



Interpreting Machine Learning Models

Learn Model Interpretability and
Explainability Methods

Anirban Nandi
Aditya Kumar Pal



Interpreting Machine Learning Models

**Learn Model Interpretability
and Explainability Methods**

**Anirban Nandi
Aditya Kumar Pal**

Apress®

Interpreting Machine Learning Models: Learn Model Interpretability and Explainability Methods

Anirban Nandi
Bangalore, India

Aditya Kumar Pal
Bangalore, India

ISBN-13 (pbk): 978-1-4842-7801-7
<https://doi.org/10.1007/978-1-4842-7802-4>

ISBN-13 (electronic): 978-1-4842-7802-4

Copyright © 2022 by Anirban Nandi and Aditya Kumar Pal

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Celestin Suresh John
Development Editor: James Markham
Coordinating Editor: Shrikant Vishwakarma
Copyeditor: Kim Burton Wiseman

Cover designed by eStudioCalamar

Cover image designed by Pexels

Distributed to the book trade worldwide by Springer Science+Business Media LLC, 1 New York Plaza, Suite 4600, New York, NY 10004. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at <http://www.apress.com/bulk-sales>.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-7801-7. For more detailed information, please visit <http://www.apress.com/source-code>.

Printed on acid-free paper

*I would like to dedicate this book to my wife Madhuparna,
who pushes me to “aim for the moon” and then outdo
my achievements, and my family members and friends who always
guide and support me during the good and the difficult times*

—Anirban

*I would like to dedicate this book to my sister Rati,
who constantly motivates me to work hard and
never give up*

—Aditya

Table of Contents

About the Authors.....	xvii
About the Technical Reviewers	xix
Acknowledgments	xxi
Introduction	xxiii
Chapter 1: The Evolution of Machine Learning	1
Defining Machine Learning	1
The Evolution of Machine Learning.....	2
Learning a Machine Learning Algorithm	4
Piece It Together.....	4
Focus on Specific Algorithm Descriptions.....	5
Design an Algorithm Description Template.....	5
Start Small and Build It Up	6
Investigating Machine Learning Algorithm Behavior	7
Step 1. Select an Algorithm	7
Step 2. Identify a Question	7
Step 3. Design the Experiment.....	8
Step 4. Execute the Experiment and Report Results	8
Step 5. Repeat.....	9
What Does Machine Learning Model Accuracy Mean?	9
Why Model Accuracy Is Not Enough.....	10
Summary.....	14

TABLE OF CONTENTS

Chapter 2: Introduction to Model Interpretability.....	15
Humans Are Explanation Hungry	15
Explanations in Machine Learning	17
What Are Black-Box Models?.....	21
What Is Interpretability?.....	24
The Motivation Behind Interpretability.....	26
To Make Better Decisions	26
To Eliminate Bias	27
To Justify Processes.....	27
To Reproduce Operations	28
Displacement Strategy	28
To Determine Practical Accuracy.....	28
To Maintain Privacy	29
To Understand Security Risks.....	29
The Research Behind Interpretability.....	29
Summary.....	34
Chapter 3: Machine Learning Interpretability Taxonomy	35
Scope-related Types of Post hoc Model Interpretability.....	37
Global Model Interpretability on a Holistic Level	37
Local Model Interpretability.....	38
A Group of Predictions	39
Model-related Types of Post hoc Model Interpretability.....	39
Result-related Types of Post hoc Model Interpretability.....	40
Categorizing Common Classes of Explainability Methods.....	41
Summary.....	42

TABLE OF CONTENTS

Chapter 4: Common Properties of Explanations Generated by Interpretability Methods.....	45
Explanation Defined	46
Properties of Explanation Methods	47
Template of Expression	47
Transparency	47
Mobility.....	48
Algorithmic Feasibility.....	48
Properties of Individual Explanations.....	48
Correctness	48
Loyalty	49
Dependability.....	49
Resoluteness	49
Lucidness	49
Reliability.....	50
Significance.....	50
Originality	50
Representativeness	50
Human-Friendly Explanations	51
Contrastiveness	51
Selectivity.....	51
Social.....	52
Focus on the Abnormal.....	52
Truthful	52
Consistent with Prior Beliefs	52
General and Probable	53
Summary.....	53

TABLE OF CONTENTS

Chapter 5: Human Factors in Model Interpretability	55
Interpretability Roles.....	55
Technical Expertise Builders.....	56
Domain Knowledge Reviewers.....	56
Stakeholders or End Users	56
Interpretability Stages.....	57
Ideation and Conceptualization Stage	57
Building and Validation Stage.....	58
Deployment, Maintenance, and Use Stage	59
Interpretability Goals.....	61
Interpretability for Model Validation and Improvement	61
Interpretability for Decision-Making and Knowledge Discovery	62
Interpretability to Gain Confidence and Obtain Trust.....	62
Human-Friendly Themes Characterizing Interpretability Work	63
Interpretability Is Cooperative	63
Interpretability Is Process.....	64
Interpretability Is a Mental Model Comparison.....	65
Interpretability Is Context-Dependent.....	65
Design Opportunities for Interpretability Challenges	65
Identifying, Representing, and Integrating Human Expectations.....	65
Communicating and Summarizing Model Behavior.....	66
Scalable and Integrable Interpretability Tools	66
Post-Deployment Support	67
Summary.....	67
Chapter 6: Explainability Facts: A Framework for Systematic Assessment of Explainable Approaches.....	69
Explainability Facts List Dimensions.....	70
Functional Requirements	71
F1: Problem Supervision Level	71
F2: Problem Type	71

TABLE OF CONTENTS

F3: Explanation Target	71
F4: Explanation Breadth/Scope	72
F5: Computational Complexity	72
F6: Applicable Model Class.....	72
F7: Relation to the Predictive System.....	72
F8: Compatible Feature Types.....	73
F9: Caveats and Assumptions.....	73
Operational Requirements	73
01: Explanation Family	73
02: Explanatory Medium.....	74
03: System Interaction	74
04: Explanation Domain.....	75
05: Data and Model Transparency	75
06: Explanation Audience	75
07: Function of the Explanation.....	76
08: Causality vs. Actionability.....	76
09: Trust vs. Performance	76
Usability Requirements	76
U1: Soundness.....	77
U2: Completeness.....	77
U3: Contextfullness.....	77
U4: Interactiveness	77
U5: Actionability.....	78
U6: Novelty.....	78
U7: Complexity.....	78
U8: Personalization.....	79
Safety Requirements.....	79
S1: Information Leakage.....	79
S2: Explanation Misuse	80
S3: Explanation Invariance	80

TABLE OF CONTENTS

Validation Requirements	80
Summary.....	82
Chapter 7: Interpretable ML and Explainable ML Differences	83
Interpretable ML and Explainable ML Basics	83
Analyzing the Decision Tree.....	84
Digging Deeper	85
Key Issues with Explainable ML.....	87
Trade-offs Between Accuracy and Interpretability	87
Beware of the Unfaithful.....	88
Not Enough Detail.....	88
Key Issues with Interpretable ML.....	89
Profits vs. Losses.....	89
Efforts to Construct.....	90
Hidden Patterns.....	90
Explanatory and Predictive Modeling.....	90
Explaining or Predicting: The Key Differences Between Two Choices	91
Validation, Model Evaluation, and Model Selection.....	93
Validation.....	93
Model Selection.....	94
Model Use and Reporting Explanatory Models.....	94
Summary.....	95
Chapter 8: The Framework of Model Explanations.....	97
Data Sets at a Glance.....	97
Types of Frameworks for Tabular Data.....	99
Feature Importance (FI)	99
Predictive Power of Feature Subsets	100
Additive Importance Measures.....	102
Removal-based Explanations for Feature Importance.....	103
Feature Removal.....	106

TABLE OF CONTENTS

Explaining Different Model Behaviors	107
Summarizing Feature Influence	107
Rule-based Explanations	108
Prototypes	109
Counterfactuals	110
Explanations for Image Data	111
Saliency Maps	111
Concept Attribution	112
Text Data.....	112
Sentence Highlighting	113
Attention-based Methods	113
Summary.....	114
Chapter 9: Feature Importance Methods: Details and Usage Examples	117
Data Set Name	117
Abstract.....	117
Sources	118
Data Set Information	118
Attribute Information.....	118
Random Forest Feature Importance	134
Accuracy-based Importance.....	135
Gini-based Importance	135
Permutation Feature Importance	137
Advantages.....	139
Disadvantages	139
Code	139
SHAP	141
Property 1 (Local Accuracy).....	148
Property 2 (Missingness).....	148
Property 3 (Consistency).....	148

TABLE OF CONTENTS

SAGE	157
How SHAP and SAGE Are Related.....	161
LIME	163
FACET.....	170
Model Inspection	171
Model Simulation.....	171
Enhanced Machine Learning Workflow	171
Code	172
Synergy.....	175
Redundancy.....	177
Partial Dependence Plots (PDP)	180
Code	184
Individual Conditional Expectation	185
DALEX.....	186
Introduction to Instance-level Exploration.....	189
Breakdown Plots for Additive Attributions	189
Breakdown Plots for Interactions	191
Ceteris Paribus Profiles	192
Local Diagnostics Plots.....	194
Implementation Example of DALEX on the <i>Titanic</i> Data Set	194
Create a Pipeline Model.....	196
Predict-level Explanations.....	198
Model-level Explanations	204
Summary.....	209
Chapter 10: Detailing Rule-Based Methods.....	211
MAGIE (Model-Agnostic Global Interpretable Explanations).....	212
MAGIE Algorithm Approach	215
Preprocessing the Input Data	216
Generating Instance Level Conditions	216
Learning Rules from Conditions	216

