorrow night (this is our unseen dataset) and then apply our final model to make the final selection based on the parameters (day of week, holiday or not, who’s at the house, etc.) that we determined earlier → This will be our final prediction for the show and setting for tomorrow night’s family movie night.

This generalized approach for any supervised learning task is shown in Figure 10.10.

### 10.2.8 Unsupervised Learning via k-Means

Besides the supervised learning approach we’ve seen with our classification model so far, one of the most frequently used techniques when exploring a new dataset is to use clustering, a form of unsupervised learning. Unsupervised learning techniques don’t need any labelled data and skip the training phase entirely!

Clustering is a simple technique that tries to put sets of objects that are similar into the same group; these groups are the clusters. The approach used for k-Means clustering is to minimize the sum of squared distances from each point of the cluster to the center point, the mean, of the cluster. It iterates between first assigning each data point to its nearest centroid by minimizing some distance measure and then re-calculating the center of the cluster by taking the mean of all data points (the mean in k-Means) assigned to that cluster.

We can utilize this unsupervised learning approach on our Family Movie Night problem and dataset by adding this code to the bottom of our current program so far:

```python
### Do the Clustering (unsupervised so no training)
from sklearn.cluster import KMeans # kMeans classifier from scikit-learn
clf = KMeans(n_clusters=2) # Declare model with 2 clusters (via prior knowledge)
model = clf.fit(X) # Fit the model (no training)
print("\nCluster Centers:\n", model.cluster_centers_) # View cluster centers

### Prediction: All Observations and New Observation
y_pred = model.predict(X) # Predict all observation labels
print("\nPredict labels for all shows:\n", y_pred)
```

Those 8 lines are the entirety of the code we need in order to implement the k-Means algorithm, run it on our dataset, and compute the performance measure results for the Family Movie Night! In the above code, we import the KMeans object and use it to fit() the data, X. Note we fit it on the
Figure 10.10: Supervised learning approach for a general predictive model:

1. Start by picking a learning algorithm that’s appropriate for the task at hand
2. Use the Training Dataset to learn the algorithm’s parameter values
3. Use the Validation Dataset to tune the algorithm’s hyperparameters and architecture
   The resulting algorithm is the tentative model
4. Use the Testing Dataset to gauge the tentative model’s performance
   If no changes are needed, this is the final model
5. Now you can apply the final model to unseen data to make your final prediction for that task

entire dataset without breaking it up into the training/testing datasets using train_test_split() as we did in our decision tree code above. We can view the clusters the k-Means model creates via model.cluster_centers_, which will result in the following output:

Cluster Centers:
[[5.00000000e-01 1.98295455e+03 3.44545455e+01]
 [4.82142857e-01 1.98237500e+03 9.08392857e+01]]

Predict ALL labels:
[0 0 0 1 1 1 0 1 1 1 0 0 1 0 1 0 1 1 1 0 1 0 0 0 0 1 0 1 0 0 1 1 0 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 0 0 0 1 1 1 1 1 1 1 0 1 1 0 1 1 1 0]

Here, we see that there are two clusters (two rows) with the center-point of each cluster, its centroid, being determined by three values, one for each of our predictor variables: the cartoon, year, and
### Load libraries

```python
from sklearn import datasets  # Various datasets in scikit-learn
from sklearn.cluster import KMeans  # kMeans classifier from scikit-learn
import matplotlib.pyplot as plt  # Plotting library for graphs
import pandas as pd  # Data processing library
```

### Get Iris Data

```python
iris = datasets.load_iris()  # Load the dataset
X = iris.data  # Features
y = iris.target  # Labels/Targets
```

### Iris Exploratory Data Analysis (EDA)

```python
iris_df = pd.DataFrame(iris.data, columns = iris.feature_names)  # Summary Statistics
print("\nSummary:\n", iris_df.describe())
print("\nFeatures:\n", iris.feature_names)  # Feature Names
print("\nLabel Names:\n", iris.target_names)  # Target Names
print("\nActual Labels:\n", y)  # Print the class labels

# Slice the Dataset and Plot it as a scatter plot
x_axis = iris.data[:, 0]  # Sepal Length
y_axis = iris.data[:, 2]  # Petal Length
plt.scatter(x_axis, y_axis, c=iris.target)  # Display the scatter plot
plt.show()
```

### Do the Clustering (unsupervised so no training)

```python
clf = KMeans(n_clusters=3)  # Declare the model (make 3 clusters using prior knowledge)
model = clf.fit(X)  # Fit the model (no training)
print("\nCluster Centers:\n", model.cluster_centers_)  # View cluster centers
```

### Prediction: All Observations and New Observation

```python
y_pred = model.predict(iris.data)  # Predict all observation labels
new_pred = model.predict([[3.1, 1.3, 5.7, 7.5]])  # Predict new observation label
print("\nPredict ALL labels:\n", y_pred)
print("\nPredict New label:\n", new_pred)
```

Listing 10.2: k-Means clustering on the Iris Dataset.

**k-Means on the Iris Dataset**

But let’s get a better handle on the details of how easy it is to use clustering on a new dataset by doing some **Exploratory Data Analysis (EDA)** on the well-known Iris dataset and clustering the samples.¹⁴ EDA usually involves doing both statistical analysis and visualization of the dataset in order to characterize the dataset and get a more intuitive sense of it before we formally model it.

The Iris dataset consists of 150 **samples**, where each sample is an Iris flower. Four measurements are taken for each flower and these measurements will be our **features**: sepal length, sepal width, petal length, and petal width. There are three kinds of Iris flowers and so these are our **class labels**: setosa, versicolor, and virginica.

We start by loading the dataset from the scikit-learn package on Line 8 of Listing 10.2 and then runtime features. Not bad for about 8 lines of code!

Chapter 10. Machine Learning and Data Science

Figure 10.11: Scatterplot graph of the Iris dataset showing three different classes: setosa, versicolor, and virginica.

separating out the Features and Labels on Lines 9 - 10. Then, in Lines 12 - 22, we do some EDA; in this case, we load the numpy data matrix (iris.data) into a pandas dataframe object and then use the dataframe’s describe() method to display the statistical summary as shown below.

We also print out the feature names (these are the flower measurements we used as our features or columns) as well as the label names for the three classes. Then, on line 17, we print out the actual labels for all 150 samples; each of the 150 flowers has a label of 0, 1, or 2 depending on whether it’s a setosa, versicolor, or virginica. After that, we use the matplotlib library’s pyplot object to continue our EDA and create a scatterplot of the sepal length vs petal length, as shown in Figure 10.11.

We then get back to the analysis and do the actual clustering, creating the clf classifier object. We fit() the data to the classifier to create our final model. We then view the clusters formed by the model on Line 27. The clusters are shown as a 3 x 4 dimensional matrix. This is because we selected 3 clusters (the 3 rows) and each row is made up of 4 dimensions, 1 column for each of the 4 features (sepal length, sepal width, petal length, and petal width).

Finally, in Lines 29 - 33, we predict the labels for each of the samples as per the k-Means algorithm (on Line 30) and for a new, fake flower observation that would form the 151’t sample (on line 31). We finally print these values and the entirety of the results from running the code in Listing 10.2 is below:

<table>
<thead>
<tr>
<th>Summary:</th>
<th>sepal length</th>
<th>sepal width</th>
<th>petal length</th>
<th>petal width</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>150.000000</td>
<td>150.000000</td>
<td>150.000000</td>
<td>150.000000</td>
</tr>
<tr>
<td>mean</td>
<td>5.843333</td>
<td>3.057333</td>
<td>3.758000</td>
<td>1.199333</td>
</tr>
<tr>
<td>std</td>
<td>0.828066</td>
<td>0.435866</td>
<td>1.765298</td>
<td>0.762238</td>
</tr>
<tr>
<td>min</td>
<td>4.300000</td>
<td>2.000000</td>
<td>1.000000</td>
<td>0.100000</td>
</tr>
<tr>
<td>25%</td>
<td>5.100000</td>
<td>2.800000</td>
<td>1.600000</td>
<td>0.300000</td>
</tr>
<tr>
<td>50%</td>
<td>5.800000</td>
<td>3.000000</td>
<td>4.350000</td>
<td>1.300000</td>
</tr>
<tr>
<td>75%</td>
<td>6.400000</td>
<td>3.300000</td>
<td>5.100000</td>
<td>1.800000</td>
</tr>
<tr>
<td>max</td>
<td>7.900000</td>
<td>4.400000</td>
<td>6.900000</td>
<td>2.500000</td>
</tr>
</tbody>
</table>

Features:
['sepal length', 'sepal width', 'petal length', 'petal width']

Label Names:
['setosa', 'versicolor', 'virginica']
10.3 Elements of Machine Learning

The rise of machine learning is connected to the increase in the amount of digital data that the human species has started producing of late. Man has been around for about 2 million years; Homo Sapiens for probably close to 300,000 years; behavioural modernity was achieved about 50,000 years ago and written history has been around for about 5,500 years. According to some estimates, in the year 2009 CE alone, human beings have generated more data than in all these previous years combined. In fact, in each subsequent year since then, the amount of data we’ve generated has only increased!

The amount of data we collect from social, scientific, and other organizational systems seems to be increasing exponentially. Data from the various sub-fields of physics and computation is huge but even biological data collection is quite daunting: for example, the human brain consists of around 100 billion neurons and about 100 trillion synapses. Human beings, on average, consist of about 30
trillion cells with about 35 trillion more cells that are made up of microbial hangers-on. Analyzing these connectomes and biological systems generates an enormous amount of data.

This growth in data is popularly referred to as **big data**. Although the *volume*, or size, of data is one aspect of big data, other characteristics like the *velocity* of data (how fast it’s increasing) and the *variety* of data (the different formats and media of the data) are also cause for concern.

This kind of data comes in many forms: it can be highly **structured**, like typical database tables or Excel spreadsheets; it can be **un-structured**, like emails, documents, images, etc. – anything that doesn’t neatly fit into a database; or it can be **semi-structured**, living somewhere between the two, like XML or JSON files. It’s important to note that un-structured data, in this sense, does not mean the data itself doesn’t have some internal organization; instead, data is called un-structured in the sense that normal data mining tools cannot easily parse it. Most data generated today is either semi-structured or un-structured, as shown in Figure 10.12.

This is relevant for us as the type of data we have will determine the kinds of analyses and visualizations we can make. Whenever we deal with different kinds of data, we need to consider:

- What kinds of questions can we ask using this kind of data?
- What types of analytics can be carried out on this data?
- What data sources are available?
- What types of data will we have (structured, semi-structured, un-structured)?
- Who uses or consumes this type of data?

As you might imagine, analyzing this amount of data is quite challenging and studying these kinds of data-intensive problems is almost prohibitive using traditional tools. In these cases, statistical machine learning techniques have proved especially effective in tackling such big data problems.
10.3 Elements of Machine Learning

Figure 10.13: How Artificial Intelligence (AI) and Machine Learning (ML) are related.

10.3.1 Machine Learning Writ Large

Machine learning is thus used in contexts that involve large-scale complex systems, multi-dimensional datasets, or computational solutions that are simply too complex to be written manually. Complex systems like the billions of nodes that make up the internet, sensor networks from multi-modal sensors, genomics, proteomics, connectomics, etc., have significant analytical complications.

For such problems, having machines (or, more accurately, programs and algorithms) that can learn is invaluable. Thus, we don’t use machine learning when we need to calculate the payroll for a company but we do use it when we’re dealing with situations where:

- Big Data challenges loom large (as outlined above)
- The requisite human expertise isn’t readily available (e.g., exploring new planets)
- We cannot explain the mechanism behind existing human expertise (e.g., motion recognition)
- The actual solution changes in time (e.g., how routing occurs over a computer network)
- The final solution must be adapted for particular cases (e.g., in user biometrics)

In these kinds of situations, machine learning is indispensable. For example, it is easier to write a program that learns to play chess by self-play instead of fully characterizing the expertise of chess masters as a rule-based program. Indeed, machine learning is used successfully in a wide spectrum of applications, including natural language processing, search engines, medical diagnosis, detecting credit card fraud, stock market analysis, bio-informatics, classifying DNA sequences, speech and handwriting recognition, object recognition in computer vision, playing games, robot locomotion, etc.

As we saw earlier, machine learning started off as an off-shoot of artificial intelligence, which not only gave birth to machine learning but is still concerned with designing approaches to support learning and perhaps replicate human intelligence in the future. This relationship is shown in Figure 10.13, where it also shows some of the more powerful sub-fields of machine learning, including deep learning, a variant of artificial neural networks (ANNs) which does hierarchical feature learning, or representation learning, using neural networks with considerably more layers than is typical for simpler neural networks. We’ll learn more about this especially effective approach to tackle large, unstructured datasets in just a bit.

Machine learning is also intimately related to and borrows from several fields, including: statistics, which is usually concerned with inference from a sample; data mining, which develops techniques for searching through large volumes of data; and, in general, computer science, which, in broad terms, deals with developing efficient algorithms and complex models.
10.3.2 ML Definition: Tasks

Pragmatically speaking, there are usually eight kinds of questions a machine learning algorithm can answer or tasks that it can do. Each of these eight questions is usually addressed using a different machine learning algorithm. These eight common questions and their corresponding approaches are shown below and in Figure 10.14:

1. Find if there are any unusual data points: is this weird? → Anomaly Detection
2. Discover how this is organized or structured? → Clustering and Dimensionality Reduction
3. Predict a number, like how much or how many? → Regression
4. Predict if something is A or B? → Classification
5. Predict if this is A or B or C or ...? → Multi-Class Classification
6. Find the best way to make your next move; what should you do next? → Reinforcement Learning
7. Is the data super-complicated and hard to understand? → Deep Learning
8. Is accuracy quality really important? → Ensemble Methods

There are also additional questions that are often asked like “Is this the best?”, which can be answered using optimization or some such approach. In fact, most machine learning is about optimization: either an explicit reward as in reinforcement learning or a score or loss function used in estimating parameters (e.g., the weights in a neural network) or learning structures (e.g., the directed tree structure...
created in decision trees). These algorithms, or models, learn based on the data we give them and we can then measure how much they’ve learned using some performance metric.

There are many machine learning methods but they’re differentiated from the previous computational solutions we have usually developed in computer science. In traditional computational solutions, the input data and the program for a specific task together determine the output; in the machine learning paradigm, this is usually turned on its head and instead the input data and the output help determine the structure of the computational model or program we develop for a task, as shown in Figure 10.15.

10.3.3 ML Definition: Models

A computer program, or algorithm, is based on some model. So we can group different machine learning algorithms by the category which characterizes those algorithms or the tasks those machine learning algorithms can perform. We also need to specify the metrics, or performance measures, to gauge the success of those algorithms, as we’ll do in Section 10.3.4.

These main categories of machine learning are shown in Figure 10.16. We can further categorize these different machine learning algorithms in many ways. We can group them by learning style or by similarity in the form and function or by the model type. If we categorize them by learning style, we might end up with a listing like this, which sometimes also adds Reinforcement Learning, Representation Learning (Feature Learning), and Combined Models to the list:

- Supervised
- Un-supervised
- Semi-supervised

The two most ubiquitous learning styles are Supervised Learning and Unsupervised Learning; the
Figure 10.17: Elements of the classical machine learning components of supervised and unsupervised learning.

various kinds of supervised and unsupervised learning are shown in Figure 10.17. The third kind of learning style, Semi-Supervised Learning, lies somewhat intermediate to Supervised and Unsupervised Learning. Semi-supervised algorithms use a combination of labelled and unlabelled data for their training like in semi-supervised SVMs.