TABLE OF CONTENTS

Postprocessing Rules	218
Sorting Rules by Mutual Information	218
GLocaLX	221
Local to Global Explanation Problem	221
Local to Global Hierarchy of Explanation Theories	222
Finding Similar Theories.....	223
Code	226
Output.....	230
Skope-Rules.....	233
Methodology	234
Implementation	235
Anchors.....	238
Finding Anchors.....	241
Advantages.....	243
Disadvantages	244
Getting an Anchor	245
Summary.....	247
Chapter 11: Detailing Counterfactual Methods.....	249
Counterfactual Explanations	249
Use Case 1: Banking Software	250
Use Case 2: Continuous Outcome	251
Counterfactual Explanations at a Glance	251
Generating Counterfactual Explanations.....	252
Counterfactual Guided by Prototypes	257
DiCE	257
MOC (Multi-Objective Counterfactuals)	258
Comparison Between the Algorithms.....	258

TABLE OF CONTENTS

DICE.....	260
Diversity and Feasibility Constraints	261
Proximity	261
Sparsity	261
Optimization	262
Advantages.....	262
Disadvantages.....	263
Summary.....	269
Chapter 12: Detailing Image Interpretability Methods	271
Image Interpretation Using LIME.....	272
Step 1. Generate Random Perturbations for Input Image.....	274
Step 2. Predict Class for Perturbations	276
Step 3. Compute Weights (Importance) For the Perturbations	277
Step 4. Fit an Explainable Linear Model Using the Perturbations, Predictions, and Weights	277
Image Interpretation Using Pixel Attribution (Saliency Maps).....	279
Image Interpretation Using Class Activation Maps.....	280
Step 1. Modify the Model.....	281
Step 2. Retrain the Model with CAMLogger Callback.....	283
Step 3. Use CAMLogger to See the Class Activation Map.....	283
Step 4. Draw Conclusions from the CAM.....	284
Image Interpretation Using Gradient-Weighted Class Activation Maps.....	286
Summary.....	293
Chapter 13: Explaining Text Classification Models.....	295
Data Preprocessing, Feature Engineering, and Logistic Regression Model on the Data.....	296
Interpreting Text Predictions with LIME	298
Interpreting Text Predictions with SHAP	302
Explaining Text Models with Sentence Highlighting	306
Summary.....	313

TABLE OF CONTENTS

Chapter 14: The Role of Data in Interpretability	315
Summary.....	320
Chapter 15: The Eight Pitfalls of Explainability Methods.....	321
Assuming One-Fits-All Interpretability	321
Bad Model Generalization	322
Unnecessary Use of Complex Models	322
Ignoring Feature Dependence.....	323
Interpretation with Extrapolation.....	323
Confusing Linear Correlation with General Dependence	324
Misunderstanding Conditional Interpretation	324
Misleading Interpretations Due to Feature Interactions.....	324
Misleading Feature Effects Due to Aggregation	325
Failing to Separate Main from Interaction Effects.....	325
Ignoring Model and Approximation Uncertainty.....	325
Failure to Scale to High-Dimensional Settings	326
Human-Intelligibility of High-Dimensional IML Output	326
Computational Effort.....	327
Unjustified Causal Interpretation.....	327
Summary.....	328
Conclusion	329
Engage Interpretability with a Good Plan.....	329
User Experience and Interpretability Go Hand in Hand.....	329
References.....	331
Index.....	333

About the Authors



With close to 15 years of professional experience, **Anirban Nandi** specializes in Data Science, Business Analytics and Data Engineering spanning across various business verticals and building teams from the ground up. After receiving his master's degree in economics from Jawaharlal Nehru University, Anirban started his career at a US-based multi-channel retailer and spent more than eight years developing in-house products like customer personalization recommendation and search engine classifiers. After that, Anirban became one of the founding Data Science and Analytics members for an organization headquartered in the UAE and spent several years building the onshore and offshore team working on assortment, inventory, pricing, marketing, e-commerce, and customer analytics solutions.

Currently, Anirban is associated with Rakuten India as the Head of Analytics developing Data Science and Analytics solutions for the Rakuten global ecosystem across different domains in commerce, fintech, and telecommunication. He is also involved in building scalable AI products that can support the data-driven decision-making culture of the Rakuten global ecosystem.

Anirban's interests include learning about new technologies and disruptive startups. In his spare time, he loves networking with people. Anirban loves sports and is a big follower of soccer/football (Argentina and Manchester United are his favorite teams).

You can reach him by email at aninandi1983@gmail.com and on LinkedIn at www.linkedin.com/in/anirban-nandi-89a36ab7/.

ABOUT THE AUTHORS



Aditya Kumar Pal works as a Lead Data Scientist with Rakuten at their Bangalore office. Aditya has a rich experience of more than eight years in Data Science and Business Analytics. Over the past eight years, he has worked with more than 50 stakeholders to solve their problems using data and algorithms across multiple functions such as customer analytics, pricing analytics, assortment analytics, and marketing analytics.

Aditya developed an interest in model explainability after hearing about it at a seminar. Since then, he has worked continuously on a variety of problems to gain expertise in the domain. He learned various algorithms and used to find a unique way to combine his knowledge of the business domain with explainability to create value-creating solutions for stakeholders.

Aditya has received multiple awards for Data Science and problem-solving across all the organizations he has worked at. His passion for going into the extreme depth of a topic motivated him to consolidate his knowledge on the explainability of machine learning algorithms into a textbook so that his learnings can benefit others in the industry. Aditya also participates in conferences and has spoken across multiple analytics schools and forums as a guest speaker. He also took a part-time role as a Data Science coach with one of the leading online education platforms to guide students in the data sciences. A passionate sports player and a part-time artist, Aditya also has immense love for motorcycles and cars and envisions someday combining his love for Data Science with his hobbies. He also cares about social causes such as education for underprivileged kids and environmental protection.

You can reach him by email at aditya.nitrr@gmail.com and on LinkedIn at www.linkedin.com/in/aditya-kumar-pal-1423624a.

About the Technical Reviewers



Bharath Kumar Bolla has over ten years of experience and is currently working as a Senior Data Science Engineer Consultant at Verizon, Bengaluru. He has a PG Diploma in Data Science from Praxis Business School and M.S in Life Sciences from Mississippi State University, U.S.A. He previously worked as a data scientist at the University of Georgia, Emory University, and Eurofins LLC & Happiest Minds. At Happiest Minds, he worked on AI-based digital marketing products and NLP-based solutions in the education domain. Along with his day-to-day responsibilities, Bharath is a mentor and an active researcher.

To date, he has published around ten articles in journals and peer-reviewed conferences. He is particularly interested in unsupervised and semi-supervised learning and efficient deep learning architectures in NLP and Computer Vision.



Pradeeptha Mishra is the Head of AI (Lymbyc) at L&T Infotech (LTI), leading a group of data scientists, computational linguistics experts, machine learning, and deep learning experts in building artificial intelligence-driven features for the product. He was named one of “India’s Top 40 Under 40 Data Scientists” by *Analytics India Magazine* in 2019 and 2020.

Pradeeptha is an inventor and has filed five patents in different global locations. He is an author of four books. His first book was recommended by the HSLS at the University of Pittsburgh, PA. His most recent book is *PyTorch Recipes* (Apress, 2019). Pradeeptha delivered a keynote session at the 2018 Global Data Science Conference. He has delivered more than 500 tech talks on data science, machine learning, deep learning, natural language processing, and artificial intelligence at

ABOUT THE TECHNICAL REVIEWERS

universities, meetups, technical institutions, and community-arranged forums. He is visiting faculty at Reva University, Bangalore, India, and various other universities. He has mentored and trained more than 2,000 Data Scientists and AI engineers in the last nine years.



Abhishek Vijayvargia is a Data and Applied Scientist at Microsoft. Before this, he worked in many startups related to machine learning and IoT. He developed AI algorithms to handle various cybersecurity, IoT, manufacturing, shipping optimization, and transportation problems. He is also an expert and active researcher in explainable AI and ML system design. He is a data science author and research paper reviewer and has presented his ideas at many AI conferences.

Acknowledgments

This book is a motivation-driven endeavor. Over the last few years, we got fascinated by the importance of building models, which can be explained since we had to interact with stakeholders from different businesses as a part of our daily work. We found one common problem when we started adopting data sciences to solve business problems—our clients' struggle to understand the recommendations made by us. We felt an urgent need to find explainable models. We started reading about model interpretability and found that the domain is very new and has a lot of potential. For a few use cases, we got fascinated by the kind of difference it could bring to our analysis.

We would like to thank Takuya Kitagawa, Kazuhito Nomura, and Yusuke Kaji at Rakuten. They introduced us to this domain and helped us experiment across various methods and use cases to understand the true potential of model interpretability.

Finally, we would like to acknowledge the invaluable help and guidance of the Apress publishing team for giving us this opportunity to present our work. Special thanks to all the reviewers for patiently reviewing our work and working with us through multiple iterations to give the best version to the readers.

Introduction

Interpretability and *explainability* have become two of the top trending search words in machine learning. This book explains machine learning interpretability by using different explainability algorithms. The book begins by talking about the theoretical aspects of machine learning interpretability. The first few chapters explain interpretability, the common properties of interpretability methods, the general taxonomy for classifying methods into different sections, and how methods should be assessed in terms of human factors and technical requirements.

In the first few chapters, readers holistically learn about choosing an interpretability method. These chapters are designed to provide information about interpretability in an academic style, with each section explaining the significance in detail with proper examples. We include quotes from actual business leaders and technical experts to showcase how the real-life users perceive interpretability and its related methods, goals, stages, and properties.

In the next few sections of the book, we deep dive into the technical details of the interpretability domain. Starting with the general frameworks of different methods, we then use a data set to show how each method generates output with actual codes and implementations. The various methods are divided into different types based on their explanation frameworks. Common categories are feature importance-based methods, rule-based methods, saliency maps methods, counterfactuals, and concept attribution. The book concludes with how data affects interpretability and the common pitfalls of explainability methods.

On completing the book, you will understand the working of model interpretability and explainability methods whenever you encounter them and select and apply the most suitable interpretation method for a machine learning project. After reading this book, readers will be easily able to convert a black-box model into a white box.

CHAPTER 1

The Evolution of Machine Learning

This chapter talks about the evolution of machine learning. It briefly discusses how machine learning as a domain grew over time with a year of key events and research. The chapter also covers how to learn a machine learning algorithm. We believe the described method can then be further applied to interpretable algorithms. You can use these learnings to learn about the different methods covered in detail later in the book. Finally, we discuss why accuracy metrics alone are insufficient to evaluate a machine learning algorithm and make a case for learning about interpretability.

Defining Machine Learning

We are witnessing a global revolution of technology in this century. There have been phenomenal advancements in the field of computational power and machine learning applications. The twenty-first century has witnessed major advancements in artificial intelligence (AI) research. Machine learning is one of the most successful and widespread applications of technology, which affects a wide range of industries and impacts billions of users every day. Machine learning is a subset of artificial intelligence that uses algorithms and statistical models for computers to perform specific tasks without human interaction. Machine learning utilization opens the door to futuristic technologies that people use in their daily life for automation of tasks like using Alexa for voice commands or gesture control for a TV remote and so forth.

Next, let's go over the evolution of AI and related machine learning techniques.

The Evolution of Machine Learning

This section talks about the evolution of machine learning. Many people believe that machine learning is a very new concept; however, the roots of machine learning date back to the 1950s. Figure 1-1 is a timeline of some important machine learning discoveries.

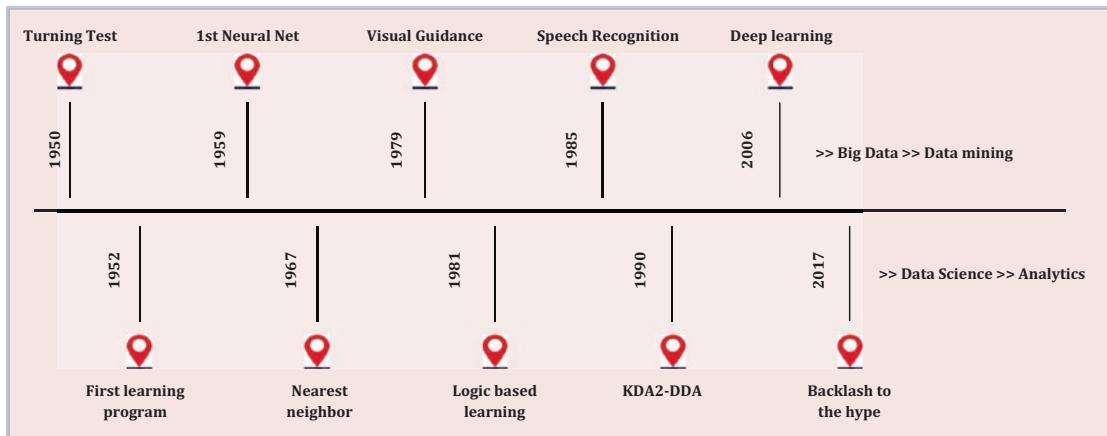


Figure 1-1. Timeline of the evolution of machine learning over time

- **1950:** Alan Turing introduced the Turing test to determine whether a computer can exhibit the same intelligence as a human. This was the very first example of artificial intelligence or machine learning.
- **1956:** John McCarthy coins the term *artificial intelligence* at a conference at Dartmouth College. Later that year, Allen Newell, J. C. Shaw, and Herbert Simon created the Logic Theorist, the first-ever running AI software program.
- **1967:** Frank Rosenblatt built Mark 1 Perceptron, the first computer-based neural network that “learned” through trial and error. One year later, Marvin Minsky and Seymour Papert published a book called *Perceptrons*, which became the landmark work on neural networks and an argument against future neural network research projects.
- **1980s:** AI applications widely adopted Neural networks which used backpropagation algorithms to train themselves.

- **1997:** World chess champion Garry Kasparov was beaten by IBM's Deep Blue in a chess match.
- **2011:** Champions Ken Jennings and Brad Rutter beat by IBM Watson at *Jeopardy!*
- **2015:** Baidu's Minwa supercomputer used a deep neural network called a *convolutional neural network* (CNN) to identify and categorize images with a higher rate of accuracy than the average human. AlexNet was born out of the need to improve the results of the ImageNet challenge. This was one of the first Deep convolutional networks to achieve considerable accuracy on the ImageNet LSVRC-2012.
- **2016:** DeepMind's AlphaGo, powered by a deep neural network, beats the world champion of Go players in a five-game match.

Combining machine learning with cognitive technologies can make it even more effective in automating tasks that humans perform.

The process of learning begins with observations, such as examples, direct experience, or instructions, to look for patterns in data and make better decisions in the future based on the inputs that we provide. The aim is to allow the computers to learn automatically without human assistance and accordingly adjust actions.

Supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning are different categories of machine learning algorithms.

- **Supervised learning** is a subset of machine learning in which machines or algorithms are trained on well-labeled data and then predict the output based on the acquired information. Supervised learning is a process of providing input data as well as correct output data to the model.
- **Unsupervised learning** analyzes unlabeled data sets without the need for human interference or assistance. This is widely used to extract generative features, identify meaningful trends and structures, create groupings, and explore exploratory analysis. The most common unsupervised tasks are clustering, density estimation, feature learning, dimensionality reduction, finding association rules, and anomaly detection.

- **Semi-supervised learning** can be defined as a hybrid of supervised and unsupervised methods, as it operates on both labeled and unlabeled data. Some application areas where semi-supervised learning is used include machine translation, fraud detection, labeling data, and text classification.
- **Reinforcement learning** is a machine learning algorithm that enables software agents and machines to automatically evaluate the optimal behavior in a particular context or environment to improve its efficiency; that is, an *environment-driven approach*. This type of learning is based on reward or penalty, and its goal is to use insights obtained from environmental activists to increase the reward or minimize the risk.

Learning a Machine Learning Algorithm

Learning a machine learning algorithm can be a very difficult task. There are so many websites, books, and papers describing how the algorithm works mathematically and textually. If you do proper research, you might find a pseudocode description of the algorithm.

You can quickly learn when you want to implement a method from research papers; however, algorithms are rarely sufficiently described for easy reproduction. The reasons vary, from the micro-decisions left out of the paper, to whole procedures summarized in text, to symbols used inconsistently displayed.

Hence to learn algorithms properly, you should follow the steps listed here and described in the following sections.

Piece It Together

To understand the algorithm, it is very important to break down the algorithm into several components. Then for each component, secondary research should be done. Sometimes an algorithm contains components that are referenced from other papers or algorithms. Understanding each component in detail helps tie the learning back to the overall working of the algorithm. Upcoming chapters in the book talk about various algorithms and cover subcomponents that are directly referred from any other algorithms. This ensures that no blind spots are left in the algorithm understanding, and all processes and steps are covered.

Focus on Specific Algorithm Descriptions

Focusing only on the algorithm definitions can sometimes prove very harmful. You must source algorithm descriptions from various sources and look for only verified sources.

For example, when we study random forests, we sometimes focus only on understanding bagging as a concept and how the description of random forest talks about various trees working together to make a prediction. We also focus on other algorithm descriptions that tell how random forest differs from other algorithms and how feature importance is computed. Inside the algorithm, there is a lot of information that can be useful for various use cases. For example, usage statistics for an algorithm are embedded in papers. Examples include the standard experimental data sets used for testing the algorithm and the general classes of problems to which the algorithm is suited.

Detailed algorithm descriptions ensure that we dive into all details of the method and are not biased by a few sources stating half-cooked information.

Design an Algorithm Description Template

It becomes effortless to learn an algorithm using a description template. Start with a blank document and list out the overview of sections you might want to learn. Refer to the following sample to guide you on what to include in your template or list the questions you want to answer about the algorithm.

- **Introduce** the algorithm. Write a brief introduction about the overview of the algorithm.
- Why is this algorithm needed? Do not start learning about an algorithm without any use case. First, decide a use case to solve and then relate various algorithms and select one. Hence in the template, it is very important to mention which use case the selected algorithm solves.
- What is the **math** behind the algorithm? Start with the mathematical model and formulas required in the algorithm. This sets up a strong base for learning the algorithm.