One of the easiest ways to categorize different models is to group them based on the input to or the output from that algorithm. For example, we can see what kinds of input we might have:

1. If the input data is labelled, the problem is one of supervised learning
2. If the input data is not labelled, we can find the structure in this kind of unsupervised learning problem
3. If the algorithm needs to concentrate less on the data and more on the environment, continuously getting input by interacting with the environment in order to optimize an objective function, this kind of problem would involve Reinforcement Learning (RL).\(^{15}\)

We could also choose to categorize based on the kind of output we’ll produce:

1. If our algorithm needs to output a number, the problem is called regression
2. If our algorithm needs to output the class to which something belongs, it’s called classification
3. If our algorithm needs to find any outliers, it’s called anomaly detection

The complexity of the input data can introduce additional complications, as well. For example, if the number of features in a dataset is huge, you might need to employ some method for reducing the number of dimensions in your dataset. Why do we need this kind of dimensionality reduction?\(^\)\(^{15}\)

One reason is the Curse of Dimensionality, which is concerned with the issues that arise with high-dimensional problems and which aren’t seen in lower-dimensional problems. The number of dimensions is related to the number of features, or columns, in our datasets. As the number of features increases, the number of samples needed to well-represent all combinations of feature values also increases commensurately. But if we increase the number of features, we also increase the complexity of the model. As the model becomes more complex, the chances of overfitting increases and when we apply it to new data, it will give poor results as it won’t generalize well. Thus, dimensionality reduction helps reduce this risk of overfitting which accompanies the Curse of Dimensionality.

Dealing with complex models also makes it harder to understand how the machine actually learned what it did. For example, ensemble methods, which blend multiple models, and highly complex models with millions of parameters, like neural networks, are effective but are often hard to understand and so we just treat them as an opaque black box which ineffably churns out answers. If the answers and their accuracy are more important than understandability, then you might very well pick these algorithms for a certain problem.

\(^{15}\)In RL, the machine keeps track of the state of the environment, represented by a feature vector. Like a Turing Machine, it associates different actions with different states, and each action can elicit a reward and move the machine to a different state. The machine then formulates a policy that outputs the optimal action, the one that maximizes the reward.
10.3 Elements of Machine Learning

10.3.4 ML Definition: Performance Measures

In general, we determine an appropriate machine learning algorithm based on the desired performance measures and available resources with consideration for the data and computational complexity, including training time and model complexity. We would normally create a test harness for any complete machine learning solution that utilizes training data, testing data, and validation data along with the performance measures. We’ll meet some of these performance measures in detail later but, for now, let’s see just a brief overview of the kinds of measures we might utilize:

1. We can use measures like a confusion matrix, heat map, lift charts, etc. to visualize and evaluate the results.
2. We can evaluate the statistical significance of the proposed solution, like r-square, f statistic, p-value, etc., as we’ll see in Section 10.5.
3. We can employ standard statistical evaluation metrics like precision, recall, accuracy, F1, ROC curve, etc., as seen in Section 10.5.7; we often need precision, recall, and their various combinations as accuracy is insufficient for imbalanced classification problems.

Making Reliable Predictions

The process for making reliable predictions requires rigorous testing and validation of your learning model. You should only use it on unknown data to make your final prediction once you’re confident in its reliability.

At a minimum, as you develop your model, you should:
- Test it on a toy dataset or a simulated dataset
- Validate it on a known dataset
- Only then use it for predictions on an unknown dataset

10.3.5 Model Building and Assessment

In order to properly assess our model, we need to construct two hold-out sets (that aren’t used to assess the algorithm) called the training and validation datasets, as well as a testing dataset; all three should follow the same probability distribution as the underlying problem space. For small datasets, the split is usually 70/15/15 and, for large datasets, can be 90/5/5 or even 95% for training, if needed, since often more data helps. Sometimes, though, model complexity needs to be increased to manage the bias-variance tradeoff in classification tasks. This kind of N-fold Cross-Validation with Bias-Variance Tradeoff analysis in supervised learning can help set the algorithm’s configuration to avoid overtraining.

The main idea of machine learning is to first select an algorithm or multiple candidate algorithms to address some problem. Each algorithm will learn certain parameters using the training dataset. Once the parameters are set for the learning algorithm, we will have determined our tentative model.

---

16In a 2-class problem, if the positive class greatly outnumber the negative class, as with diseases, terrorists, etc., we encounter the accuracy paradox where accuracy is no longer an effective measure.
Parameters are the configuration variables that are internal to the algorithm and are usually estimated or learned from the data rather than set manually; once learned, they’re often stored as part of the learned model. If a model has a fixed number of parameters, the model is called parametric; if the number of parameters in a model is not fixed, that model is called non-parametric. Parameters in a model are things like the weights and biases in a neural network, the support vectors in SVMs, the coefficients in linear regression, etc.

Then, the validation dataset is used to tune the hyperparameters and choose the best-performing validation dataset. A model’s hyperparameters are external to the algorithm and cannot be estimated or learned from the data. The values of the hyperparameters can help estimate or learn the values of the model’s parameters. They are often set manually but can also be set using heuristics or tuned via techniques like assessing validation curves, doing grid search, etc. Tuning the algorithm to come up with a good predictive model for a particular task involves tuning the hyperparameters in order to find the model parameters that give you the best predictive capability. We can see this process for a general machine learning task in Figure 10.19 where the Testing Metrics are the Performance Measures from Section 10.3.4.

Examples of hyperparameters are things like the number of trees in a random forest, the regularization parameter \( c \) in SVMs, which determines the tradeoff between testing and training error, the sigma smoothing parameter in the RBK of an SVM, various parameters (number of layers, size of each layer, number of connections in neural networks, the learning rate) in neural networks. Finding the best values for these hyperparameters can be cast as a search problem so typical approaches are search methods like grid search, random search, etc.

If the validation set is very small, we can use cross-validation to construct a simulated validation set. We can do 5-fold cross validation on the training set to create both a training and validation set in each fold. The test dataset is used to assess the final chosen model’s performance to see how well it generalizes beyond the training/validation tuning. We can also do things like use grid search, as we saw in Section 10.2.4.

We often need to do feature selection, where we determine which variables don’t add any new information or are correlated with others, and feature engineering, in which we develop new features (feature generation or feature extraction) or transform existing features (feature transformation and aggregation). This can be facilitated by gauging the information content of features using measures like information gain, entropy, gini impurity, chi-square, etc., as in Section 10.6.2 or automatically via representation learning or feature learning, as in neural networks.

Figure 10.19: The machine learning paradigm visualized with data: training data helps train the model’s parameters which are then tweaked depending on how it performs on the testing data. Hyperparameters are also tuned separately.
10.4 Data Science and Data Analytics

This kind of approach to making new predictions is helpful not only in a home application like the Movie Night example but also in applications to organizations, as well, especially for applications that are mission critical or safety critical.\(^{17}\) No wonder then that there is an interplay between the different fields that create and utilize machine learning algorithms.

The three main areas that exhibit this interplay can be categorized as:

- **Computer Science** emphasizes theoretical and empirical approaches to manipulating data via computation or algorithmic processes.
- **Informatics** deals with a broader study of data and its manipulation in information processes and systems, including social and cognitive models.
- **Data Science** tackles structured and unstructured data and uses both computational methods and cognitive or social methods, especially when visualizing complicated data analytics.

All three of these approaches rely upon the usage and manipulation of data, especially in regards to making decisions. The central idea behind a data-driven decision-making approach is summarized by these three components:

1. Analyze historical trends
2. Forecast future outcomes
3. Assist decision making with actual quantitative predictions

These three components can be thought of as the result of three kinds of activities: Exploratory Data Analysis, Data Analytics, and Data Science. When these kinds of activities are applied to business problems, the field of **Business Intelligence/Business Analytics (BI/BA)** can map its components to these three activities as: Descriptive Analytics (maps to Exploratory Data Analysis), Predictive Analytics (maps to Data Analytics), and Prescriptive Analytics (maps to Data Science).

Let’s examine each of these, along with their corresponding terms, next:

- **Exploratory Data Analysis (EDA)** mainly asks, “What happened in the past?”. It thus analyzes historical data and, in the BI/BA paradigm, is called **Descriptive Analytics**. Descriptive analytics tells us what happened; this involves using data to understand past and current performance to make informed decisions. This is sometimes said to include **Planning Analytics**, where the plan for the forecasted goals is established, including planning and budgeting concerns.

- **Data Analysis** tries to answer, “What could happen in the future?” In the BI/BA paradigm, this is called **Predictive Analytics** and, sometimes, also called **Business Informatics** when it addresses organizational behaviour/science and systems, as well. Usually, this involves building a **machine learning model** to **predict** an outcome. Since predictive analytics deals with what could happen, it will involve using historical data to predict future outcomes. This sometimes also includes **Diagnostic Analytics**, which looks at why things happened in the past using data mining and data discovery to also establish correlations, causal relationships, and patterns in historical data. The focus is mainly on processing **existing** data sets and performing statistical analyses on them.

  - Data analysis involves using machine learning algorithms to derive insights and support decision-making. This also requires extracting, cleaning, and munging data in order to transform it into a useable format for modeling that data to support decision-making and

\(^{17}\)There are 4 kinds of critical systems (mission, business, safety, and security), with machine learning essential for each.
confirming or informing conclusions. Cleaning and munging the data involves dealing with missing fields and improper values, as well as converting raw files to the correct format and structure.

- **Data Science** answers the question, “What should we do next?” It uses predictions and helps decide what to do next and how to make it happen. In the BI/BA paradigm, it is called **Prescriptive Analytics**.

  In essence, prescriptive analytics looks at what you can do about what’s about to happen; this involves a consideration of what we can do about what’s going to happen and how we can make it happen; prescriptive analytics sometimes concentrates on optimization and tries to identify the solution that optimizes a certain objective.

  - Data science requires the user to be able to innovate algorithms, find unique, additional data sources, and have deep domain knowledge. It uses ideas from computational science and information systems in order to gain actionable insights and make predictions from both structured and un-structured data about what to do or which decisions to make. This is why a data scientist should also have some domain knowledge in addition to the ability to create machine learning models and identifying ways to predict trends and connections by exploring novel, perhaps disconnected, data sources.

### 10.4.1 Business Intelligence/Business Analytics

Each of these three layers, EDA (Descriptive Analytics), Data Analysis (Predictive Analytics/Business Informatics), and Data Science (Prescriptive Analytics), builds upon or incorporates the previous layer. In addition, Prescriptive Analytics usually incorporates new data sources for supporting decision making while Descriptive and Predictive Analytics use previously available historical data sources.

These three different kinds of analytics usually fall under the rubric of **Business Intelligence (BI)**. Although these definitions aren’t rigorous, in general, they encompass the idea of applying data analysis and data science to business data at the enterprise level. In fact, Business Intelligence is often used interchangeably with the term **Business Analytics (BA)**. Sometimes, though, Business Intelligence is defined as covering only up to Descriptive Analytics while Business Analytics is then defined to cover Predictive Analytics and Prescriptive Analytics.

As such, in the BI/BA paradigm, while descriptive analytics looks at what has happened from a statistical perspective, predictive analytics looks at what could be and also makes relevant predictions based on that data. Prescriptive analytics not only foresees these identified trends and predictions but also focuses on complete solution strategies relevant to the organization, often using optimization to identify the best alternatives, identifying new data sources, and prescribing what actions need to be taken in order to meet these trends. Prescriptive Analytics thus adds a causal part, not just a correlation, somewhat akin to the difference between kinematics and dynamics in physics. This also focuses on suggesting a course of action and takes on the role of **data-driven decision support systems**, helping transform data into actionable insights to inform an organization’s strategic and tactical decision making.

A somewhat surprising heuristic maps the BI/BA activities to the Army Physical Fitness Test (APFT): in this approach, you can make a quick decision by following three simple rules; those three rules can be mapped to our three Business Intelligence categories as follows:

---

18 So far, there are no standardized definitions of these terms derived from first principles. There are some standardization efforts that are more popular than others but none that is universally accepted so far. So none of these terms should be taken as hard “definitions” as the terms themselves are somewhat ambiguous, overlap with each other, and are still being refined.
10.4 Data Science and Data Analytics

1. Understand the overall situation or mission intent: Descriptive Analytics
2. Enumerate the possible outcomes: Predictive Analytics
3. Create and execute the plan: Prescriptive Analytics

Some of the factors that affect a choice of specific machine learning algorithm have more to do with the business constraints than optimization and technical characteristics of the particular algorithm. As such, there is often an overlap of terminology between Data Science and Business Intelligence/Business Analytics.

We can contextualize all three aspects of business intelligence/business analytics with a financial example based around balance sheets. In terms of **Exploratory Data Analysis** (or **Descriptive Analytics**), we might identify areas of influence on the bottom line by examining the underlying data by drawing charts for budgets, sales, revenues, costs, etc. This allows us to get a rigorous picture of the balance sheet which provides plain fact information about an organization’s finances for some period of time, thus giving you insight into the facts of what happened to the organization, by drilling down to see which exact resources are most profitable.

In terms of **Data Analysis** (or **Predictive Analytics**) of the balance sheet, we might combine the information from multiple balance sheets and compare that over a period of time to gain information about where the finances may trend, such as surges in revenue during certain periods or decline in market profits. Finally, **Data Science** (or **Prescriptive Analytics**) would use these identified trends, combined with analysis of new supporting datasets, to propose several options to achieve the desired outcome, usually along the lines of maximizing profit while minimizing costs.

10.4.2 The Jataka Analytics

Following the example of the Jataka Tales, suppose a troop of monkeys in an Indian jungle go on their weekly foraging trip to find some yummy bananas. They contract a free-lancing parrot to gather data detailing all the places it discovered bananas last year as it flew around in the vicinity of their water hole. The parrot summarizes its findings in a report and this report would be an example of **Descriptive Analytics**. In practice, rather than the parrot, such historical patterns are accumulated and analyzed constantly by companies like Google, who gather all kinds of data on your web and social...
media activity and then sell it to data houses that accumulate these historical patterns.

Suppose, after analyzing the parrot’s historical data, our troop of monkeys predicts that they would be able to find bananas on the east side of their regular water hole. This insight would be an example of **Predictive Analytics**. In practice, this process results in hypotheses about how an organization might maximize some element of their practice, usually the profit for a commercial enterprise. For example, suppose a company wanted to create a new product; they might consider questions like: does it align with what the markets want and need? Will it reach the right target market? Does the public want it but just don’t know what it is yet? The answers to these questions would be the hypotheses that formulate our predictions.

Finally, our enterprising troop of monkeys uses this data, along with their hypotheses about the best places to forage bananas, to modify the routes they take for their weekly foraging trip. Their goal is to maximize the number of bananas they find and, as they adjust their routes, they also record data about all the banana trees along these new routes so that they can further optimize their routes to increase the number of bananas and also to collect them before other banana foragers. These plans and the new datasets they collected are the result of their **Prescriptive Analytics**. Companies, in fact, often utilize such prescriptive analytics to strategize about approaches to make their organizations more efficient and capitalize on the information created and discovered by the organization. This might help them do things like optimizing their supply chain, targeting new products to appropriate markets, etc.