- **How do you** implement the mathematics and generate **pseudocode**?
Once you are confident about the mathematics involved in the model, you should generate simple pseudocode for the various steps. Most of the algorithms work by systematically coding the different mathematical calculations.
- **Are there** similar algorithms? **Why use** this specific algorithm?
Most of the time, a good use case justifies your need for choosing a particular algorithm.
- **What can be solved** using this algorithm? Think of other use cases that can be solved using the same algorithm. This expands the algorithm's horizon.
- **What is the process strategy?** Mention the strategy that you adopt to process this algorithm. Think about any intermediate outputs. Fix on how the final output looks like.
- **Which abbreviations are used?** Mention any abbreviation which might require you to do additional research. Most algorithms mention the modules borrowed from other algorithms and display them in the form of abbreviations.
- **What are the secondary resources?** Make a list of all secondary resources which have any info about the algorithm.

Start Small and Build It Up

The advantage of this approach is that there is no need to be an expert in the algorithm or research. Find a resource on the algorithm and capture notes about it in the template. You can then pick individual straightforward modules and run them for sample data. Once you are comfortable with simpler modules, expand the scope to incorporate more challenging modules which require more coding. Once that is done, incorporate more data and scale up horizontally or vertically. With this approach, you learn in a step-wise manner and do not miss the finer details.

Investigating Machine Learning Algorithm Behavior

To understand the behavior of machine learning algorithms, we often tend to read about the various descriptions and literature available about the algorithm. However, a very good method to learn or investigate the behavior of machine learning models is to run some simple tests and experiments using various data sets and observe the pattern of metrics or predictions. Generally, all machine learning models are complex, and hence it is very difficult to understand their behavior via theoretical knowledge alone. It is best to see an algorithm in action. Experimenting helps you understand the cause and effects of algorithm parameters and how to tune a model.

Let's look at a simple five-step process to investigate a machine learning algorithm.

Step 1. Select an Algorithm

Choose an algorithm about which you have concerns. The selection of the algorithm can be based on an ongoing project or simply based on popularity. Selection is an important thing because there are multiple algorithms available with a variety of approaches for each problem. Some approaches are basic, while some are much more complex. Hence it is important to choose an approach that explains the basic level of the problem without directly jumping into innovative complex techniques. Take an off-the-shelf implementation of the algorithm. Implement the algorithm and introduce additional variables into the experiment, such as bugs and other micro-decisions that must be made for each algorithm implementation.

Step 2. Identify a Question

Decide a research question about which you would want to get some insights about the algorithm. The more specific the question, the more useful it is to get an answer. The following are some example questions.

- What is the effect of the depth of tree in a random forest model?
- What is the effect of selecting kernels in support-vector machines (SVM) on binary classification problems?

- What are the effects of different attribute scaling on logistic regression on classification problems?
- What effect does adding attributes to the training data set have on classification accuracy in a random forest model?

Design the question to which you need an answer about your algorithm. Consider listing a few variations of the question and focus on the one that is the most specific.

Step 3. Design the Experiment

Pick the elements out of the questions that make up your experiment about the model algorithm. For example, let's look at the following question: What are the effects of attribute scaling on logistic regression of classification problems?

You can select the following elements.

- **Attribute scaling methods:** Methods like normalization, standardization, raising an attribute to a power, and taking the logarithm
- **Logistic regression:** Different implementations of logistic regression
- **Binary classification problems:** Multiple problems are required, some with attributes all the same scale and others that have attributes with a variety of scales
- Performance: A model performance score, such as classification accuracy.

Take enough time to carefully select the elements to best answer your question.

Step 4. Execute the Experiment and Report Results

Complete your experiment. If you are looking for differences in results between experimental runs, you may want to use a statistical tool to indicate whether the differences are significant or not. Summarize the results of your experiment. You may want to use tables and graphs.

What results did we get from the research questions?

Put on your skeptical hat. What limitations can you place on the results? Knowing the limitations is just as important as knowing the outcomes of an experiment.

Step 5. Repeat

Repeat the process. Continue to investigate your selected algorithm. You may even want to repeat the same experiment with different parameters or different test data sets. Don't stop with one experiment. Start building a knowledge base and an intuition for the algorithm. With some simple tools, some good questions, you can very quickly start coming up with a world-class understanding of the behavior of an algorithm.

What Does Machine Learning Model Accuracy Mean?

In machine learning algorithms, accuracy measures how successful a model is at capturing patterns in the training data and then using this learning to predict unseen data. This shows how well the model generalizes unseen data. It is believed that the better the predictions of a model, the higher the business value they can produce based on actions that we take from model insights. Thus, accuracy is a measure of the success of the model based on various metrics. For example, for a classification model, a measure of accuracy may be how well the model can predict all positive and negative classes in the data.

There is a myriad of metrics that are available to evaluate a model's accuracy. These metrics are mostly model agnostic and work well with different models, while some are specific to a particular model. Since accuracy does not always give a good picture of model performance, for example, in imbalanced data sets, you should use different metrics to evaluate the model. However, for the sake of discussion, we mentioned all these metrics under the general term of model accuracy whenever we must explain our model to a non-technical person. The following are some useful metrics on the accuracy of models that check how good your model performance is.

- Confusion matrix
- F1 score
- Gain and lift charts
- Kolmogorov-Smirnov chart
- AUC-ROC curve

- Log loss
- Gini coefficient
- Concordant/discordant ratio
- Root-mean-square-error

Diving into each metric is beyond the scope of this book. We leave it to you to research these metrics and their potential usages.

Why Model Accuracy Is Not Enough

Companies use machine learning models to make business decisions, and more accurate model outcomes result in better decisions. The costs of errors are often huge, but optimizing model accuracy mitigates those costs. There is a point of diminishing returns when the value of developing a more accurate model does not result in a corresponding profit increase. However, often it is beneficial across the board. A false positive cancer diagnosis, for example, costs both the hospital and the patient. The benefits of improving model accuracy include saving time and money and avoiding stress.

The idea of building machine learning models works on a feedback principle. You build a model, get feedback from metrics, make improvements, and continue until you achieve the required accuracy. Evaluation metrics explain the performance of a model and discriminate model results.

Many analysts and aspiring data scientists are not concerned to see how robust their model is. Once they're finished building a model, they hurriedly map predicted values on unseen data. This is a wrong approach.

Simply building a predictive model isn't your motive. It's about creating and selecting a model which provides high accuracy in sample data. Hence, it's crucial to see the accuracy of your model before computing predicted values.

In our industry, we consider different sorts of metrics to gauge our models. The choice of metric ultimately depends on the model and, therefore, the implementation plan of the model.

Accuracy is good, but we need additional info for the business consumption of models.

Models can achieve high accuracy by memorizing the unimportant features or patterns in your data set. If there is a bias in the input data set, this can also affect the model. In addition, the info within the training environment might not be an honest representation of the info within the production environment during which the model is deployed. Even if it's sufficiently representative initially, if we consider that the information within the production environment isn't stationary, it can quickly become outdated.

Thus, we cannot rely only on the prediction accuracy achieved for a selected data set. We need to know more. We need to demystify the machine learning models and improve transparency and interpretability to make them more trustworthy and reliable.

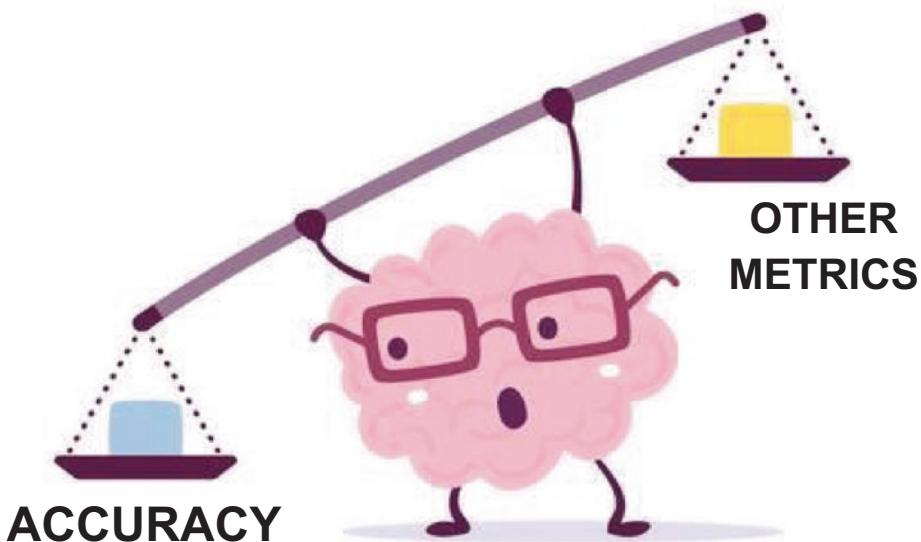


Figure 1-2. A pictorial description of how accuracy gains more importance than other model evaluation metrics

As the machine learning models became more and more advanced, the accuracy that they could achieve on different data sets increased; however, with this increase in accuracy, the models became more and more complex, and the explainability of the model results went for a toss.

However, when it comes to actual implementation, accuracy is not enough to justify the decisions made by the model. So, in real-world scenarios, even if the underlying model is very complex with high accuracy metrics, the business teams sometimes find it very difficult to adopt the models because accuracy does not explain why the model took a particular decision.

Let's look at a few examples to understand how some of the highly accurate models failed in real-world scenarios and hence were discarded

- In 2019, the Polish government added an amendment to a banking law that gives a customer the right to demand an explanation in an unfavorable credit decision. One of the consequences of implementing General Data Protection Regulation in the European Union is that the effect of GDPR legislation is that a bank must explain why the loan was not granted in a scenario where the decision process was automatic.
- In October 2018, world headlines reported about an AI recruiting tool that favored men. The Fortune 500 company's model was trained on biased data that were skewed toward male candidates. It had built rules that penalized specific resumes that included the word *women*.