---

**Recap of Analytics**

1. Descriptive analytics is what happened; this involves using data to understand past and current performance to make informed decisions
2. Predictive analytics is what could happen; this involves using historical data to predict future outcomes
3. Prescriptive analytics is looking at what you can do about what’s about to happen; this involves a consideration of what can be done about what’s going to happen

Thus, the goal of data science is to make predictions: precise, quantitative predictions that will be useful in any organization or organizational context for decision-making. **Decision analysis** uses ideas from computer science, psychology, economics, management science, and decision science to deal with uncertainty and complex, sometimes competing, objectives in order to evaluate possible actions.

---

19The Jataka Tales tell various parables, often humorous and insightful, about the former lives of Gautama Buddha, especially those that were in animal forms. Jataka is derived from the Sanskrit word, Jaataak (“birth”), and most of these stories were written between 300 BCE and 400 CE in Pali. Some of the most entertaining, and enlightening, stories center around a troop of monkeys, hence the simian example above. The ever-popular Amar Chitra Katha series of comics also devoted a special series to the Jataka Tales.
10.4.3 Is Data Science a Science?

Before we tackle the idea of whether Data Science is a science or not, something that doesn’t seem to have a definitive answer, let’s step back and look at the idea of proof. This is a word that is overused quite frequently as there are many different kinds of proof: for example, there are scientific proofs, legal proofs, and mathematical proofs.

In mathematics, a proof is an inferential argument that shows a statement is true as supported by axioms, definitions, theorems, and postulates. Mathematicians normally use deductive reasoning to show that the premises, also called statements, in a proof are true in a direct proof. A direct proof is one that shows a given statement is always true and the proof is usually written in a symbolic language. In an indirect proof, mathematicians usually employ proof by contradiction, where they assume the opposite statement is true and eventually reach a contradiction showing the assumption is false.

In law, legal proof is the process of establishing a fact by using evidence. In science, we might call this a validation of some theory as that usually also takes the form of an argument where you present a series of premises in support of some conclusion. Similarly to law, proof in science is usually limited to proof of facts in the sense of using data to establish the validity of facts. This is discussed at length by D.H. Kaye in [18] which shows that the use of quantitative observation-statements provide evidence to prove or, as we’d say in science, show the validity of, facts.

In science, an inherently inductive enterprise, we cannot prove any hypothesis to be true as that would require an infinite number of observations so the best we can hope to do is use inductive reasoning as the basis of our generalization and hold it to be provisionally true. As noted by Lee Loevinger about Karl Popper, “In this view, which is fairly widely accepted, an hypothesis can be falsified, or disproved, but cannot be verified, or proved.” Once it’s validated extensively and consistently, and we deem it to be sufficiently substantiated, we then call it a theory. So we could, in some sense, say that legal arguments use evidence to show the validity of a theory whereas science uses data to falsify a theory.

For example, following [18], collecting quantifiable data of the intensity and polarization of radiation at various frequencies from a radio telescope pointed at the Crab Nebula is the evidence that shows (in law, proves) something in the direction of the Crab Nebula is a radio source, the fact. Such facts can be deduced or induced from statements of observations, the evidence. Thus, a fact is based on some repeatable observation or measurement that is generally agreed upon to recur with the same value or in the same way under the same kinds of circumstances.

These facts are then used to inductively reason about a hypothesis or model of the system being studied. The predictions made by that model are further verified and, when enough predictions are verified independently, the hypothesis, or set of hypotheses, is considered sufficiently validated to be called a theory.

This process, this scientific method, is exactly what we employ when we utilize our machine learning models, like Hypothesis Testing in Section 10.5.1 or Decision Trees in Section 10.7, within a Data Science framework and use data to iteratively test and improve our models. I might further argue, following Feynman’s formulation, that as long as you’re using a systematic model to make predictions and then testing those predictions with data and using those results to validate or improve your model iteratively, you’re doing science.

Applying these scientific models to specific business problems without iteratively changing or further developing those models results in engineering and technology principles. As such, I might be

---

20Inductive at least to the extent that thinkers like Richard Feynman and Karl Popper would find it to be so, Feynman in his exposition on the Key to Science and Popper in his formulation of conjecture and criticism.

21Where, once again, we mean any organization when we say business.
inclined to categorize Data Analysis as an engineering discipline and Exploratory Data Analysis as a technological application.

10.5 Bayesian Inference and Hypothesis Testing

Now that we have some of this mathematical machinery under our belt, let’s build upon it by looking at an essential tool for validating scientific inquiries. We know that all of science is built upon the foundation of falsifying hypotheses so looking at how this is done is central to any scientific endeavour.

10.5.1 Null Hypothesis Significance Testing

Suppose someone takes a pregnancy test to check if they’re pregnant. The test, which looks for hCG levels in urine, reports a negative result, indicating the hCG levels are below the test’s threshold for establishing pregnancy. Since the result was negative, they feel safe going to the dentist to take some X-Rays. But what if the test was wrong and they were actually pregnant? As you might surmise, a false negative like this might be devastating!

These kinds of situations often arise in Null Hypothesis Significance Testing (NHST), also known as Hypothesis Testing. NHST is a way to draw a conclusion, or infer, whether some expression is true or not, usually by comparing two datasets or a sample from some dataset.

NHST depends upon a Null Hypothesis, \( H_0 \), the falsifiable expression that’s assumed to be true unless there’s evidence to the contrary. In this case, we might think of it as the fact that we’re not pregnant unless there’s evidence to the contrary.

As a side note, the hypothesis should usually be stated in terms of an independent variable, a feature of the experiment we don’t directly control but that can help predict the value of a dependent variable, where the researcher gets to decide which variable is independent or dependent in a given experiment. The independent, or predictor, variable is usually plotted on the x-axis (abscissa) and is usually the input to some function where the output, or the y-axis (ordinate) value, is the dependent variable.

The hypothesis statement can then be written as an “if ... then” construction, as we saw in our discussion of formal logic, in terms of the independent and dependent variables as such, “If (we do something to the independent variable) then (we expect some result in the dependent variable).” Using the hypothesis statement, you’ll formulate a null hypothesis, which should be a falsifiable expression of a result that follows from the hypothesis statement. Since the null hypothesis is falsifiable, you can end up with the opposite state of the null hypothesis for a result, as well. This opposite expression is termed the Alternative Hypothesis, \( H_1 \) or \( H_A \).

In terms of the person taking the test, perhaps their hypothesis statement was, “If there is a full moon night, then they cannot get pregnant,” where the full moon night is the independent variable and not becoming pregnant is the dependent variable.\(^{22}\) To test this hypothesis statement, they might formulate the null hypothesis they’re not pregnant since there was a full moon last night: \( H_0 = \text{Not_Pregnant} \). This will also give them the alternative hypothesis that they are pregnant, despite the full moon: \( H_1 = H_A = \text{Pregnant} \).

In order to check their pregnancy state, they take a pregnancy test and the result is negative. The result of this pregnancy test is the data we use to assess whether our falsifiable \( H_0 \) is true or not. We can determine the probability, or p-value, of this assertion being true as:

\[
p-value \equiv P(\text{observing this data} \mid H_0 \text{ is true})
\]

\(^{22}\)As with all examples in this book, the situations tend to either be convoluted, unlikely, or, as in this case, both.
Thus, the p-value is the **conditional probability**, or likelihood, of seeing the current data, or data that is *more* extreme than the current data, if the null hypothesis is true. As you can see in the definition above, the p-value assumes that \( H_0 \) is true; the p-value is not the probability of \( H_0 \), or \( H_1 \), being true but rather the likelihood of seeing the current evidence if \( H_0 \) is true.

The goal of NHST is to see if it’s *reasonable* to reject \( H_0 \). The way that’s normally determined is by deciding that a probability lower than a certain level, the *significance level* (\( \alpha \)), indicates the results are too unlikely to be by random chance alone.\(^{23}\) You can use different levels for this significance level but a generally accepted rule of thumb is to reject the null hypothesis if the p-value is below \( \alpha = 0.05 = 5\% \) or, for an even stricter standard, if the p-value is below \( \alpha = 0.01 = 1\% \).

<table>
<thead>
<tr>
<th>Null Hypothesis Significance Testing</th>
<th>consists of three components:</th>
</tr>
</thead>
</table>
| 1. The hypotheses: a Null Hypothesis, \( H_0 \), and one or more Alternative Hypotheses, \( H_A \)
2. The statistical test: often a z-test or t-test, for \( H_0 \)
3. The significance evaluation: a probability, the p-value, is compared to a threshold, \( \alpha \) |

**Hypothesis testing** determines whether the data provides enough evidence to indicate if our initial guess, \( H_0 \), was wrong or if our initial guess was actually right and random chance alone generated the data we see. The significance level, \( \alpha \), is the percentage, or probability, of risk we are willing to take in accepting the null hypothesis even if it is actually wrong. The p-value is the probability of seeing the data and it is calculated for the sample statistics, not the population, by looking it up in a z-table or using statistical software.

If you find the p-value, the probability of seeing the data if \( H_0 \) were true, is below this significance level, \( \alpha \), then you have to reject the null hypothesis (that \( H_0 \) is true) as the relationship in the sample is then unlikely to be by chance alone and likely reflects a relationship in the underlying population. This means that the likelihood of seeing the data you observed is so low that the alternative hypothesis, \( H_A \), must be true.

\(^{23}\) A more formal way to look at this is when you have to analyze some values, or *parameters*, of a large population; because you can’t examine the entire population, you draw a *sample* from the population and measure some *variables* for that sample instead; you might compute some descriptive statistics on those variables, as well. These *sample statistics* on the values of the variables from the sample can help you draw some conclusions about the corresponding values, or parameters, of the underlying population.

The sample statistics can thus be used to estimate the population parameters. This idea of drawing conclusions about a population by computing statistics on small samples drawn from that population is called *inferential statistics*. However, these sample statistics cannot perfectly estimate the corresponding population parameters because of the *random variation* in computing any statistic for multiple samples. This random variation is inherent in the process of drawing samples and is called the *sampling error*, even though it’s nobody’s fault. When you draw different samples from a population, the values of those variables will usually exhibit some random variation by virtue of picking different subsets of the underlying population.

All this means that if you notice some statistical relationship in a sample, it might be from random variation and that statistical relationship might not exist in the underlying population at all! NHST is thus a formal method to check if the statistical relationship in the sample occurred by chance and only reflects sampling error; the idea that the relationship in the sample occurred by chance and is not found in the underlying population is your null hypothesis, \( H_0 \).

Once you formulate the null hypothesis, \( H_0 \) which reflects a sample statistic relationship but no relationship between the parameters in the population, then you have to compute the probability of the sample statistic relationship, or how likely the sample statistic relationship would be if the null hypothesis were true. This probability is the p-value.

If the probability, or p-value, is lower than the significance level that means the sample’s statistical relationship would be extremely unlikely, then we reject the null hypothesis and accept the alternative hypothesis, \( H_1 \) or \( H_A \), which says that the statistical relationship in the sample reflects a similar statistical relationship in the underlying population, as well. NHST on the sample estimates can also be used to create a *confidence interval*, a range of values derived from the sample statistics that is likely to contain the actual value of the population parameter.
Chapter 10. Machine Learning and Data Science

In our running example, if the probability, or p-value, of the pregnancy test showing positive given that we’re not pregnant is lower than the significance level, or α = 0.01 = 1%, we will reject the null hypothesis (H₀ = Not_Pregnant) and start printing baby shower announcements instead. If that probability is higher than 0.01 (or 1%), though, then we will not reject H₀ and instead grab a drink and go to the dentist as planned. So how do we determine this probability?

We employ some kind of a statistical test like the Student’s t-Test or Welch’s t-Test, or, in the case where you know the actual population standard deviation, the one-sample z-Test. Suppose we decide on the one-sample z-Test; the next step is to compute the z-score associated with the significance level, α, we chose. In addition, we have to calculate the z-score for our problem. Then, if our problem z-score is higher than the z-score associated with α, we reject the null hypothesis, H₀. ²⁴

Alternatively, we can state this in terms of probabilities by looking up the p-value probability corresponding to the z-value in a table. If this problem probability, or p-value, is lower than the significance level probability, α = 0.01 = 1%, then we have to reject the null hypothesis, H₀. ²⁵

By the power of hypotheses... I have the power analysis!

Hypothesis testing basically asks the question: for a given initial position, does the evidence we’ve seen compel us to change our minds away from that initial position? Thus, we start by assuming the null hypothesis is true; this is our initial position. We then look at some evidence and calculate a p-value, the probability or likelihood of our initial position, the null hypothesis. This p-value helps determine if we learned enough from the evidence to change our minds and reject our initial position, the null hypothesis. We do that only if the evidence is surprising or, as we might say in the information theory paradigm, increases the amount of information gained. If the p-value, the probability of our null hypothesis being true, is lower than some significance level, the amount of surprise, or the amount of information gained, is large and we should likely change our minds and reject the null hypothesis.

The power of an experiment is the probability of avoiding a false negative (β); i.e., the probability of rejecting the null hypothesis when it actually is false: Pr(reject H₀|H₁). In essence, the power of a process measures whether we have collected enough evidence to justify changing our minds. Power is equal to 1 − β and is proportional to the sample size and inversely proportional to the significance level (α) and effect size, d, (using something like Pearson’s Correlation Coefficient).

Power analysis helps identify how much power is associated with a given amount of data and usually determines the sample size for given values of alpha, beta, and the effect size, which are sometimes set to default values of α = 0.05, β = 0.8, and d = 0.8, respectively.

10.5.2 Hypothesis Testing

Let’s calculate some of these probabilities so we can quantify the test of our hypothesis. Suppose the pregnancy test claims it can successfully detect 98.5% of pregnancies. This usually refers to True Positives (TP), where it tests positive (t+) when the person is pregnant (p+) with probability P(t+|p+), as seen in Figure 10.20b. The test kit’s insert also gives you the True Negative (TN) probability as 95%, i.e., the probability of testing negative (t−) when they’re not pregnant (p−): P(t−|p−).

²⁴ In the case of a two-tailed test, we would also compute the z-score for the left tail and would reject H₀ if the problem z-score was lower than that.

²⁵ In Bayesian Hypothesis Testing, we add a prior probability and then compute the posterior probability, which is the conditional probability after all the evidence is considered, as encapsulated in Bayes’ Theorem shown in Section 10.5.3.
This means the p-value, the probability of testing positive if they’re not pregnant, \( P(t_{+} | p_{-}) \), is 5% for this pregnancy test kit, as seen in Figure 10.20a. That means the likelihood of the test coming back positive and your not being pregnant is only 5%. Since this p-value probability is not lower than our stricter significance level probability, \( \alpha = 0.01 = 1\% \), you cannot reject the null hypothesis, \( H_0 = \text{Not-Pregnant} \), and decide to further confirm being not pregnant.

We can do so by using a z-score based approach towards checking our hypothesis. We might calculate the z-score corresponding to our p-value, the probability of testing positive if they are not pregnant, and then compare that to the value 2.58, the two-tailed z-score at \( \alpha = 0.01 \) as seen from a standard z-score table or by using the Python `scipy.stats` package: \( \text{scipy.stats.norm.ppf}(1-(1-0.99)/2) \). For our pregnancy test kit, the z-score corresponding to our p-value of 5% is 1.96.\(^{26}\) Since the test’s z-score of 1.96 is not higher than the z-score of 2.58 for \( \alpha = 0.01 = 1\% \), we again cannot reject the null hypothesis, \( H_0 = \text{Not-Pregnant} \). But can we be sure we’re really not pregnant?