In these examples, both models are very complex tools called *black-box classifiers*, which offer very high accuracy; however, these models don't offer straightforward human-interpretable explanations.

Predictive algorithms, also interpretable, tend to be less accurate than more advanced methods that are not as easily explained. It boils down to how easily can the model be explained to a human being? Model accuracy, however, is any quantitative metric that explains how well the model predicts the outcome of interest. For an example of all known observations, how many did the model predict correctly.

For being such a quantitative field, the importance of interpretability in predictions remains a divisive topic for data scientists. As more companies explore advanced analytics to drive growth, this tension becomes relevant to analytic success. As data science practitioners, how can we instill confidence in our methods if our clients are unfamiliar with them? Additionally, how do we justify the value of analytics if we restrict our models to make them uninterpretable?

It is easy to build a case in favor of weighting model accuracy over interpretability. Accuracy can be easily scored, allowing alternate methods to be compared readily with one another. If a predictive model performs better in the selected score, that should weigh heavily in its favor. When the cost of an error is high, the incentive for higher performance is much greater. *Interpretable* models like decision trees or linear/logistic regression can also rapidly become non-interpretable. For example, after the first few levels of a decision tree, it becomes difficult for humans to understand the reasons for selecting one variable over another.

If a human can understand why a model is wrong, that is an excellent example of interpretability. We are explanation hungry and can easily make spurious arguments based on the results presented from the data or even with no data.

The relative importance of accuracy picks up even more as the cost of having an error increases. For example, IBM's Watson technology supports medical diagnoses. Recently, Watson successfully diagnosed a rare form of leukemia that oncologists were unable to. This is a fantastic example of how accurate machine learning algorithms can support existing workflows.

But imagine that Watson was wrong, and the patient would have been enrolled in an expensive and time-consuming course of treatment that would have provided little to no benefit and would almost certainly have been harmful to the patient. Like all other machine learning algorithms, Watson can only learn about the data on which it was trained; it cannot learn about alternative hypotheses not associated with the data on which it was trained. Accuracy is critical in the healthcare environment, but accuracy is only as good as the data used to train the model. When Watson makes a diagnosis, the logic behind the diagnosis must also be evident. The cost of an error is significant to be left to a black-box model. In the world of financial services, a mortgage loan or insurance claim cannot be denied based on race. If race has been included in a model, it violates regulations that prohibit discriminatory practices.

This swings the needle back from accuracy toward interpretability.

Summary

This chapter discussed the evolution of machine learning and the different categories of machine learning models. It also described the best practices for learning a machine learning model and exploring its behavior. You learned what model accuracy means and our viewpoint on why accuracy may not be enough to evaluate models. This chapter made a case for involving explainability or interpretability as support for accuracy when evaluating models. Hence, the next chapter dives into interpretability and how it relates to model evaluation.

CHAPTER 2

Introduction to Model Interpretability

This chapter discusses the importance of interpretability. We start with an example to explain the importance of explanations in a real-life scenario. Then, we try to correlate this example with our models and how explanations are an integral part of your model learning journey. We explain the different types of motivations behind adopting interpretability and end the chapter with details about current researches going on in the interpretability domain

Humans Are Explanation Hungry

Humans are explanation hungry animals. Since the start of civilization, we have always tried to find the *why* and *how* of things. Using the knowledge of why and how that has been collected over centuries, we humans have built rules or best practices for specific tasks. With the evolution of technology, we have expanded the rules and have been able to create computer software to process these rules. The following story highlights why simple explanations or the *why* and *how* behind processes are important.

Two doctors were in the same room attending their patients. Most of the patients were suffering from diabetes. Arjun sat in front of his doctor with curious eyes. Occasionally he looked across the room. A person of similar age, weight, and height sat across the room with the other doctor. Arjun's doctor told him that his diabetes was not under control, and he prescribed medicines for a few more weeks. Arjun was disheartened. This illness was like a black box to him. He tried to discuss why the medicines were not working, but his doctor was too busy to handle additional questions. He asked the attendant to send in the next patient and politely signaled Arjun to leave and come back after two weeks.

Arjun slowly walked out of the room and noticed the same person sitting with the other doctor. Hoping that he might be experiencing the same problem, Arjun approached them and introduced himself. The fellow was very joyful. His name was Vikas, and he was the same age as Arjun. He also had diabetes, but currently, it was under control, and he had come to his doctor to express his gratitude for the treatment. Arjun was excited after hearing his treatment story. He puzzlingly asked Vikas how he got it under control. What medicines did the doctor prescribe? Vikas showed his prescription to Arjun, and Arjun was completely surprised. Every medicine was the same as his. Arjun was curious about how the same medicines that worked for Vikas did not change his own symptoms. They agreed to get some coffee nearby and started talking.

Vikas then explained how he was also struggling at the start of the treatment and how he felt that diabetes was like a black box until he read a book that explained diabetes to ordinary people in simple terms. After that, his life changed. Vikas told Arjun how diabetes is not just a disease that can be treated with medicines, but it also requires life style changes, good food habits, and a healthy workout regime. The book had simple explanations for everything. How a particular food affects sugar levels in the body, how the sleep cycle affects sugar levels, the most important factors that increase sugar levels, and the important exercises that reduce stress and manage sugar. The answers to these questions helped Vikas convert his black-box disease into an explainable one. He knew the reason behind his fluctuating levels, and he started taking relevant actions. He suddenly knew which foods elevate his sugar levels and what actions to take post-eating to bring it down. With his newfound knowledge and medicine, Vikas was very soon able to control his diabetes. He now enjoys his favorite foods and feels healthy more than before.

Arjun was awestruck after listening to Vikas. He suddenly saw a ray of hope. Simple explanations might change his life. Earlier, he only got to know about whether his sugar levels were up or down. With these explanations, he can now tie back those levels to the food he ate, the amount of sleep he took, and whether he was active during the day or not. He suddenly knew all the reasons, and medicines were merely the icing on the cake for him. Arjun thanked Vikas, and they both said goodbye to each other. A few weeks down the line, Arjun managed his diabetes, and he was off medicines.

The knowledge that humans now have about everything is far greater than what we can comprehend at any given time. Hence, sometimes we blindly follow the rules without going deep into understanding what created those rules. Arjun was curious enough to ask about his diabetes problem, but most people do not because they *trust* the

medicines and the doctor. The rules that have been made were made over centuries of research and evolved to ensure that they are not biased toward a particular community, color, age, or body type. The rules are *unbiased* and are expected to offer the same solution for all humans. Since the rules have evolved, they have become the best practice for doing certain tasks; hence, we can hold them *accountable*. You want to trust the medicine prescribed by doctors for your illness and whether it works or not, we hold the doctor accountable.

Explanations in Machine Learning

Let's look at an example of a bank loan processing application. The rules for processing bank loans have been made after years of research. In the past, the bank manager decided who should get a loan based on an applicant's income and credit history. For an application denied, the bank manager had a straightforward answer to why the loan didn't get through. These days banking companies have machine learning models trained on millions of loan applications with hundreds of variables. These models can help the bank manager determine with high accuracy whether a loan should be granted or not. But since it is now an algorithmic decision with a very complex process, a few questions arise: Why was the loan denied? Or why was the loan granted? Is the decision made by the algorithm correct?

Throughout this book, we try to answer such questions and explain methods that help companies or individuals answer such questions.

Chapter 1 explained machine learning and how its importance is rising. Now, let's apply this concept of understanding *why* and *how* to machine learning models by looking at a simple loan approval or rejection.

Figure 2-1 shows simple banking loan model data.

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CooapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0

Figure 2-1. Sample data for a bank loan classification model

The data has variables like loan status, loan amount term, income, education, credit history, and age. While building models, we would fit a logistic regression model to predict whether a loan application is approved or rejected. We have some independent variables and one target variable (i.e., `Loan_Status` in the data set). In logistic regression, the target y is binary (**Approved** $p = 1$ / **Rejected** $p = 0$), and the probability of granting/rejecting the loan (p) is determined based on the cutoff value. The goal is to estimate the coefficients α_i .

$$\text{Logit}(p) = \log(p/[p-1]) = \alpha_0 + \alpha_1 \cdot \text{age} + \alpha_2 \cdot \text{income} + \alpha_3 \cdot \text{age} + \alpha_4 \cdot \text{credit history}$$

To find coefficients α_i , we train the classification model with a labeled data history. The decision approved/rejected is already known, using cross-entropy as a loss function to compare the predictions \hat{y} , vs. labels, y (see Figure 2-2).

$$L(\hat{y}, y) = -(y \cdot \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$

Figure 2-2. Logistic regression equation representation

The function shown in Figure 2-2 represents binary cross-entropy loss. For binary classification, we have binary cross-entropy. Cross-entropy loss is also called *logarithmic loss*, *log loss*, or *logistic loss*. Each predicted class probability is compared to the actual class desired output 0 or 1. A score/loss is calculated that penalizes the probability based on how far it is from the actual expected value. The penalty is logarithmic, yielding a large score for large differences close to 1 and a small score for small differences tending to 0. This function is negative because when we train, we need to maximize the probability by minimizing the loss function—decreasing the loss increase the maximum likelihood assuming that samples are drawn from an identically independent distribution. The values of α_i are those that minimize $\alpha_0, \dots, \alpha_4$ using its first derivative and an optimization algorithm like gradient descent.

We should be very cautious while interpreting the estimated coefficients. The slope coefficient is now interpreted as the rate of change in the “log odds” as X changes compared to the rate of change in Y (the dependent variables) as X changes in the LP model or OLS regression. The logit coefficient is the “odds ratio,” which is the effect of the independent variable on the probability of the event divided by the probability of the non-event. For example, if $\exp\alpha_3 = 2$, then a one-unit change in X_3 would make the event twice as likely (67%/33%) to occur. Odds ratios equal to 1 mean a 50% chance that the

event occurs with a small change in the independent variable. Negative coefficients lead to odds ratios less than one: if $\exp a_2 = .67$, then a one-unit change in X_2 leads to the event being less likely (40%/60%) to occur. Negative coefficients tend to be more complex to interpret than odds ratios greater than positive coefficients. Coefficients comprise the very first form of interpretability for generalized linear models. The example shows how to determine the effects of independent variables on the dependent loan status variable using coefficients. The coefficients tell which variables have the most impact and the direction of impact (positive/negative).

Linear and logistic models are self-explanatory models. By understanding the math behind the models, you can deduce why the model is taking a certain decision based on the input data.

However, with time, the amount of data has been expanding rapidly. With the increased amount of data, newer and more advanced algorithms are being researched daily to perform specific tasks. Over time, the popularity of decision trees has increased significantly.

A decision tree is a supervised machine learning algorithm with the capability of performing both classification and regression. A decision tree is a series of sequential decisions made to reach a specified output. Figure 2-3 shows how a decision tree works.

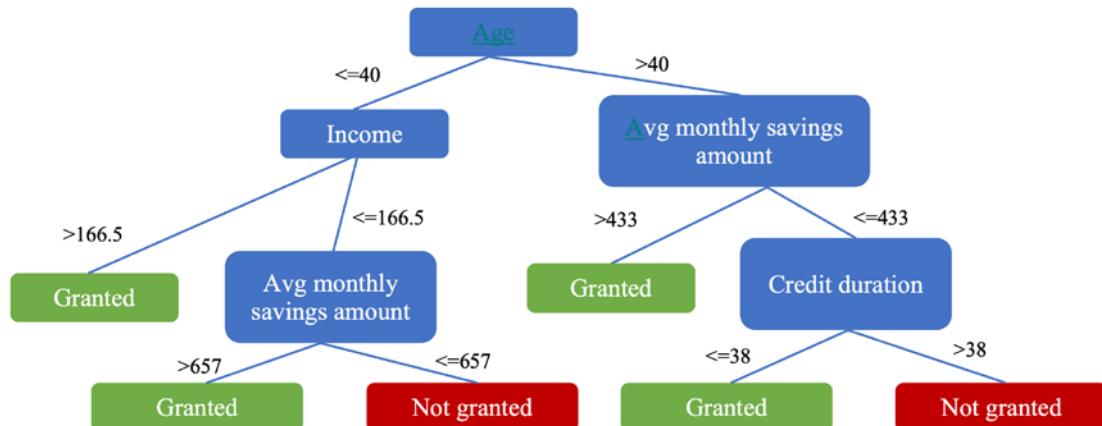


Figure 2-3. Decision tree representation

The tree classifies customers into two groups. One group is customers with good credit history, and the other group contains customers with bad credit history. Further, it checks the income of the customer and again classifies the customers into two groups. Finally, it checks the loan amount requested by the customer. Based on the outcomes

from checking these three features, the tree decides whether the customer's loan should be approved or not. The decision tree algorithm is relatively easy to understand and interpret as it divides data into two parts. However, often, a single tree is not sufficient for producing effective results. This is where the random forest algorithm can be used.

A *random forest* is a tree-based machine learning algorithm that leverages multiple decision trees for generating decisions. As the name suggests, it's a "forest" of trees (see Figure 2-4)!

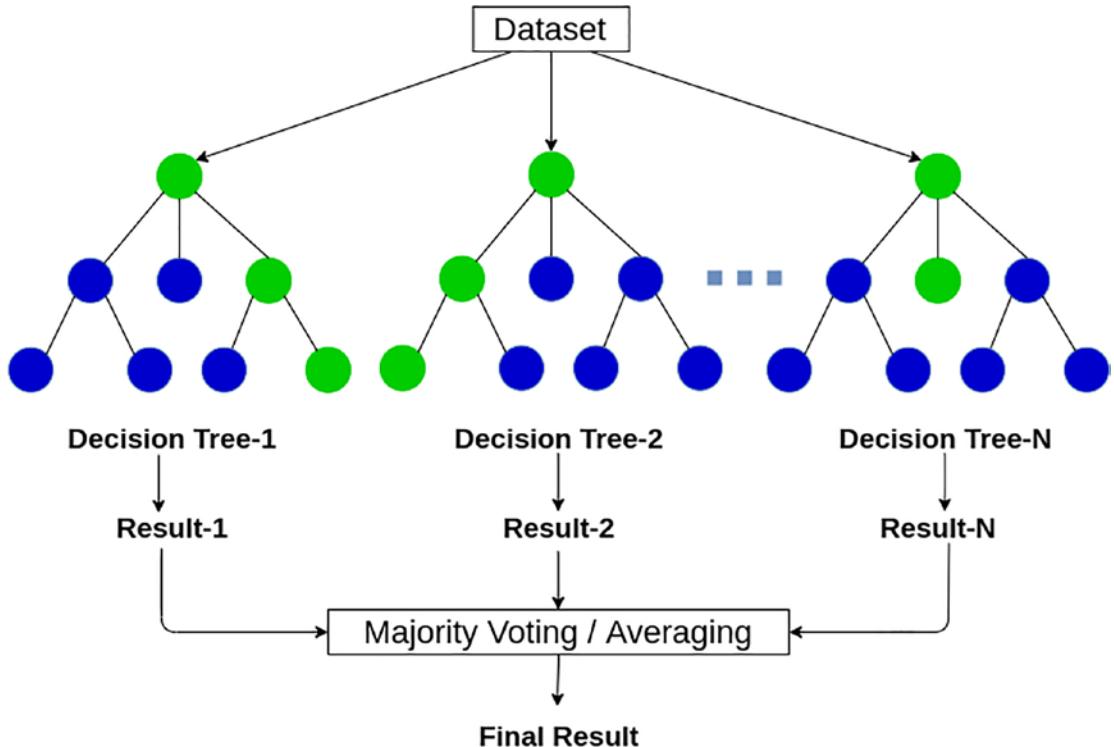


Figure 2-4. Random forest representation

But why is it called a *random forest*? That's because it's a forest of randomly created decision trees. Each node in the decision tree works on a random subset of features to calculate the output. The random forest then combines the output of individual decision trees to get the ultimate output. Figure 2-4 shows a simple representation of a random forest.

Recently, neural networks have overtaken decision trees as preferred for modeling huge data sets due to their ability to handle various problems such as image classification and text classifications apart from performing well on tabular data. Neural

networks also are ideally suited to assist people in solving complex problems in real-life situations. They can do various tasks like a model, learn nonlinear relationships between input and output; generalize and inferences; reveal hidden relationships, patterns, and predictions. Figure 2-5 is a simple representation of a neural network.

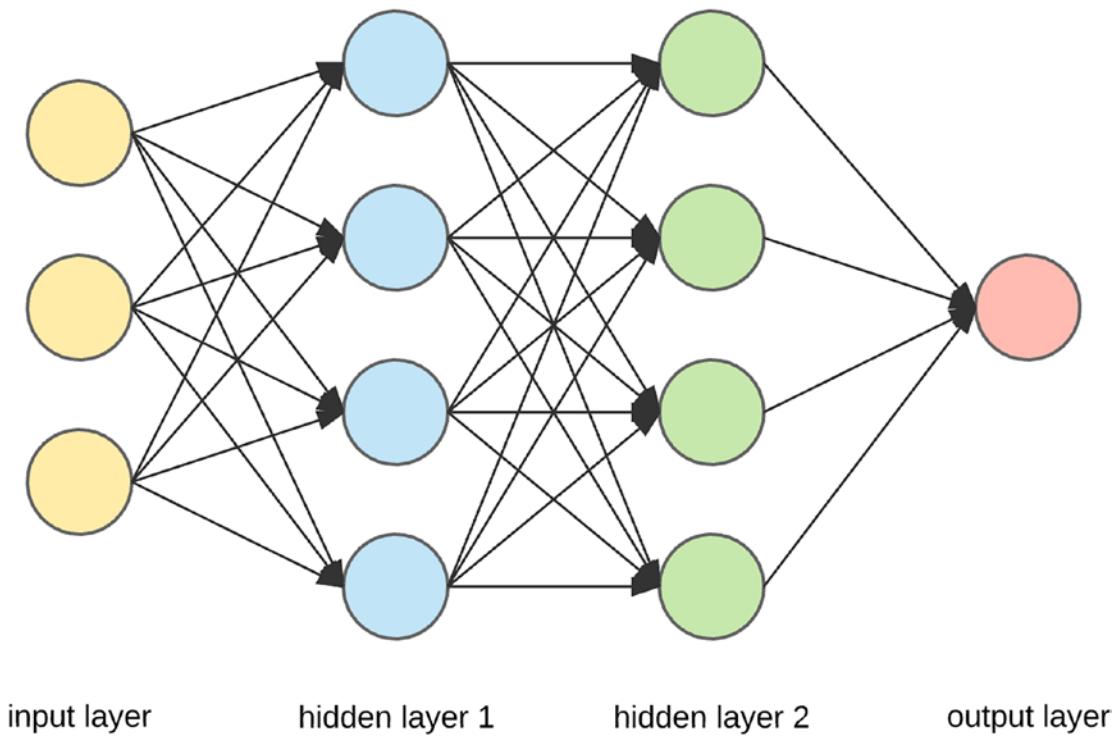


Figure 2-5. Neural network representation

Unlike logistic regression, which is based on a simple equation, other advanced algorithms like decision trees and neural networks are complex and have multiple steps. It becomes difficult for even a data scientist to understand these models' inner workings, let alone their business counterparts. These new techniques developed post linear models are called *black-box models*.