This points to one of the most common misconceptions about p-values: the p-value by itself isn’t intended to be a final probabilistic threshold of truth; instead, it’s usually intended to be a guide to help answer the question of interest when it’s combined with some prior knowledge. For example, combining a low p-value with the prior probability of pregnancy in the population, along with the general viability of the person using the test, will help you gauge the likelihood of being pregnant.

Suppose the prior probability of someone using a pregnancy test kit actually being pregnant is 40%: \( P(p_{+}) = 40\% \). For our pregnancy test kit, when someone pregnant takes the pregnancy test, it will correctly indicate pregnancy 98.5% of the time, so that \( P(t_{+} | p_{+}) = 98.5\% \). Someone who is not pregnant, on the other hand, would get a correct indication 95% of the time, so that \( P(t_{+} | p_{-}) = 95\% \) (and thus, \( P(t_{+} | p_{-}) = 5\% \), which is our p-value probability). Continuing our example, we can combine these probabilities with the prior probability to get the probability of someone being pregnant, given that the test gives a positive result, as:

\[
P(p_{+} | t_{+}) = \frac{P(t_{+} | p_{+}) P(p_{+})}{P(t_{+})} = \frac{0.985 \times 0.4}{0.424} = 93\% 
\]

(10.2)

since the numerator is \( P(t_{+} | p_{+}) P(p_{+}) = 0.985 \times 0.4 = 0.394 \) and the total probability for a test showing positive is \( P(t_{+}) = P(t_{+} | p_{+}) P(p_{+}) + P(t_{+} | p_{-}) P(p_{-}) = 0.985 \times 0.4 + 0.05 \times 0.6 = 0.424 \).\(^{27}\) This rather high probability might make us consider taking the test again, just to confirm!

### 10.5.3 Bayes Theorem

How should retaking the test inform or change our current understanding? It turns out that Equation (10.2), also known as Bayes Theorem or Bayes Rule, lets us do just that! It is usually written as:

\[
P(p_{+} | t_{+}) = \frac{P(t_{+} | p_{+}) P(p_{+})}{P(t_{+})} 
\]

(10.3)

where the left hand side is the probability of the hypothesis being true, \( P(p_{+} | t_{+}) \), called the posterior, or the probability we’re interested in figuring out; in this case, the posterior is the probability of being pregnant given the test returns a positive. The right hand side is made up of three parts:

- The probability of the observed data/experimental outcome, \( P(t_{+} | p_{+}) \), called the likelihood

\(^{26}\)In a normal distribution, approximately 95.45% of the area under the curve lies between \( \mu - 2\sigma, \mu + 2\sigma \).

\(^{27}\)This is also the Precision, as we calculate in Section 10.5.7 and see in Figure 10.20.
Chapter 10. Machine Learning and Data Science

Figure 10.20: Probability Tree, Confusion Matrix, and Two-Tailed Z Test:

(a) Probability Tree: Suppose in a population of 1,000 people, the prior probability of being pregnant, $P(p^+)$, and not being pregnant, $P(p^-)$, is 40% ($\frac{400}{1000}$) and 60% ($\frac{600}{1000}$), respectively.

(b) Confusion Matrix: shows number of people who tested pregnant versus those actually pregnant, along with False Positives (FP), False Negatives (FN), True Positives (TP), and True Negatives (TN).

(c) A Two-Tailed Z-Test showing the significance levels.

From the probability tree, we can see that $P(p^+) = 40\%$, $P(p^-) = 60\%$, $P(t^+|p^+) = 98.5\%$, $P(t^-|p^+) = 1.5\%$, $P(t^+|p^-) = 5\%$, $P(t^-|p^-) = 95\%$. So the probability of being pregnant, given that the test gives a positive result, is $P(p^+|t^+) = \frac{P(t^+|p^+)P(p^+)}{P(t^+)} = \frac{98.5\% \times 40\%}{98.5\% + 40\% + 5\% + 95\%} = \frac{394}{424} = 93\%$.

- The a priori probability, $P(p^+)$, called the **prior**
- The overall probability of the observed data, $P(t^+)$, called the **evidence**

In a Bayesian approach, we’d combine the prior with the likelihood and then iteratively update the formula as per Bayes Theorem in Equation (10.3). This is an alternative to the **frequentist** approach in NHST. In **Bayesian hypothesis testing**, evidence can be provided against the null hypothesis but also in favour of it. This is important as one of the biggest critiques against NHST, as explained by Zoltan Dienes in “Bayesian Versus Orthodox Statistics: Which Side Are You On?”, is that the NHST approach gives us the probability of the data being observed if the theory is true, $P(Data|Theory = "true")$, whereas what we really want in experiments is the probability of the theory being true given that the

---

\footnote{For our running example, the evidence, $P(t^+)$, is the probability for a positive result on the test, regardless of whether or not the underlying condition is true. That is, it includes probabilities for both the true positives AND the false positives. As such, we could even think of Bayes theorem as expressing the precision, $\frac{TP}{TP+FP}$, as seen in Section 10.5.7.}
data is seen, \( P(\text{Theory}|\text{Data} = \text{"observed"}) \).

Thus, the only conclusion NHST can give us is whether or not to reject the null hypothesis, \( H_0 \), but this is often misunderstood and a significant p-value is incorrectly interpreted as evidence for the alternative hypothesis, \( H_A \). There are also issues with the effect size and the size of the sample, as well.

Bayes rule, on the other hand, allow us to assess and adjust our confidence in the alternative hypothesis against the competing null hypothesis. But the Bayesian approach is not free from criticism, especially its reliance upon priors without a clear method for designing and setting priors.

### 10.5.4 Bayesian Hypothesis Testing

As we look at these pregnancy test examples, it’s important to note, from a probabilistic perspective, that the test for pregnancy is different from the event of being pregnant. Tests themselves can be flawed: they might say you’re pregnant when you’re not (false positive) or say you’re not pregnant when you actually are (false negative).

False positives can especially skew results: suppose some phenomenon, unlike pregnancy, were fleeting rare in the population, like 1 in every million people is super-rich (there are something like 2,500 billionaires in the world so that would be closer to 1 in 3 million or so). In this case, if someone tests positive for being a billionaire, it’s highly likely that the test is a false positive, given the rarity of billionaires in the wild. So if you see someone spending big money at a fancy restaurant, are they likely to be a billionaire? One way to make your guess, or prediction, better is to look for more data.

**Problem 10.1** Besides eating an expensive meal at a fancy restaurant, what other evidence might make you more inclined to think a person was a billionaire? What kind of evidence would make you less inclined to think that?

As more data comes in, you might start to revise the probability of that person being a billionaire, either making it more likely or less likely. This idea of revising your beliefs (defined as the probability assigned to a certain event) based on new facts or evidence, is not only at the heart of science but also captured in Bayes’ Rule or Bayes’ Theorem, where the strength of evidence leads to revising our reasoning approach by going back to our pregnancy test example.

As we worked out in Equation (10.2), if the a priori probability of your being pregnant is given by the prior, \( P(p+) = 0.4 \), and the probability of the test showing positive if you’re already pregnant is given by the likelihood, \( P(t+|p+) = 0.985 \), then the probability of the test showing positive, regardless of whether someone is pregnant or not, is given by the evidence, \( P(t+) = P(t+|p+) \ast P(p+) + P(t+|p-) \ast P(p-) = 0.985 \ast 0.4 + 0.05 \ast 0.6 = 0.424 \).

Then, the posterior probability of our alternative hypothesis, \( H_A \) ("you are pregnant"), is given by Equation (10.3) which gives us the odds of being pregnant if the test comes back positive as

\[
P(p+|t+) = \frac{P(t+|p+) \ast P(p+)}{P(t+)} = \frac{0.985 \ast 0.4}{0.424} = 93\%
\]

In the Bayesian approach to Hypothesis Testing, we form a **Bayes Factor**, which is just the ratio of the posterior odds divided by the prior odds of the two competing hypotheses, \( H_0 \) and \( H_A \). In our case, the null hypothesis, \( H_0 \), was that “you’re not pregnant” and the alternative hypothesis, \( H_A \), was that “you are pregnant.” The corresponding posteriors are \( P_o(p-|t+) = \frac{P(t+|p-) \ast P(p-)}{P(t+)} \) and \( P_A(p+|t+) = \frac{P(t+|p+) \ast P(p+)}{P(t+)} \)
and the corresponding priors are $P_o(p−)$ and $P_A(p+)$. Then, the Bayes factor is given by:

$$BF_{H_AH_o} = \frac{\text{posterior odds}}{\text{prior odds}} = \frac{P_A(p+|t+)}{P_A(p−|t−)} = \frac{93}{28} = 3.28 = 20.12$$  (10.4)

The Bayes factor uses the observed data to provide a measure of the evidence for $H_A$ versus $H_o$. In this case, the probability to observe the data (positive test result) under the alternative hypothesis (“you are pregnant”), $H_A$, is about 20 times more likely than observing that same data (the positive test result) under the null hypothesis (“you’re not pregnant!”), $H_o$.

Although thresholds are viewed skeptically in the Bayesian formulation, there are some suggested scales for interpreting the relative plausibility of the two models. In one popular approach, as elaborated by Alonso Ortega and Gorka Navarrete in “Bayesian Hypothesis Testing: An Alternative to Null Hypothesis Significance Testing (NHST) in Psychology and Social Sciences,” a value of 1 is considered to be no evidence; a value above 100 is considered extreme evidence for $H_A$ and a value less than $\frac{1}{100}$ is considered extreme evidence for $H_o$. In our case, a value of 20 is strong evidence for $H_A$, that “you are pregnant.” Should we cancel the dental appointment to have X-Rays then?

As a final confirmation, perhaps you decide to take the test again just to confirm everything and it comes back positive again. We can use the Bayesian approach iteratively to calculate the actual probability of being pregnant given this new data. In the Bayesian approach, we use the previously calculated posterior probability as our new prior.

We then do the calculation just as we did before: in this case, the new prior becomes $P(p+) = 0.93$, our previously calculated posterior probability. We then calculate the new likelihood as $P(t+|p+) = 0.985 * 0.93 = 0.916$ and the new evidence as $P(t+|p+) * P(p+) + P(t+|p−) * P(p−) = 0.985 * 0.93 + 0.05 * 0.07 = 0.9195$.

The final new posterior then becomes $P(p+|t+) = \frac{0.916}{0.9195} = 99.6%$; that means our posterior probability, $P(p+|t+)$, of the alternative hypothesis, $H_A$, “you are pregnant,” went from 93% after the first pregnancy test to 99.6% after the second pregnancy test and it confirms our expectation as we slowly throw all the alcohol out of our house.

### 10.5.5 Bayesian Inference and Statistics

Bayes theorem is a very good starting point to infer individual posterior probabilities and learn iteratively from new data, one observation at a time. But it can be extended to continue to iteratively learn statistically in Bayesian statistics by modeling each of the components (the prior, likelihood, posterior, and evidence) as probability distributions rather than individual probability values. **Bayesian inference** deals with the properties of probability distributions and deducing the properties of a population using some observed data and Bayes Theorem.

In this statistical approach, the end result is that rather than using single numbers for each of the terms in Bayes rule, we instead use **probability distribution functions (PDFs)**. Since we use PDFs for the likelihood, the prior, and the evidence, the posterior is also expressed as a PDF. These PDFs constitute the particular model for our data and the system and are represented with certain symbols. Instead of dealing with an event like $p+$, we now deal with probability distributions that are characterized by some set of parameters, $\Theta$. For example, a Gaussian distribution is characterized by its mean and standard deviation so we’d represent those parameters as $\Theta = \{\mu, \sigma\}$.

The data we saw in the single number version of Bayes theorem was something like $t+$, which represented the test result and is used to calculate probability numbers like the evidence, $P(t+)$. In
a statistical inference model, the data consists of some set of observations and is usually represented as \( Y = \{ y_1, y_2, \ldots, y_n \} \); instead of \( Y \), you might see the letter \( D \) used for the observed data instead. In this formulation, the evidence \( P(Y) \), the denominator in Equation (10.3), doesn’t reference any of the model parameters (\( \Theta \)) in which we’re interested and, in fact, the evidence isn’t even a distribution; it’s just a number that’s needed to ensure the final product is normalized between 0 and 1 like a good probability should be.

As it turns out, this normalizing constant which ensures that the integral (or the sum) of the PDF adds up to 1 is usually very difficult to compute so the evidence (the denominator in Equation (10.3)) is often just ignored and we instead change the equality in Equation (10.3) to a proportionality and carry on as if nothing else has changed as we’re usually interested in only the peaks of the distribution not the actual value of the entire distribution itself. When we visualize these probability distributions, the narrower the curve (e.g., for the posterior), the more informative it is about the value of the parameters; so a wider curve is, in general, less informative about the parameter values.

Statistical inference problems are hard as these probability distributions can get quite messy. One trick that’s widely used to make the math easier is to employ distributions that can serve as conjugate distributions: these are distributions that belong to the same family of distributions (like the Gaussian or multinomial) such that when you multiply them together, you end up with another distribution in the same family.

Suppose you have a likelihood PDF, as observed from data, that follows say a Gaussian distribution; if you then pick a prior probability distribution function that’s also a Gaussian, your posterior probability distribution function will end up also being a Gaussian. In this case, a Gaussian is considered self-conjugate but you have other such pairings which are quite popular, like the Multinomial-Dirichlet conjugate prior pairing for topic modeling approaches such as Latent Dirichlet Allocation (LDA) or the Binomial-Beta conjugate prior pairing for simple question scenarios.

Topic models like the LDA are an approach for unsupervised classification of unlabelled data by clustering words, either text words or other symbolic representation “words”, into natural groups that might correspond to an abstract semantic label, or topic. For example, documents on physics will mention entropy more often than documents on cake recipes. Topic models model both topic and word distributions as Dirichlet distributions so a document is made up of a mixture of topics and each topic is made up of words, all following such distributions.

If you can’t pick your own priors to be nice like conjugate priors, you have to rely on other approaches like using Markov chain Monte Carlo (MCMC) methods and its many variants. MCMC is useful for sampling from a complex distribution, perhaps like the posterior distribution we want, when we can’t easily compute that entire distribution. The MCMC variants will allow us to draw samples from such distributions and can also be used to do things like compute high-dimensional integrals which are common in machine learning and physics.

### 10.5.6 Supervised Learning with Naive Bayes

Bayes Theorem tests how much you can trust the evidence that’s coming in, as we saw in the case of the pregnancy test, in which the second test confirmed even more strongly our initial suspicion. In the same way, we can utilize Bayes rule to help machines learn from data as the machine sees more and more of the data, or evidence.

The most popular application of Bayes theorem to machine learning is in classification and it makes a couple of seemingly strong assumptions: it assumes that feature sets for both of the two classes are identical and most importantly, and naively, that all features are independent of each other; that means
Chapter 10. Machine Learning and Data Science

for two features, \( A \) and \( B \), their joint probability will be, \( P(A, B) = P(A)P(B) \) since they’re independent of each other. This naive assumption sometimes actually helps the classification as it counts the same evidence twice for the two different independent features; if the features weren’t independent, that evidence would only be counted once.

The approach of using this naive assumption is called Naive Bayes and it often performs better than other models like Logistic Regression when classifying categorical variables. In these cases, it is fast and doesn’t require a lot of training data. When dealing with numerical data, you can model it as a normal distribution, although that might be a strong assumption in some cases. With all these assumptions, Naive Bayes can be a bad estimator sometimes but it’s usually good enough for some applications and an easy benchmark to implement and compare with more sophisticated approaches.

### Load Required Libraries

```python
from sklearn.datasets import load_iris # Get Iris dataset
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics

# Load Iris Dataset and get the Features matrix and Label vector
iris = load_iris()
X = iris.data
y = iris.target

### Create the Training and Testing Datasets
(X_train, X_test, y_train, y_test) = train_test_split(X, y, test_size=0.4, random_state=1)

### Create and train the model:
gnb = GaussianNB()
gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_test) # Make the predictions on Test Dataset

### Get accuracy of ground truth (y_test) vs predicted labels (y_pred)
acc = metrics.accuracy_score(y_test, y_pred)*100
print("Our model’s accuracy on the Iris dataset is: ", acc)
```

Listing 10.3: Gaussian Naive Bayes classifier on the Iris Dataset.

We can see how easy it is to implement such a Naive Bayes classifier in Python, as seen in Listing 10.3, where we again use scikit-learn, just as we did earlier for the decision tree classifier in Listing 10.1. Just as we saw there, we first import the required libraries in Lines 1 - 5, load the default Iris dataset which has data like the sepal length for various Iris flowers, and then use the same `train_test_split()`, `fit()`, and `predict()` functions before we also use the same `metrics.accuracy_score()` metric to evaluate how well our classifier did which, in this case, gave a result of 95%; not too shabby for a default classifier cobbled together in about 20 lines of code!

---

The only difference is that we used a Gaussian Naive Bayes classifier since we’re dealing with continuous values; as such, each continuous value (like sepal length = 5.1 for Iris flower #1) is assumed to be distributed according to a Gaussian distribution and so the likelihood is converted to the result of a Gaussian, like 5.1 being converted to 1.094 as follows:

\[
P(x_i|Y = \text{"I.setosa"}) = \frac{1}{0.35\sqrt{2\pi}} e^{-\frac{(5.1 - 5.1)^2}{2(0.35)^2}} = 1.094
\]
10.5 Bayesian Inference and Hypothesis Testing

We can figure out all the possible combinations of true positives, false positives, true negatives, and false negatives in a confusion table. We can then use this confusion matrix, as shown in Figure 10.21, to derive some associated metrics to help gauge the validity of our machine learning models:

1. **Accuracy**: The ratio of the correctly predicted observations to the total number of observations. Accuracy works well for symmetric datasets where the values of FP and FN are proportional or for datasets with an even, balanced distribution where the TN or TP are not dis-proportionately large; if the dataset is imbalanced, it leads to the **accuracy paradox**, which requires other measures like Precision, Recall, or F1 Score to fully assess a model. Overall, how often is it right?

   \[
   \frac{\text{All Correct Picks}}{\text{All Possible Choices}} = \frac{TP + TN}{TP + TN + FP + FN}
   \]

2. **Error**: Also known as **Misclassification**: the complement of Accuracy is **Error** = (1 - Accuracy). Overall, how often is it wrong?

   \[
   \frac{\text{All INcorrect Picks}}{\text{All Possible Choices}} = \frac{FP + FN}{TP + TN + FP + FN}
   \]

3. **Prior Probability**: P(Disease+) = (TP+FN) / (TP+FP+TN+FN) Also called the **Pre-Test Probability**, this is the estimated likelihood of disease before the test is conducted. If this is calculated
on the entire population, it will also be the Prevalence of the disease in the population at large.

\[
\text{ACTUAL Positives} \over \text{All Possible Choices} = \frac{TP + FN}{TP + TN + FP + FN}
\]

4. The Predictive Value of a Negative Test: \( P(\text{Disease-} \mid \text{Test-}) = \frac{TN}{TN + FN} \) This is the proportion of patients who do not have the disease and test negative for the disease. The Post-Test Probability of the disease given the patient tests negative is 1 - Predictive Value of a Negative Test.

\[
\frac{\text{TRUE Negatives}}{\text{Predicted Negatives}} = \frac{TN}{TN + FN}
\]

5. Precision: \( P(\text{Disease+} \mid \text{Test+}) = \frac{TP}{TP + FP} \) The Predictive Value of a Positive Test is the proportion of patients who actually have the disease and test positive for the disease. This is helpful when False Positives are costly. When it predicts positive, how often is it correct?

\[
\frac{\text{TRUE Positives}}{\text{Predicted Positives}} = \frac{TP}{TP + FP}
\]

6. Recall: \( P(\text{Test+} \mid \text{Disease+}) = \frac{TP}{TP + FN} \) This is also known as Sensitivity and the True Positive Rate. This is helpful when False Negatives are costly. When it’s actually positive, how often does it predict positive?

\[
\frac{\text{TRUE Positives}}{\text{ACTUAL Positives}} = \frac{TP}{TP + FN}
\]

7. Specificity: \( P(\text{Test-} \mid \text{Disease-}) = \frac{TN}{TN + FP} \) Also known as the True Negative Rate. When it’s actually negative, how often does it predict negative?

\[
\frac{\text{TRUE Negatives}}{\text{ACTUAL Negatives}} = \frac{TN}{TN + FP}
\]

8. False Positive Rate: When it’s actually negative, how often does it predict positive?

\[
\frac{\text{False Positives}}{\text{ACTUAL Negatives}} = \frac{FP}{TN + FP}
\]

There are also alternate measures that are effective like the F1 Score, the weighted arithmetic mean of Precision and Recall. This measure can be helpful if you need to strike a balance between Precision and Recall in a dataset with an un-even, skewed distribution where it has a large number of TN, for example.

\[
F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}
\]

Although this weighted average isn’t as intuitive as accuracy, it’s more useful for uneven class distributions as it accounts for both the False Positives and the False Negatives. E.g., if we want to catch terrorists and examine a billion people, we can label all of them as Not Terrorists. In this case, our accuracy is awesome but our TP is 0 so both Precision and Recall are 0 and so our F1 is also 0.

There are also additional measures that are widely used, like the Receiver Operating Characteristic (ROC) curve, Area Under the ROC Curve (AUC), etc., which are useful under certain circumstances,
especially when you want to rank predictions about both the positive and negative classes and your dataset is relatively well-balanced. The ROC curve gives a measure of the performance of a classification model at all classification thresholds. When it is well-balanced, accuracy is also an informative metric. But it’s likely always a good idea to show the confusion matrix and F1 score and, especially if your data is unbalanced or costly, to use the false positive and false negative via the precision and recall, respectively.

When to use which metrics?

Many classification algorithms like decision trees use all of these metrics as well as metrics used for general predictive models like the cross-validation error. In general, the following metrics are usually good fits with Supervised and Unsupervised learners as a rough heuristic:

- Supervised Learning: Classification: Precision-Recall, ROC, AUC, Accuracy, Log-Loss, etc.
- Supervised Learning: Regression: MSPE, MSAE, R-Squared, Adjusted R-Square, etc.
- Unsupervised Learning: Rand Index, Mutual Information, etc.

Problem 10.2 We know that, when evaluating a potential model, we can gauge it as:

- High Precision + High Recall: Great model
- High Precision + Low Recall: Misses positive cases but great at the ones it does catch
- Low Precision + High Recall: Gets most of the positives but has lots of false alarms
- High Specificity: Minimize false positives: let 10 guilty go free but don’t convict an innocent
- Low Precision + Low Recall: Terrible model, better to just guess randomly

So the first model obviously performs the best and the last model obviously performs the worst but, of the remaining three evaluation metrics for three different models, which model would you use to detect diseases like cancer? How about for spotting terrorists? How about winning football teams?

10.6 The Entropy Strikes Back

The mathematical underpinnings of creating decision trees are important as they’re applicable to many other algorithms, not just algorithms for classification. These underpinnings also bring together fundamental ideas from Part I, especially the idea of Shannon Information and Shannon Information Entropy as we saw in Section 1.9. Let’s dive right in by looking at random variables in detail. This section will be a bit concept heavy so please do feel free to skim it if it seems to plod along.
Summary of Entropy

As we saw in Section 1.9, $h(p)$ is Shannon’s Information Function and is a measure of the decrease in uncertainty at the receiver’s end: $h(p) = \log_2 \left( \frac{1}{p} \right)$. It’s a measure of unpredictability or uncertainty. It is also the amount of surprise: the smaller the probability, the higher the surprise and the higher the information. The goal in machine learning is usually to reduce uncertainty.

Shannon’s Information Entropy is $H(p)$ and is the average Shannon Information per symbol: the sum of Shannon Information for each symbol weighted by the probability of that symbol and is defined as $-\sum p \log_2 (p)$, as seen in Section 1.10. The higher the entropy, the higher the Shannon Information content of a system: this is how many bits of Shannon Information we need on average to describe its exact state. We can also think of it as the average of the surprise associated with all the possible observations. This is also the average uncertainty: the goal of most machine learning is to lower uncertainty and hence to lower Entropy!

Entropy is maximized when the states are equiprobable and all have the same likelihood. In order for the average Shannon Information to be high, the distribution of probabilities must allow for a large number of unlikely events. We can think of it as a measure of the spreading out of the probabilities. So the more spread out the probabilities (all states are equally likely), the higher the entropy.

10.6.1 Random Variables

In order to define a random variable, we’ll need to establish a few other definitions related to the possible observations of some random phenomenon. A random phenomenon or a random experiment, $\varepsilon$, is one where we don’t know exactly what the result will be ahead of time. A random action or a series of actions, perhaps the result of an experiment, is called a trial. A random, non-deterministic phenomenon or experiment can result in a certain outcome, $\omega$. The outcome is the result of the trial.

The set of all possible outcomes is the sample space, $\Omega$. The specific outcomes recorded for a particular experiment are a subset of the sample space and are called the event. The event is just a specific collection of outcomes and the size of an event is the number of outcomes in that event. The set of all possible events is the event space, $\Sigma$. An event space contains all the possible events for a particular experiment or phenomenon.

Although an individual random event is non-deterministic or unpredictable, random phenomena often follow a probability function overall so we can assign a probability or frequency to different outcomes over a number of events or trials. Randomness can also be considered a measure of the uncertainty of a particular outcome. A random process thus follows a probability distribution.

We can then define the probability of a particular event, $E$, as:\[^{30}\]

$$P(E) = \frac{\text{Size of Event}}{\text{Size of Sample Space}} = \frac{\# \text{ of Outcomes in the Event}}{\# \text{ of Possible Outcomes in the Sample Space}}$$

For example, if you have a die, the sample space, $\Omega$, might be all possible rolls: $\Omega = \{1, 2, 3, 4, 5, 6\}$. A particular outcome, $\omega$, might be $\omega = 3$, where you roll a 3. An event, $E$, might be rolling a number greater than or equal to 3: $E_{\text{roll} \geq 3} = \{3, 4, 5, 6\}$. Then, the $P(E_{\text{roll} \geq 3})$ is:

[^30]: The most fundamental, and unrealizable, definition of probability is the frequentist one: $P(E) = \lim_{n \to \infty} \frac{n(E)}{n}$ where $n$ is the number of trials of an experiment and $n(E)$ is the number of trials that result in $E$. 

Contact adopt@cognella.com, 800.200.3908, for more information © COGNELLA

DO NOT DUPLICATE, DISTRIBUTE, OR POST
10.6 The Entropy Strikes Back

A **probability distribution function (PDF)** defines the probability of every possible event in order to compute the **probability distribution**. It assigns a probability to every event possible such that the probability of \( \Omega \), the entire sample space, is equal to 1. A **probability space**, thus, is a sample space that has a probability function defined on it. We can then define a **random variable** as a function on that probability space that maps outcomes in the sample space (like \( \omega \)) to the set of real numbers, \( \mathbb{IR} \).

**Notational Information**

To simplify notation, I differentiate between the probability of an event in \( \Sigma \), described by a probability distribution function, and the intermediate functions used to calculate the probability of a random variable assuming a certain value. So I use \( P(\cdot) \) to indicate the former (the probability value from the Probability Distribution Function) and \( Pr(\cdot) \) to indicate the latter, the probability of a random variable having a specific value (discrete random variable) or being near a specific value (continuous random variable).

In addition, I abbreviate a general Probability Distribution Function as **PDF** and I abbreviate the particular functions used to calculate the probability of a random variable as either **pmf** (Probability Mass Function), if it has a specific value, or as **pdf** (Probability Density Function), if it’s near a specific value, as we’ll see shortly.\(^\text{31}\)

Suppose we are interested in the event of a die roll being greater than or equal to 3 for a single die roll. Our event would be made up of four outcomes as, \( E_{\text{roll} \geq 3} = \{3, 4, 5, 6\} \). We could create a random variable called \( X \) that has the values of 0 or 1, two real numbers. The random variable \( X \) maps those two real values to the outcomes as follows: it maps 0 to the outcomes of rolling a 1 or 2 and it maps 1 to the outcomes of rolling a 3, 4, 5, or 6. We can say the first mapping corresponds to the event of rolling less than a 3: \( E_{\text{roll} < 3} = \{1, 2\} \) and the second mapping corresponds to the event of rolling a 3 or higher: \( E_{\text{roll} \geq 3} = \{3, 4, 5, 6\} \). The random variable \( X \) is then \( X = \{0, 1\} \). We can calculate a probability for each of these values of the random variable, \( X \):

\[
P(X = 1) = \frac{\# \text{ of Outcomes in the Event}}{\# \text{ of Possible Outcomes in the Sample Space}} = \frac{4}{6} = \frac{2}{3}
\]

\( P(E_{\text{roll} \geq 3}) = \frac{\# \text{ of Outcomes in the Event}}{\# \text{ of Possible Outcomes in the Sample Space}} = \frac{4}{6} = \frac{2}{3} \)

\(^3\text{1 Another function that’s related to the probability distributions of random variables is the Cumulative Distribution Function (CDF). In general, a non-random, known, deterministic variable doesn’t need a distribution to describe it, of course. But for a random variable, the CDF gives us the probability of a particular random variable having a value up to a certain quantity. For example, the probability distribution for rolling a die is given by a pmf. You could compute the probability for this discrete random variable assuming a value less than or equal to 3. The 3 outcomes which fulfill that condition each have a probability of \( \frac{1}{6} \). The cumulative probability would be the addition of those three probabilities and so is \( \frac{3}{6} \). The CDF of a discrete random variable is then } F(x) = Pr(X \leq x) = \sum_{x_i \leq x} Pr(X = x_i), \text{ with integrals replacing sums for pdfs.} \)
\[
Pr(X = 0) = \frac{\text{Number of Outcomes in } E_{\text{roll} < 3}}{\text{Number of Outcomes in } \Omega} = \frac{2}{6} = \frac{1}{3}
\]
\[
Pr(X = 1) = \frac{\text{Number of Outcomes in } E_{\text{roll} \geq 3}}{\text{Number of Outcomes in } \Omega} = \frac{4}{6} = \frac{2}{3}
\]

It’s important to note that the random variable \( X \) is a function, regardless of what the name might imply. As such, the random variable \( X \) is a function that maps outcomes to real numbers. We could technically represent this as \( X = 0 \implies \{ \omega \in \Omega : X(\omega) = 0 \} \) which says that \( X = 0 \) is a shorthand for referring to the set that consists of all the outcomes \( \omega \) that are mapped to the real number 0. We forego this rather cumbersome technical expression and just look at \( X = 0 \) without any direct reference to \( \Omega \).