What Are Black-Box Models?

Black-box model is a term to describe models that have complex workings to compute an output typically spread over multiple steps. In a black-box model, only the input and the output are known to the users, while all interim steps and calculations are difficult

to comprehend (see Figure 2-6). Over time the black-box models have gained popularity as the preferred modeling technique due to their ability to generate high accuracy over a variety of data sources.

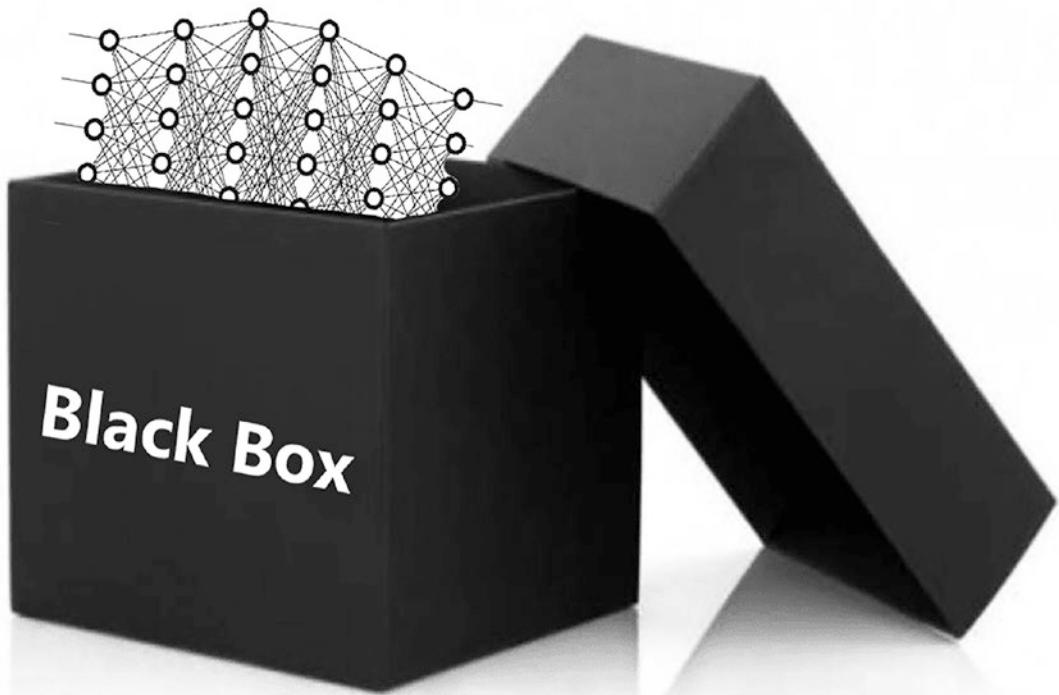


Figure 2-6. Black-box models representation

But adopting these advanced techniques poses a few questions.

- Can we interpret a deep neural network?
- How about a random forest with 500 trees?
- Building a complex and dense machine learning model has the potential of reaching our desired accuracy, but does it make sense?
- Can you open the black-box model and explain how it arrived at the result?

The spark of black-box models was further ignited in recent years on competitive coding platforms such as Kaggle, where black-box models started topping the chart

across multiple problem statements. By 2015, *gradient boosting* and *neural networks* were the most popular terms among data scientists, and very soon, these black-box models entered the world of actual implementation across businesses. In an ideal world, every model would be explainable and transparent, useful in the following.

- Critical decisions (e.g., healthcare)
- Seldom made or non-routine decisions (e.g., M&A work)
- Stakeholder justification-required decisions (e.g., strategic business choices)
- Situations where interactions matter more than outcomes (e.g., root cause analysis)

In the real world, however, there's a time and place for both sorts of models. Not all decisions are equivalent, and developing interpretable models is extremely challenging (and in some cases impossible; for instance, modeling a posh scenario or a high-dimensional space, as in image classification). Even in easier problems, black-box models typically outperform white-box counterparts due to black-box models' ability to capture high non-linearity and interactions between features. Despite the advantage of having high performance in terms of accuracy, there is a very prevalent downside to black-box models: the lack of ability to provide explanations behind the predictions made to internal teams or audit firms and regulators.

Figure 2-7 shows how different methods are placed on a two-way axis between interpretability and accuracy. We can see in the image that more complex methods like deep neural networks and support vector machines are high in accuracy but fall on the lower value of the interpretability axis.

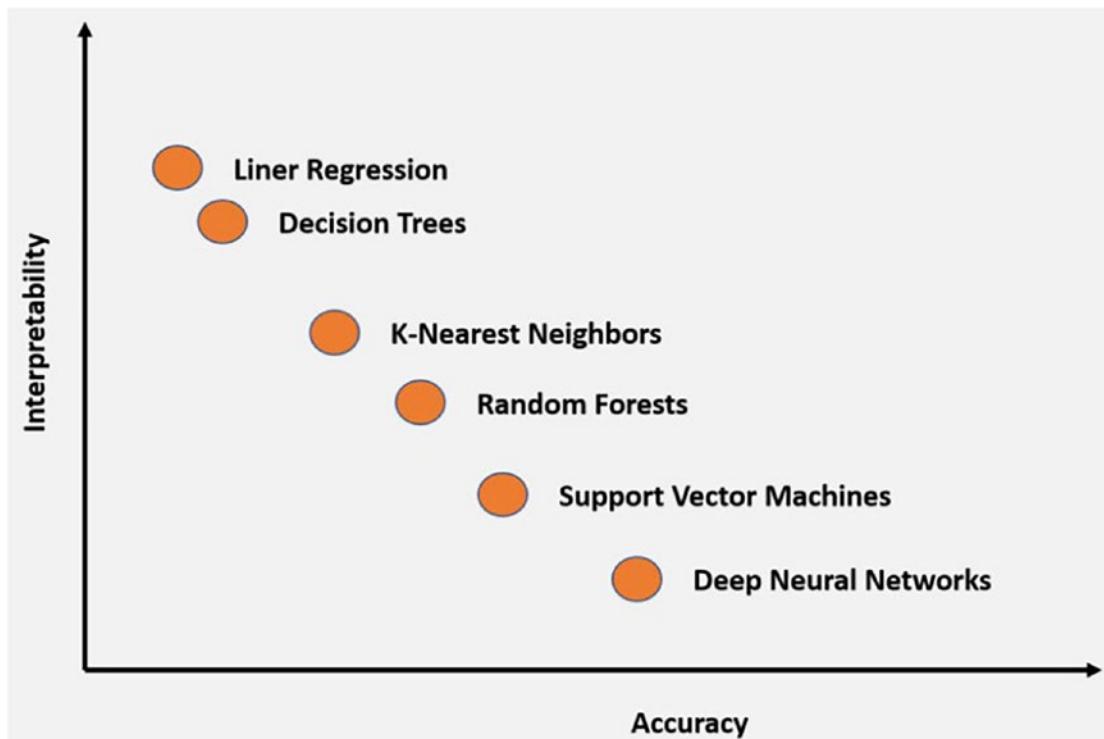


Figure 2-7. A 2D chart of accuracy and interpretability with different methods as data points. This chart shows that as the models became more and more accurate, their interpretability or understanding of such models decreased

The second downside to black-box models is that there could be a lot of unseen problems impacting the output, such as overfitting, correlations, or garbage in, garbage out. These problems are impossible to catch due to the lack of understanding of the black-box model's operations.

To prevent business users and model builders from the curse of a black-box model's interpretability plays a major role.

What Is Interpretability?

Interpretability is that the degree to which we can understand the explanation for a choice. Another one is: Interpretability is the degree to which a person can consistently predict the model's result. Higher the interpretability of a machine learning model, the

better it's for somebody to grasp why certain decisions or predictions are made. A model is more interpretable than another model if its decisions are easier to grasp.

Most machine learning systems require the power to explain to stakeholders why certain predictions are made. When choosing an appropriate machine learning model, we frequently think about the accuracy vs. interpretability trade-off.

The accuracy vs. interpretability trade-off is based on an important assumption: explainability is an inherent property of the model. We believe, however, that with the right techniques, any machine learning model can be made more interpretable, albeit at complexity and cost, which is higher for some models than others.

When a model predicts or finds insights, it takes certain decisions and choices. Model interpretation tries to understand and explain these decisions taken by the model (i.e., the what, why, and how). The key to model interpretation is transparency, the ability to question, and the ease of understanding model decisions by humans. The three most important aspects of model interpretation are explained as follows.

- **What caused the model to make certain predictions?** We should have the ability to query our model and find out feature interactions to get an idea of which features might be important in the decision-making rules of the model. This ensures the *fairness* of the model.
- **Why did the model take a particular decision?** We should also validate and justify why certain key features were responsible for driving decisions made by a model during predictions. This ensures the *accountability* and reliability of the model.
- **Can we trust model predictions?** We should evaluate and validate any data point and how a model takes decisions on it. This should be demonstrable and easy to understand for key stakeholders that the model works as expected. This ensures the *transparency* of the model.