The random variable \( X \) has its own sample space and events so we can refer to the probability that \( X \) takes on a specific value, or a range of values, as the value of the probability function for \( X \): \( Pr(X = 1) = 4/6 = 2/3 \). We can think of this as \( Pr(X = \text{value}) = \text{probability of that value} \) and this is read as “Probability that \( X \) is equal to 1 is 2/3”.

### Discrete vs Continuous Random Variable

A **Discrete Random Variable** is a random variable that can take any of a discrete set of values from the set of integers. In this case, the Probability Distribution Function (PDF) uses the probabilities calculated by a **Probability Mass Function (pmf)** which is defined, for any value \( x \), as the probability that the random variable \( X \) equals the value \( x \): \( Pr(x) = Pr(X = x) = P_X(x) \). The pmf \( Pr(x) \) has to be non-negative for each possible value \( x \) and must sum to 1; these conditions are expressed as \( Pr(x) \geq 0 \) and \( \sum Pr(x) = 1 \). For example, the pmf for a die is \( Pr(x) = \begin{cases} \frac{1}{6} & \text{if } x \in 1, 2, 3, 4, 5, 6 \\ 0 & \text{otherwise} \end{cases} \)

A **Continuous Random Variable**, on the other hand, can take any real number on the interval \([0, 1]\) with equal probability. It can take any value from any continuum actually but it’s easier to transform, or normalize, it to the \([0, 1]\) range. Since a continuum is uncountable, we can’t sum over it directly and must integrate over that range instead. Thus, the Probability Distribution Function (PDF) for a continuous variable depends upon a **Probability Density Function (pdf)**, which is represented as \( \rho(x) \) and which assigns a probability that \( X \) is near a value \( x \) rather than exactly equal to a value \( x \). We use the pdf to get the probability distribution by calculating the probabilities as \( Pr(X \in A) = \int A \rho(x) dx \).

Thus, the probability that \( X \) is in any set \( A \) (such that \( X \in A \)) is obtained by integrating \( \rho(x) \) over the set \( A \). This means the probability for a single point is 0 since the integral over a single point is 0 (i.e., the set \( A \) contains a single element above). In addition, the pmf must be positive and integrate to 1 since \( \rho(x) \geq 0 \) and \( \int \rho(x) dx = 1 \) (the integral is over all values of \( X \)).

A common pdf that’s often found in nature is a Gaussian distribution which, for standard Gaussian Random Variables with mean \( \mu = 0 \) and standard deviation \( \sigma = 1 \), is:

\[
\rho(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \rightarrow \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \tag{Standard Gaussian}
\]

For a Gaussian Distribution the mean = mode = median, which is a very helpful property as you can then completely specify it with just the mean, \( \mu \), and standard deviation, \( \sigma \). For the standard Gaussian random variable above, this is shown as \( X \sim N(0,1) \).
In addition, the Central Limit Theorem states that if you take sufficiently large random samples from some population, the mean of the sampling distribution will be the mean of the population for many systems. In fact, quite a few systems exhibit a Gaussian distribution naturally; systems like diffusion of particles, heights of people in a university, etc., since any natural process that preserves information about only the mean and the standard deviation can be characterized as a Gaussian.

Finally, as we saw in Section 10.5.5 when we discussed conjugate priors, unlike other distributions, a Gaussian tends to produce another Gaussian as the result of a transformation like product, sum, convolution, fourier transform, etc.

### Binomial Distribution

The binomial distribution is another popular distribution in which we deal with the likelihood of things like coin tosses, a pregnancy test, making a basketball shot, etc. In all these situations, each outcome has only two possibilities or states: success or failure. The event space is finite, which means we have a finite number of trials only.

Each event’s outcome must be mutually exclusive (either success or failure but not both) and the event’s likelihood should be complementary, in that the sum of the probabilities for each outcome must be 1. Finally, each sample or trial should be independent of the previous one such that the previous trials don’t affect the current trial.

If an experiment meets these criteria, its distribution can then be described as:

\[ P(X = k) = \binom{n}{k} p^k (1 - p)^{n-k} \]

where \( \binom{n}{k} \) is “n choose k” and is defined to be:

\[ \binom{n}{k} = \frac{n!}{k!(n-k)!} \]

### 10.6.2 Shannon Information, Shannon Information Entropy, and Information Gain

We can then define the entropy of a random variable in a couple of ways: either we can define it axiomatically as a certain measure with certain properties or we can motivate its definition by showing it can answer natural questions that arise in studying physical phenomena. Once we’ve defined entropy, perhaps as in Section 1.10, we can see that the larger the entropy of a specific distribution, the harder it will be to predict it (or compress it) and the more spread out (or less spiky) it will be.

Overall, entropy becomes a measure of how much information can be encoded in a message. The higher the entropy, the higher the information content that can be potentially resolved from it. Entropy can also be considered to be a measure of the amount of uncertainty in a message since information and uncertainty are equivalent concepts in this formulation. When looking at message encoding, entropy can even give the actual number of bits of information that are contained in a message source.

We can also look at entropy from a physical perspective, as we saw in Section 1.10.1. Since any given macrostate can be achieved by a large number of microstates, which particular microstate
describes the current state of the system is missing and is captured by this idea of entropy. For example, if I roll three dice and say the sum was 11 (this is the macrostate of the three dice system), you don’t know which particular configuration of the dice (which microstate) corresponds to that macrostate: was it 2 5’s and 1 1? Was it 2 4’s and 1 3? Was it 2 3’s and 1 5? etc.

As we saw in Section 1.10.1, entropy is really a bookkeeping tool to keep track of all the degrees of freedom we abstracted out by not including them directly in our calculation. We can do a little bookkeeping and keep track of the possibilities by calculating a number to represent all the microstates; if all these microstates of a system \( P, \Omega_p \), are equally likely, the number we can use to make sure we can add the microstates and probabilities correctly (e.g., if we add more die to our dice system) uses the logarithm. And our bookkeeping quantity becomes the Entropy, \( S = \ln(\Omega_P) \), what we called \( S = \ln(N) \) earlier in Section 1.10.1. If there’s only one possible microstate, then \( S = \ln(1) = 0 \) and we know everything there is to know about the system. As we increase the number of microstates, the amount we don’t know, the missing information, increases, as does the uncertainty about which state we’ll discover when we measure or observe the system. Entropy is a way to keep track of our ignorance about the underlying physical system: it is not a property of the system itself but a property of what we know or measure about the system.

And this is what we keep track of when we observe a particular outcome of one random variable of the system and see how it reduces the overall uncertainty based on another random variable of the system; i.e., \( \Delta H = H(X) - H(X|Y = y_i) \) and this reduction in uncertainty is our increase in information of the system. Shannon Information, which is defined as \( h(x) = -\log(p(x)) \), reflects a decrease in uncertainty (or an increase in the amount of surprise) when that event \( x \) occurs: this is what we call the amount of information contained in a discrete event \( x \) that we can potentially observe. An event is a particular outcome, or a subset of the sample space, from an experiment or a physical system that is modelled by a random variable.

The idea is to gauge how surprising some single event is; if it has a low probability of occurrence (like winning the lottery), it’s very surprising and reduces a larger uncertainty (high information); if it has a high probability of occurrence and will almost definitely happen (like the sunrise tomorrow), it’s not very surprising and so doesn’t really reduce the uncertainty as it was expected (low information).

We can generalize that decrease in uncertainty (or an increase in the amount of surprise) to a random variable itself: this lets us quantify the amount of information that we can potentially observe for a random variable \( X \) with a probability distribution \( P \). We can do this by calculating the amount of information we can potentially observe for the probability distribution of all the events for that random variable. In terms of just the probability distribution of a random variable, we can think of it as the number of bits required to encode or represent or transmit an event that is drawn from the probability distribution for that random variable.

In terms of physical systems, this is the amount of potential information in that system. Entropy is sometimes also called the expected value of self-information: entropy is the average amount of information we could gain about a system when we sample the random variable. The more balanced the probability distribution, the more surprising drawing a single event will be, so the higher the entropy, and the higher the potential information in that system. Conversely, the more skewed a probability distribution is, with some events being exponentially more likely than others for example, the less surprising it is to draw one of those likely events, so the lower the entropy, and the lower the potential information in that system.

Information gain is the decrease in entropy when we transform a system or a dataset in some way. The decrease in entropy is the same as a decrease in uncertainty (or an increase in the amount of surprise) and so an increase in the amount of information gained: \( I(X : Y) = H(X) - H(X|Y) = H(Y) - \)
10.6 The Entropy Strikes Back

10.6.3 Entropy and Information in Terms of Number of States of Physical Systems

Information, therefore, allows us to make predictions about a system that are better than by chance alone. In order to assess a physical system, we characterize it by first defining what we consider to be the system (using abstraction) and what we’ll measure on that system. Different definitions of the same underlying mechanism can resolve to different models of the system.

As we’ve seen, entropy is the lack of information, or the amount of uncertainty, and is given in terms of the log of the number of unknown states so that joint entropies can be added instead of multiplied. For example, if System 1 has $N_1$ states and System 2 has $N_2$ states, their joint system would have $N_1 \times N_2$ states.

We then consider information to be a difference of entropies. Information, the difference between two entropy values, is the actual content whereas the entropy values themselves can vary depending on how the system is defined. In fact, as noted by physicist and biologist Christoph Adami, the entropy of any physical object is infinite and only has a finite realization due to the finite nature of the measuring devices chosen (the choice of which depends on how the system is defined). This, as Adami shows in [6], is analogous to renormalization in Quantum Field Theory.

A Random Variable defines which states some system can take and with what probability for each state. These states were the outcomes with which we associated real numbers previously; those real numbers were like the labels or identifiers for those outcomes. How do we get those probabilities? Perhaps by frequencies of observation as we saw in the footnote to the definition of probability above. A random variable assumes you know things about the underlying system such as: the number of possible states to expect, what those states are, and possibly even what the likelihood is of experiencing those states. The entropy formula depends on the random variable and thus on knowing the possible outcomes and the probability distribution, perhaps from some underlying theory or frequentist observation.

Conditional Probability is the likelihood of two events occurring simultaneously when the second event has already occurred. Simultaneity means conditional probability cannot be defined unambiguously in a relativistic physics framework. We know that entropy is the uncertainty defined for a random variable. In addition, physical systems can be modeled by random variables only when you create a model, abstracting out information as reflected in the measurement devices we can use on that system. Entropy is defined by this model (and the corresponding measuring device). Information Theory, then, is just the theory of the relative states of measurement devices!

Conditional Entropy

Specific Conditional Entropy, $H(X|Y = v)$, is the entropy of the random variable $X$ among only those cases in which the random variable $Y$ has the value $v$. The overall conditional entropy or general conditional entropy, $H(X|Y)$, is the average of all the specific conditional entropies; i.e., it is the average of the specific conditional entropy across all values of $Y$. Since this involves probabilities, we cannot simply take the mean of the values but can, instead, weight them with the probabilities themselves.
Figure 10.22: Rényi Entropy: generalizes the Hartley entropy, the Shannon entropy, the collision entropy and the min-entropy: \( H_0, H_1, H_2, \) and \( H_\infty \) respectively.

If the probabilities in \( H(X) \) are not uniform (equiprobable), then they can be considered conditional probabilities by default (for any non-uniform distribution). The entropy of the equiprobable probability states is the Unconditional Entropy \( H_{\text{max}} \) and is the maximum value it can have. Information is then \( H_{\text{max}} - H_{\text{conditional}}, \) the unconditional entropy minus conditional entropy. Information can also be characterized as the Shared Entropy: \( I(X : Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) \). These are the observed entropies of \( X, H(X), \) and the actual observed entropy of \( X \) given what we know about the state of \( Y, H(X|Y) \). This means that we’re not aware we got our information from source \( Y \). This can be derived from the joint entropy, \( H(X,Y) = H(X) + H(Y|X) = H(Y) + H(X|Y). \)

Entropy can also be thought of as “potential” information; how much we could know. Here, we say \( I = H_{\text{max}} - H(X) \); in this case, \( H_{\text{max}} \) is what we don’t know about \( X \) and \( H(X) \) is what we observed about \( X \) (without regard to another system \( Y \)). This means that we’re not aware of how we got the information.

### 10.6.4 Entropy for Classification

Entropy is very useful for classification algorithms, as well. Categorical features, like non-numeric features, are unordered. We can encode the classes as integers but the numeric order will still be meaningless for them. Because of this, for class variables, we can check the mode but not other averages like the mean.

Instead, we can use Entropy as a way to measure the “amount of mix” in class variables since entropy measures the extent of mixture of values in the variable(s). It’s analogous to the “spread” in a numeric variable. As we saw earlier, entropy can also be thought of as the expected surprise from some observation. How surprising an event is depends on its probability. If it has a really low probability, we’ll be really surprised to see it → That means, it has higher entropy!

Let’s see an example by examining how to define entropy for a variable with only two possible values. Let’s say we have a coin; in this case, \( Pr(X = \text{Heads}) = 0 \) means we never get heads while \( Pr(X = \text{Heads}) = 1 \) means that we always get heads. Those are extreme probability values; for intermediate probability values, we’ll need a “mix” of the possible values.

Entropy is the function that allows mixing of the two possible values to intermediate values. Entropy measures the uncertainty in the state of a system and is a measurement function that is maximum at maximum impurity, where we would say that the values 0 and 1 would be pure values. Maximum uncertainty is when it’s “mixed” or “impure”. Entropy is thus used in machine learning for feature selection or deciding on splits in decision trees.

In general, entropy is at a maximum when all outcomes have equal probability. Any movement
away from that, in either direction, reduces the entropy value regardless of whether it’s towards heads or tails, as seen in Figure 10.22; in other words, the order or the class doesn’t matter. For more than two values, or classes, evenly spaced or evenly spread probability distributions have higher entropy than distributions with concentrations of probability values in a subset of possible values.

We can then use joint entropy, \( H(X,Y) = H(X) + H(Y|X) = H(Y) + H(X|Y) \), to derive the information as, \( I(X : Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) \). Here, we use the conditional entropy, \( H(X|Y) \), which measures the entropy remaining in a system or variable after it has been conditioned by another variable. The main goal of machine learning is to lower the entropy or uncertainty, which is the weighted average of the entropies of the various possible conditions or variables. The total entropy can be thought of as the remaining entropy or uncertainty: the lower the conditional entropy, the lower the remaining entropy!

As we saw in Section 1.9, there are many different types of entropies. Rényi Entropy generalizes these various entropies, including the Hartley entropy, the Shannon entropy, the collision entropy and the min-entropy: \( H_0, H_1, H_2, \) and \( H_{\infty} \) respectively (but it does not include the von Neumann or other quantum entropies). In general, entropy is at a maximum when the system is impure or maximally mixed or where all the outcomes have equal probability, as seen in Figure 10.22.

**Problem 10.3** Suppose you’ve been busy with this CS0 course and your room has descended into chaos and become a complete mess. You come in and clean it up so it’s neater than it’s ever been. Has the entropy of your room system gone up or down after you cleaned it up? Was information gained or not?

### 10.6.5 Entropy for Decision Trees

We know that entropy is related to the number of microstates: that’s why it’s often said to be a measure of disorder. Data might be similarly disordered or unorganized or impure; this means that entropy is like a measure of the variance or the impurity of the data. The goal of a lot of machine learning, like decision trees, is to reduce entropy which will decrease uncertainty and so increase information.

This means that information gain is equal to a decrease in entropy. This is also the amount of uncertainty an observation resolves. This then implies that the greater the Shannon Entropy, the greater the Information we could gain after observing the outcome of a probabilistic event. This is a way to quantify the potential reduction in uncertainty, the amount of information that’s lacking before observing the outcome of a probabilistic process. Since entropy can measure how impure or mixed some system is, we can use entropy in cases where we need to separate out parts of it, just as we do in decision trees, as we’ll see next.

### 10.7 Learning in Decision Trees

Now that we have all the mathematical and theoretical underpinnings, let’s revisit our original decision tree model from Section 10.2.3. Our first algorithm needed to be simple and intuitive and to allow us to clearly see the logic that leads to solution, as opposed to some esoteric algorithm that worked like a black box. The decision tree algorithm fit these criteria by creating a graph structure that looks like an upside down tree, as seen in Figure 10.4.\(^{32}\) This directed tree structure consists of a top, root node which branches to other, internal nodes via edges or links; the final, terminal level of nodes are called the leaves of the tree.

\(^{32}\) A decision tree, in general, is a non-linear classifier that is locally linear, piecewise linear in a neighbourhood, and can deal with both categorical (classification tree, binary or multi-class) and numerical values (regression tree). In fact, decision
Each of the nodes represents a different feature or attribute we deem to be informative; the branches represent the questions or decisions or rules associated with each feature; and the leaves represent the final outcome, which is usually a categorical answer to the question or the result of the decision/rule (it could also be a continuous value like a probability in the general case). Our overall goal is to use our dataset to end up with a single classification outcome.\(^\text{33}\)

Our approach will be to find characteristics, or features, in our data. We’ll map each feature to a node that will test that feature. Each outgoing branch from the node will correspond to a feature value and, at the end, each leaf node will assign a classification, or a probability distribution over the classifications. The decision tree will thus be a function that maps training examples from our dataset to one of the possible classes or categories. The path from the root node to leaf node contains all the training samples that will be used for the final outcome’s calculation where each branch is considered an AND operation.

The learning in this algorithm will consist of deciding how to build the tree, including the ordering of the features, represented as nodes, as well as the number of splits or branches that go out from each node. The algorithm thus learns the structure of the tree (e.g., the number of splits per node), the threshold values at each node, and the final values output by the leaves. These are the parameters.\(^\text{34}\)

Entropy itself for a random variable \(X\) is defined as:

\[
H(X) = -\sum_{i=1}^{N} p(X = x_i) \log_2(p(X = x_i))
\] (10.5)

So the total entropy of the Root Node is:

\[
H(X_{\text{root-node}}) = -p(X = \text{"lettuce"}) \log_2(p(X = \text{"lettuce"})) - p(X = \text{"carrots"}) \log_2(p(X = \text{"carrots"}))
\]

\[
- p(X = \text{"oranges"}) \log_2(p(X = \text{"oranges"}))
\]

\[
= -\frac{1}{4} \log_2\left(\frac{1}{4}\right) - \frac{1}{4} \log_2\left(\frac{1}{4}\right) - \frac{2}{4} \log_2\left(\frac{2}{4}\right)
\]

\[
= .25 \times 2 + .25 \times 2 + .5 \times 1
\]

\[
= 1.5
\]

For our first potential split, we ask the question, “Is it orange coloured?” To answer this question, we need to work out the class probabilities for Orange vs Not-Orange elements. We use upper case \(P(\cdot)\) for the class probabilities in this calculation:

---

\(^{\text{33}}\)We could also think in terms of minimizing the training error or maximizing the training accuracy instead of the single classification outcome. Each leaf node can predict its final outcome based on the average outcome of the training data in the nodes in that path. More precisely, each unique path from the Root node to a Leaf node defines a specific region, \(R\); the outcome is either set to the most common value for the training examples in \(R\) (for a classification tree) or their mean (for a regression tree).

\(^{\text{34}}\)The hyperparameters of the decision tree would be pruning parameters, either pre- or post-pruning, of a tree, like the maximum depth, maximum number of terminal nodes, and minimum samples for a node split. We can even create ensembles of multiple trees, either using boosting or bagging as in random forests. Each hypothesis in this approach, \(h\), is a decision tree and all the hypotheses constitute the set of function hypotheses, \(H = \{h : X \rightarrow Y\}\) where the output of the learning algorithm is the \(h \in H\) that best approximates the unknown target function, \(f\), based on the training examples from the dataset, by either minimizing the training error or maximizing training accuracy.
Figure 10.23: Decision Tree Example with Entropies for each node. The root node has 4 fruits and vegetables, while the decision node only has 3. The leaf nodes all have only one kind of fruit or vegetable. In each node, we calculate the entropy using the fraction of the total fruits and vegetables each kind of fruit or vegetable has in that particular node. E.g., in the Root Node, there are 4 fruits and vegetables with 2 oranges, 1 romaine lettuce, and 1 set of carrots. Hence, the fraction of carrots is \( \frac{1}{4} \), the fraction of oranges is \( \frac{2}{4} = 0.5 \), and the fraction of romaine lettuce is \( \frac{1}{4} \). The entropy is measured in units of bits; the information gained would be the difference between the entropies between two nodes.

\[
P(Y = \text{"orange"}) = \frac{2}{5} = 0.4
\]

\[
P(Y = \text{"not - orange"}) = \frac{3}{5} = 0.6
\]

Then, let’s compute the specific conditional entropy of the root node if we pick an Orange fruit or vegetable. This is the entropy of the remaining elements in the node after we’ve taken out the Orange elements; this is the same as the entropy of the first Decision Node in Figure 10.23. We can also compute the entropy of the “not-orange” part of the split and this would be the first leaf node on the left in Figure 10.23.

\[
H(X|Y = \text{"orange"}) = -p(X = \text{"oranges"})\log_2(p(X = \text{"oranges"})) - p(X = \text{"carrots"})\log_2(p(X = \text{"carrots"}))
\]

\[
= -\frac{2}{3}\log_2\left(\frac{2}{3}\right) - \frac{1}{3}\log_2\left(\frac{1}{3}\right)
\]

\[
= .67* .585 + .33 * 1.585 = 0.392 + 0.523
\]

\[
= 0.915 \approx 0.92
\]

\[
H(X|Y = \text{"not - orange"}) = -p(X = \text{"lettuce"})\log_2(p(X = \text{"lettuce"}))
\]

\[
= -\frac{1}{1}\log_2\left(\frac{1}{1}\right)
\]

\[
= 0
\]
Finally, we can compute the overall or general conditional entropy, which is composed of all the specific conditional entropies weighted by their respective probabilities, as:

\[
H(X|Y) = \sum_{i=1}^{N} P(Y = v_i)H(X|Y = v_i)
\]

\[
= P(Y = \text{"orange")}H(X|Y = \text{"orange")} + P(Y = \text{"not – orange")}H(X|Y = \text{"not – orange")}
\]

\[
= 0.4 \times 0.92 + 0.6 \times 0
\]

\[
= 0.368
\]

10.7.1 Sources of Error in Machine Learning

In supervised learning algorithms like decision trees, we can measure how well our machine learning model is performing by measuring the error in our model, which can also be thought of as how well it fits the problem and can be gauged using the signal to noise ratio. The signal of a dataset can be thought of as the underlying pattern we’re trying to discern while noise might be considered to be the randomness or irrelevant parts of the dataset. Generalization is the idea of how well a model can predict the signal on new data after being trained on known data.

Overfitting is when a model learns too much from the known, training data and is so customized to the known data that it performs poorly when faced with new, unseen data. An overfit model is usually too complex, with too many parameters or features, and starts capturing the noise and outliers in the training data. As such, an overfit model will perform well on training data but poorly on testing data. This often occurs with nonparametric and nonlinear models, like neural networks, SVMs, and decision trees, which usually contain parameters or techniques to limit how much detail the model learns.

Underfitting occurs when a model is too simple and doesn’t learn enough from the known, training data and so cannot generalize well on new data, either. An underfit model usually hasn’t seen enough data and might be too simple; in fact, it cannot even model the training data well and performs poorly in training. As such, it is easy to detect by using an appropriate performance metric and can be mitigated by increasing complexity or trying multiple machine learning algorithms. Simple models that cannot capture complex patterns in the data, like linear regression and logistic regression, often exhibit underfitting.

Although both underfitting and overfitting can be mitigated by trying multiple models, using cross-validation, or incorporating early stopping for algorithms with multiple iterations, overfitting is usually a more common problem in machine learning. We can help reduce the effects of overfitting by using regularization, which helps reduce the complexity of the model, and ensemble learning, which combines multiple algorithms. Underfitting and overfitting also exhibit a bias-variance tradeoff, where an increase in one results in a decrease in the other so a lot of machine learning ends up tweaking the model to get a reasonable balance between underfitting and overfitting.

Variance error is how sensitive the model is to small fluctuations in the dataset. This is the error due to how much a predicted value varies for a given data point. High variance is usually due to an overly-complex model that fits the training data too closely. Bias error usually occurs when not all the features are taken into account and measures how far off the predicted values are from the actual values. Bias error is thus the error that arises from incorrect assumptions in the algorithm, usually from a model with overly-simplistic or faulty assumptions that cause it to miss important trends in the dataset’s features. Bias error can also be thought of as not being able to capture all of the signal in a dataset while variance error can be thought of as capturing too much of the noise in a dataset.
Models that overfit usually have high variance and low bias whereas underfitting models exhibit low variance and high bias. There is also an irreducible error, which is the noise that might be present regardless of which particular algorithm is used since it depends on things like missing variables, measurement imperfections, unknown forces, etc. Sometimes, data cleaning can help reduce this somewhat but usually there’s some effect from such noise that cannot be completely eliminated. The total error can thus be expressed as:

\[
\text{Total Error} = \text{Bias Error}^2 + \text{Variance Error} + \text{Irreducible Error}
\]

10.8 Machine Learning, Data Science, and Computational Thinking

Let’s bring it all together in the end by going back to the beginning. As shown below, the same guiding principle of Data → Information → Knowledge that has led us from the very beginning of our adventure also provides a framework for the various fields that we’ve considered on our computational voyage so far. It culminates with the computational thinking principles like abstraction, pattern recognition, generalization, etc., being explicitly reflected in the final Machine Learning case.

In that final component, we also relate it to the sub-fields of Machine Learning (Exploratory Data Analysis, Data Analysis, and Data Science) and Business Intelligence/Business Analytics (Descriptive Analytics, Predictive Analytics, and Prescriptive Analytics). It’s been quite a journey but the lands that lie over the horizon as you delve into these areas in depth and discover new frontiers are even more exciting. With this framework in your toolbox, you’re ready to embark on these exciting new adventures... to quote another adventurer, “Excelsior!”

**Data → Information → Knowledge:** The final Data → Information → Knowledge paradigm details for the General, Fundamental Science, Information Theory, Database, and Machine Learning cases.
Bibliography


Index

Abstract Data Type, 142, 195
Abstract Data Type (ADT), 135
Characteristics, 136
Operations, 136
Abstraction, 60, 191, 203, 204
Data Abstraction, 191
Procedural Abstraction, 191
Accuracy, 59, 232, 268
ACID, 218
Alan Kay, 169
Algorithm, 12, 13, 91
Algorithmic Thinking, 29, see Thinking
American Standard Code for Information
Interchange (ASCII), 112
Application Programming Interface (API), 191
Public Interface, 191
Artificial Intelligence, 227
General AI, 227
Hard AI, 227
Narrow AI, 227
Soft AI, 227
Strong AI, 227
Weak AI, 227
Assignment Operator, 114
Bayes Theorem, 262
Evidence, 262
Likelihood, 262
Prior, 262
Bayesian Inference, 260
Bayesian Statistics, 260
Big Data, 247
Semi-structured Data, 247
Structured Data, 247
Unstructured Data, 247
Binary Files, 163
Bookkeeping Tool, 48, 275
Boolean Algebra, 68, 69
Breadth-First, 81
Bugs, 100
Business Analytics, 255
Business Intelligence, 255, 256
Business Rules, 210
Byte, 112
Changeable Program, 91
Charles Babbage, 92
Church-Turing Thesis, 16
Comment, 114
Compiler, 105
Computable, 91
Computation, 12, 91
  Computable, 8
  Computable Functions, 12, 14
  Effective Procedure, 12
  Turing Complete, 29
  Un-Computable, 18
Computational Thinking, 21, *see* Thinking, 76
  Abstraction, 78
  Approach, 76
  Basics, 96
Computational Problem Solving, 78
  Ad Hoc Thinking, 80
  Bottom-Up, 80
  Brute Force, 80
  Deductive Thinking, 80
  Exhaustive Search, 80
  Inductive Thinking, 80
  Intractable, 80
  Strategies, 80
  Top-Down, 80
Decomposition, 78
Generalization, 78
Pattern Recognition, 78
Computer Information Systems, 73
Computing Agent, 13
Computing Machines, 92
  Babbage Analytical Engine, 92
  Babbage Difference Engine, 92
  COLOSSUS, 92
  EDVAC, 92
  ENIAC, 92
  Greek Antikythera, 92
  Harvard Mark I, 92
  Incan Quipus, 92
  Mesopotamian Abacus, 92
  Programmable Loom, 92
  Z1, 92
Confusion Table, 268
Control Structures, 123
  Böhm Jacopini, 123
  Control Flow Graphs (CFGs), 123
  Iteration, 123
  Repetition, 123
  Selection, 123
Sequence, 123
Structured Program Theorem, 123
Data, 3, 45
  Structured Data, 203
Data Analysis, 218, 256
Data Lake, 218
Data Mart, 218
Data Mining, 218
Data Model, 142, 195, 206, 214, 218
Data Representations, 62
Data Science, 256
Data Structure, 137, 138, 142, 195
Data Structures, 62
Data Type, 112, 137, 142, 195
Data Warehouse, 218
Database, 206
  Artificial Key, 220
  Attribute, 215
  Composite Key, 220
  Composite Primary Key, 220
  Data Type, 215
  Entity, 215
  Feature, 215
  Fields, 208
  Foreign Key, 214, 215
  Indexed, 215
  Instance, 208, 215
  Natural Key, 220
  Primary Key, 214, 215
  Relations, 215
  Schema, 208, 215
  Sub-Schema, 208
  Table, 208, 215
  Views, 208
  Virtual Table, 208
DataBase Life Cycle (DBLC), 210
DataBase Management System (DBMS), 208
Database Model, 214, 218
DBMS Catalog, 218
Debugger, 98
Decision Tree, 234, 241, 279
  Random Forest, 239
DecisionTree
  Random Forest, 235
Declarative programming, 104
INDEX

Depth-First, 81
Descriptive Analytics, 255
Diagnostic Analytics, 255
Digital Circuits, 69
Digital Computers, 91
Digital Logic Circuits, 71
Discrete Event Simulation, 169

Editor, 98
Embedded SQL, 220
  Call Level Interface (CLI), 220
  Statement Level Interface (SLI), 220
Encapsulation, 191
Energy, 56
Ensemble Methods, 239
  Bagging, 239
  Boosting, 239
  BootStrapping, 239
  Stacking, 239
Entities, 204
Entity Relationship Diagrams (ERDs), 210
  Cardinality, 210
  Constraints, 210
Entropy, 206, 271
Error, 282
  Bias Error, 282
  Bias-Variance Tradeoff, 282
  Irreducible Error, 282
  Noise, 282
  Overfitting, 282
  Signal, 282
  Underfitting, 282
  Variance Error, 282
Errors
  Compile-Time, 101
  Exception, 101
  Exception Handling, 101
  Logical, 101
  Run-Time, 101
  Semantic, 101
  Syntax, 101
Exception Handling, 158, 198
  Catch, 158
  Checked Exceptions, 158, 198
  Compile-Time, 158
  Defensive Programming, 158

Error, 198
  Raise, 158
  Run-Time, 158
  RuntimeException, 198
  Throwable, 198
  Unchecked Exceptions, 158, 198
Exploratory Data Analysis (EDA), 245
Exploratory Data Analysis (EDA), 255
Expressions, 112
Extract, Transform, Load (ETL), 218

F1-Score, 268
Firmware, 65
Flat Files, 214
Flip-Flops, 69
Floating Point, 114
FooBar, 154, 180
Formal Languages, 104
Function, 30, 33
Functional programming, 104
Functions, 8, 11, 112
  Domain, 8
  Functional Notation, 11
  Range, 8
Halting Problem, 18
Hardware, 65, 95
  Accumulator, 96
  Address Bus, 95
  ALU, 95
  Command Wire, 95
  CPU, 95
  Data Bus, 95
  Fetch-Execute Cycle, 95
  RAM, 95
  Register, 96
  Harvard Architecture, 74
Hello World, 99
Hypothesis Testing, 260

Identifier, 114
Imperative Language, 104
Imperative Programming
  Modular Programming, 173
  Object Oriented programming, 173
  Procedural programming, 173
  Structured programming, 173
<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Imperative programming</td>
<td>173</td>
</tr>
<tr>
<td>Information</td>
<td>4, 45</td>
</tr>
<tr>
<td>Information Gain</td>
<td>206</td>
</tr>
<tr>
<td>Information Hiding</td>
<td>191</td>
</tr>
<tr>
<td>Information Processing</td>
<td>72</td>
</tr>
<tr>
<td>Information Technology</td>
<td>73</td>
</tr>
<tr>
<td>Information Theory</td>
<td>38</td>
</tr>
<tr>
<td>Bit</td>
<td>44</td>
</tr>
<tr>
<td>Claude Shannon</td>
<td>39</td>
</tr>
<tr>
<td>Conditional Entropy</td>
<td>46</td>
</tr>
<tr>
<td>Hartley Information Entropy</td>
<td>46</td>
</tr>
<tr>
<td>Hartley's Information Theory</td>
<td>41</td>
</tr>
<tr>
<td>Information</td>
<td>44</td>
</tr>
<tr>
<td>Ralph Hartley</td>
<td>39</td>
</tr>
<tr>
<td>Shannon Information</td>
<td>42</td>
</tr>
<tr>
<td>Shannon Information Entropy</td>
<td>46</td>
</tr>
<tr>
<td>Stochastic Process</td>
<td>42</td>
</tr>
<tr>
<td>Unit of Information</td>
<td>41</td>
</tr>
<tr>
<td>Inheritance</td>
<td>191</td>
</tr>
<tr>
<td>Instruction Set</td>
<td>96</td>
</tr>
<tr>
<td>Complex Instruction Set Computer (CISC)</td>
<td>96</td>
</tr>
<tr>
<td>One Instruction Set Computer (OISC)</td>
<td>96</td>
</tr>
<tr>
<td>Reduced Instruction Set Computer (RISC)</td>
<td>96</td>
</tr>
<tr>
<td>Interpreter</td>
<td>105</td>
</tr>
<tr>
<td>Jacquard Loom</td>
<td>92</td>
</tr>
<tr>
<td>Knowledge</td>
<td>5, 45</td>
</tr>
<tr>
<td>Konrad Zuse</td>
<td>92</td>
</tr>
<tr>
<td>Lady Ada Lovelace</td>
<td>92</td>
</tr>
<tr>
<td>Lists</td>
<td>141</td>
</tr>
<tr>
<td>Literals</td>
<td>112</td>
</tr>
<tr>
<td>Logic</td>
<td></td>
</tr>
<tr>
<td>Argumentation</td>
<td>26</td>
</tr>
<tr>
<td>Aristotelian Categories</td>
<td>26, 175</td>
</tr>
<tr>
<td>Formal Logic</td>
<td>25, 26</td>
</tr>
<tr>
<td>Predicate</td>
<td>26</td>
</tr>
<tr>
<td>Predicate Logic</td>
<td>26</td>
</tr>
<tr>
<td>Propositional Logic</td>
<td>26</td>
</tr>
<tr>
<td>Syllogisms</td>
<td>26</td>
</tr>
<tr>
<td>Truth Table</td>
<td>26</td>
</tr>
<tr>
<td>Logic Gate</td>
<td>67</td>
</tr>
<tr>
<td>Logic Gates</td>
<td>69</td>
</tr>
<tr>
<td>AND</td>
<td>69</td>
</tr>
<tr>
<td>NAND</td>
<td>69</td>
</tr>
<tr>
<td>NOT</td>
<td>69</td>
</tr>
<tr>
<td>OR</td>
<td>69</td>
</tr>
<tr>
<td>Logical</td>
<td>204</td>
</tr>
<tr>
<td>Logical Data Independence</td>
<td>210</td>
</tr>
<tr>
<td>Machine Learning</td>
<td>229, 243</td>
</tr>
<tr>
<td>Accuracy Paradox</td>
<td>253</td>
</tr>
<tr>
<td>Bias-Variance Tradeoff</td>
<td>253</td>
</tr>
<tr>
<td>Classification</td>
<td>232, 241</td>
</tr>
<tr>
<td>Cross-Validation</td>
<td>253</td>
</tr>
<tr>
<td>Curse of Dimensionality</td>
<td>251</td>
</tr>
<tr>
<td>Data Cleaning</td>
<td>232</td>
</tr>
<tr>
<td>Data Mining</td>
<td>247</td>
</tr>
<tr>
<td>Data Munging</td>
<td>232</td>
</tr>
<tr>
<td>Deep Learning</td>
<td>247</td>
</tr>
<tr>
<td>Dependent Variable</td>
<td>232</td>
</tr>
<tr>
<td>Dimensionality</td>
<td>232, 241</td>
</tr>
<tr>
<td>Feature Engineering</td>
<td>232, 253</td>
</tr>
<tr>
<td>Feature Selection</td>
<td>232, 253</td>
</tr>
<tr>
<td>Final Model</td>
<td>241</td>
</tr>
<tr>
<td>Final Prediction</td>
<td>241</td>
</tr>
<tr>
<td>Generalizing</td>
<td>251</td>
</tr>
<tr>
<td>Hyperparameters</td>
<td>241</td>
</tr>
<tr>
<td>Independent Variable</td>
<td>232</td>
</tr>
<tr>
<td>Instances</td>
<td>232</td>
</tr>
<tr>
<td>k-Means</td>
<td>241, 243</td>
</tr>
<tr>
<td>Label</td>
<td>232</td>
</tr>
<tr>
<td>Learn</td>
<td>232</td>
</tr>
<tr>
<td>Learned Model</td>
<td>235</td>
</tr>
<tr>
<td>Model</td>
<td>232</td>
</tr>
<tr>
<td>Observations</td>
<td>232</td>
</tr>
<tr>
<td>Overfitting</td>
<td>251</td>
</tr>
<tr>
<td>Parameters</td>
<td>253</td>
</tr>
<tr>
<td>Pre-Processing</td>
<td>232</td>
</tr>
<tr>
<td>Predictions</td>
<td>232</td>
</tr>
<tr>
<td>Predictor Variables</td>
<td>232</td>
</tr>
<tr>
<td>Samples</td>
<td>232</td>
</tr>
<tr>
<td>Statistics</td>
<td>247</td>
</tr>
<tr>
<td>Supervised Learning</td>
<td>232, 241</td>
</tr>
<tr>
<td>Target Variable</td>
<td>232</td>
</tr>
<tr>
<td>Testing</td>
<td>235</td>
</tr>
<tr>
<td>Testing Dataset</td>
<td>241</td>
</tr>
<tr>
<td>Training</td>
<td>235</td>
</tr>
<tr>
<td>Training Dataset</td>
<td>241</td>
</tr>
<tr>
<td>Unseen Dataset</td>
<td>241</td>
</tr>
</tbody>
</table>
Procedural programming, 104
Procedure, 12, 14
Procedures, see Sub-routines
Program, 91
Programming Languages, 21
  Binary, 76
  Declarative Programming Languages, 33
  Dynamically-Typed, 114
  Fifth Generation Language, 76
  First Generation Language, 76
  Fourth Generation Language, 76
  Functional Programming, 37
  High-Level Languages, 76
  Imperative Programming, 37
  Imperative Programming Languages, 33
  Machine Code, 76
  Machine Language, 76
  Second Generation Language, 76
  Strongly-Typed, 114
  Third Generation Language, 76
Project Triangle, 210
Pseudocode, 34, 35, 173
Python
  Dot Operator, 107
  Libraries, 107
  Modules, 107
Quantum Computer, 56
Query, 208
Query Language, 208
Random Variable, 46
Random Variables, 271
  Cumulative Distribution Function, 271
  Event Space, 271
  Outcome, 271
  Probability Density Function, 271
  Probability Distribution, 271
  Probability Distribution Function, 271
  Probability Mass Function, 271
  Sample Space, 271
Relation, 218
Relational Database Model, 213, 214, 218
Sampling
  Random Sampling, 24
  Sampling With Replacement, 24
  Stratified Sampling, 24
  Schema, 218
  Script, 99
  Self-Documenting Code, 180
  Set, 8
    n-tuple, 8
    Ordered Set, 8
    Sequence, 8
    Series, 8
    Vector, 8
  Shannon Information, 271
  Shannon Information Entropy, 271
  Side effects, 173
  SIMULA, 169
  Smalltalk, 169
  Software, 65, 95
  Software Engineering, 74
  Source Code, 14, 104
  SQL
    Ad-Hoc Query, 213
    State, 30, 33, 81, 103, 175
    State Space, 33
    State Variable, 33
    State Space, 81
    State Space Graph, 81
    Statements, 112
  Streams, 161
    Standard Error (STDERR), 118, 161
    Standard Input (STDIN), 118, 161
    Standard Output (STDOUT), 118, 161
  Strings, 139
  Structured programming, 104
  Structured Query Language (SQL), 210
Sub-routine, 147, 173
  Call, 150
  Flow of Execution, 155
  Functions, 147, 173
  Invoke, 150
  Procedures, 147, 173
Switch, 67
Symbol Table, 114
System Development Life Cycle (SDLC), 210
Systems Analyst, 74
Theory
  Context-Free Grammar, 71
<table>
<thead>
<tr>
<th>INDEX</th>
<th>293</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finite State Automata, 71</td>
<td></td>
</tr>
<tr>
<td>Formal Language Theory, 71</td>
<td></td>
</tr>
<tr>
<td>Grammar, 71</td>
<td></td>
</tr>
<tr>
<td>Regular Language, 71</td>
<td></td>
</tr>
<tr>
<td>Thinking</td>
<td></td>
</tr>
<tr>
<td>Computational Thinking, 30</td>
<td></td>
</tr>
<tr>
<td>Abstraction, 30</td>
<td></td>
</tr>
<tr>
<td>Algorithmic Expression, 30</td>
<td></td>
</tr>
<tr>
<td>Algorithmic Thinking, 30</td>
<td></td>
</tr>
<tr>
<td>Models, 30</td>
<td></td>
</tr>
<tr>
<td>Pattern Recognition, 30</td>
<td></td>
</tr>
<tr>
<td>Problem Decomposition, 30</td>
<td></td>
</tr>
<tr>
<td>Problem Specification, 30</td>
<td></td>
</tr>
<tr>
<td>Solution Evaluation, 30</td>
<td></td>
</tr>
<tr>
<td>Solution Implementation, 30</td>
<td></td>
</tr>
<tr>
<td>Declarative Statements, 28</td>
<td></td>
</tr>
<tr>
<td>Deductive Reasoning, 22</td>
<td></td>
</tr>
<tr>
<td>Imperative Statements, 28</td>
<td></td>
</tr>
<tr>
<td>Inductive Reasoning, 22</td>
<td></td>
</tr>
<tr>
<td>Probabilistic Inference, 22</td>
<td></td>
</tr>
<tr>
<td>Translation, 74</td>
<td></td>
</tr>
<tr>
<td>Compilation, 74</td>
<td></td>
</tr>
<tr>
<td>Interpretation, 74</td>
<td></td>
</tr>
<tr>
<td>Translator, 98, 105</td>
<td></td>
</tr>
<tr>
<td>Tuples, 141</td>
<td></td>
</tr>
<tr>
<td>Comparable, 142</td>
<td></td>
</tr>
<tr>
<td>Hashable, 142</td>
<td></td>
</tr>
<tr>
<td>Turing Machine, 83</td>
<td></td>
</tr>
<tr>
<td>Turing Machines, see Universal Turing Machine, 16, 81, 227</td>
<td></td>
</tr>
<tr>
<td>Type, 111, 135, 142, 195</td>
<td></td>
</tr>
<tr>
<td>Typecasting, 114</td>
<td></td>
</tr>
<tr>
<td>Unified Modeling Language (UML), 193</td>
<td></td>
</tr>
<tr>
<td>Universal Turing Machine, 14</td>
<td></td>
</tr>
<tr>
<td>Use Cases, 206</td>
<td></td>
</tr>
<tr>
<td>Values, 3</td>
<td></td>
</tr>
<tr>
<td>Variable, 30, 33</td>
<td></td>
</tr>
<tr>
<td>State Variable, 33</td>
<td></td>
</tr>
<tr>
<td>Variables, 11, 112</td>
<td></td>
</tr>
<tr>
<td>Dependent Variable, 11</td>
<td></td>
</tr>
<tr>
<td>Global, 155</td>
<td></td>
</tr>
<tr>
<td>Independent Variable, 11</td>
<td></td>
</tr>
<tr>
<td>Lifetime, 155</td>
<td></td>
</tr>
<tr>
<td>Local, 155</td>
<td></td>
</tr>
<tr>
<td>Namespace, 155</td>
<td></td>
</tr>
<tr>
<td>Scope, 155</td>
<td></td>
</tr>
<tr>
<td>Virtual Machine, 105</td>
<td></td>
</tr>
<tr>
<td>Virtual Machine Interpreter, 105</td>
<td></td>
</tr>
<tr>
<td>von Neumann Architecture, 74</td>
<td></td>
</tr>
<tr>
<td>Work, 56</td>
<td></td>
</tr>
</tbody>
</table>
About the author

Ricky J. Sethi is an Associate Professor of Computer Science at Fitchburg State University. Ricky is also Director of Research for the Madsci Network and Team Lead for SNHU Online at Southern New Hampshire University.

Prior to that, he was a Research Scientist at UMass Amherst/U-Mass Medical School and at UCLA/USC Information Sciences Institute, where he was chosen as an NSF Computing Innovation Fellow (CIFellow) by the CCC and the CRA. Even earlier, he was a Post-Doctoral Scholar at UCR, where he was the Lead Integration Scientist for the WASA project and participated in ONR’s Empire Challenge 10.

Ricky has authored or co-authored over 30 peer-reviewed papers, book chapters, and reports and made numerous presentations on his research in machine learning, computer vision, social computing, and data science.

Ricky’s work has been generously funded by the National Science Foundation (NSF), the National Endowment for the Humanities (NEH), the Institute for Advanced Study (IAS), and Amazon. He has also served as a panelist on several NSF programs, an Editorial Board Member for the International Journal of Computer Vision & Signal Processing, and an organizer and program committee member for various conferences.

You can find out more at his website http://research.sethi.org/ricky/