To take actions based on a prediction or when deploying a new model, you should trust the model's prediction, and gaining trust in the model is very important.

A vital concern remains whether humans are directly using machine learning models or deploying models within other products. If the users do not trust a model or a prediction, they will not use it.

In the bank loan model, interpretability helped answer questions like the most important feature that describes the approved loans differently from non-approved loans.

For example, let's say income is the most important variable. The output of interpretability should be something like, "the higher the income, the higher the probability that the loan will be approved." Similarly, if the second-most important variable is credit history, then interpretability provides output in the form of "a credit score between 750 and 800 has a higher probability of getting an approved loan."

In the subsequent chapters, we explain how only knowing that credit history is important versus knowing which values of credit history are important can be achieved through different interpretability methods. Apart from this, you can also get explanations for a single application in the form of income, credit history, and other variables, such as age.

Introducing such explanations on top of models help the bank manager trust the model and ease the process of loan grant. These explanations also help the strategy and marketing team of the bank to design a better loan structure and serve customers better. Having this layer of explanations on top of important models is so much better than only having an accuracy metric for the model run.

Such is the power of interpretability in real-life examples.

The Motivation Behind Interpretability

The prime motivation behind model interpretability is building trust in models. However, the motivation can also be extended to various other requirements. The following section looks at the various motivations to use interpretability methods and techniques.

To Make Better Decisions

We should always ensure that a human is present after the model building process to assess the impact of incorrect predictions. In scenarios where automation is performed using machine learning systems, it is easy to forget the impact of wrong predictions. The team of data scientists and business analysts should understand the consequences of incorrect predictions, especially when projects have a significant impact on human life (e.g., justice, health, transportation, etc.). Similar care should be taken in scenarios

where revenue is getting impacted for a business. Interpretability provides critical information we would need to assess the predictions of the models and understand why a particular decision was taken by the model and how different features of the models impact its prediction

To Eliminate Bias

We can use interpretability to drive motivation to detect, document, and monitor bias in our development and production systems. When building systems that must take decisions, we can always face the computational or societal bias inherent in the data, which might be impossible to avoid. Still, it is always possible to document or mitigate the bias. However, we should not try only to embed ethics directly into the algorithms themselves. Instead, technologists should focus on building processes and methods to identify and document the inherent bias in the data, features, inference results, and subsequently the implications of this bias. Given that the implications of the bias identified are specific to the domain and technology, we as data scientists should be able to create, identify and explain the bias in the data and features, so the right processes can be put in place to mitigate potential risks. Interpretability plays a role in setting up these processes.

To Justify Processes

You can use interpretability to develop tools and processes that improve transparency and the explainability of machine learning models. With deep learning gaining significant attraction in recent years, we as data scientists often throw large amounts of data into complex ML pipelines hoping something work without understanding how the pipelines work internally. However, we should invest reasonable efforts to continuously improve tools and processes to explain results based on features and models chosen. It is possible to use different tools and approaches to make ML systems more explainable, such as by adding domain knowledge through features instead of only allowing deep/ complex models to infer them. Even though in certain situations accuracy may decrease, the transparency and explainability gains may be significant.

To Reproduce Operations

We can use interpretability to develop the infrastructure required to enable a reasonable level of reproducibility across the operations of ML systems. Production machine learning systems often don't have the capabilities to diagnose or respond effectively when something terrible happens with a model, let alone reproduce the same results. In production systems, it is important to perform standard procedures, such as reverting a model to a previous version or reproducing an input to debug a specific functionality, which introduces complexity in infrastructure. Tools and best practices should be adopted to provide a reasonable level of reproducibility of operations.

Displacement Strategy

Interpretability can identify and document relevant information so that business change processes can be developed to mitigate the impact on jobs being automated. When rolling out systems that automate medium to large-scale processes, there is an impact on an organization or industry level, affecting multiple individuals. As technologists, we should look beyond the technology itself and have systems to support the necessary stakeholders. This helps the stakeholders develop a change-management strategy when rolling out the technology. Although data scientists or analysts may not always lead the transformation, it is still important to make sure the processes are in place when relevant, irrespective of the work being automated.

To Determine Practical Accuracy

Interpretability develops processes to ensure that my accuracy and cost metric functions align with the domain-specific applications. When building systems that learn from data, it is important to understand the underlying means to assess accuracy. Often it is not enough to only use plain accuracy or default/basic cost metrics because what may be "correct" for a computer may be "wrong" for a human (and vice versa). Ensuring the right challenge is being addressed in the right way can be achieved by breaking down the implications off-1 score metrics from a domain-specific perspective and exploring alternative cost functions based on domain knowledge.

To Maintain Privacy

Interpretability can build and communicate processes that protect and handle data with stakeholders who may interact directly and/or indirectly with the system.

When developing large-scale systems that learn from data, many stakeholders may often be affected directly and indirectly. Building trust within relevant stakeholders is done not only by informing what data is being held but also by the processes around the data and understanding why protecting the data is important. Technologists should enforce privacy by design across systems and continuous processes to build trust with users and relevant stakeholders such as procurement frameworks, operational users, and beyond. The main goal of having an interpretability privacy framework is to automatically generate explanations of inference tasks on personal data so that end users can better understand the privacy risks and implications of these tasks. The explanations should be helpful to end users so that they understand how the underlying machine learning model works and behaves depending on the inputs (i.e., the data provided by the end user).

To Understand Security Risks

Interpretability can help you develop and improve rational processes and infrastructure to ensure data and model security are being considered during the development of machine learning systems. Autonomous decision-making systems may result in new potential security breaches. It is also important to be aware that many security breaches occur due to human error instead of actual hacks (i.e., someone sending the data set attached in an email by accident or losing their laptop/phone). Technologists should commit to preparing for both types of security risks through explicit efforts, such as educating relevant personnel, establishing data processes, and assessing ML backdoor's implications.

The Research Behind Interpretability

Several industries are witnessing an increasing trend of leveraging ML for high stake prediction applications, which deeply impacts human lives.

When automated algorithms make high-stake decisions, the problem of incorrect predictions becomes even more severe. To address this issue, *explainable machine*

CHAPTER 2 INTRODUCTION TO MODEL INTERPRETABILITY

learning emerged as a field of study focusing on machine learning interpretability and shifting toward a more transparent AI. The main goal of this was to create a suite of interpretable models and methods that produce human-friendly explanations and maintain high predictive performance levels.

One of the entities in this field is the Defense Advanced Research Projects Agency (DARPA), funded by the US Department of Defense. It created the interpretability and explainability program that funds academic and military research at 11 US research laboratories. The program information states that the program aims to produce more explainable models while maintaining high predictive performance levels, enabling appropriate human trust, and understanding for better management of the emerging generation of artificially intelligent partners.

This is not the only example of public focus on AI and machine learning interpretability. In 2016, the White House Office of Science and Technology Policy (OSTP) released a report titled, “Preparing for the Future of Artificial Intelligence,” which states that AI systems are open, transparent, and understandable so that people can interrogate the assumptions and decisions behind the models’ decisions.

Also, the Association for Computing Machinery US Public Policy Council (USACM) released a “Statement on algorithmic transparency and accountability” in 2017. It is stated that explainability is one of the seven principles for algorithmic transparency and accountability. Then it is particularly important in public policy contexts.

Other countries have also made public the demand for AI and machine learning interpretability. One example is the draft version of the Dutch AI, which is utterly focused on explainable AI, stating the utmost importance of AI systems being accurate and able to explain how the system came to its decision.

In a July 2018 report on responsible AI and national AI strategies, the European Union Commission identified opaqueness (or black-box risk) and explainability as two AI performance risks.

In AI and machine learning–driven industries Google and Microsoft have released their recommended and responsible practices in different AI-related areas, one of which is entirely focused on interpretability. The advocated interpretability practices include planning for interpretability, treating interpretability as a core part of the user experience, designing the model to be interpretable, understanding the trained model, and communicating explanations to model users.

In addition to focusing on interpretability for commercialized AI products, some companies research ML interpretability. One such example is FICO, an analytics

software company famous for using AI for credit scores. It published a white paper titled “xAI Toolkit: Practical, Explainable Machine Learning,” which includes the xAI Toolkit in their Analytics Workbench product. There are other examples of companies of great dimension that actively invest in this research field.

This means that ML interpretability can have contributions from different areas. Advances from one of the areas can improve the research of the other areas. Hence, research communities should coordinate with each other to push ML interpretability further. Moreover, impactful, widely adopted solutions to the ML interpretability problems are only possible in interdisciplinary research, bridging data science with human sciences, and including philosophy. Besides the rapid growth of the research volume on interpretability, the growing interest in ML interpretability has also been reflected in numerous scientific events. From this, it is clear that interpretability is increasingly concerning to the research community. Scientific events play an important role in promoting and encouraging interpretability research.

Furthermore, interpretability competitions have also started to show up. For example, in a collaboration between Google, FICO, and renowned universities such as Oxford and MIT, the first Explainable Machine Learning Challenge appeared at Neural Information Processing Systems (NIPS) 2018, where teams were challenged to create ML models with both high accuracy and explainability. The goal is to generate new research in the area of algorithmic explainability. Such events and competitions are shown to be a critical factor in boosting the growth of the ML interpretability field. Figure 2-8 shows the recent surge in academic publications on interpretable ML-related topics. Figure 2-9 shows the growing popularity charted in Google Trends.

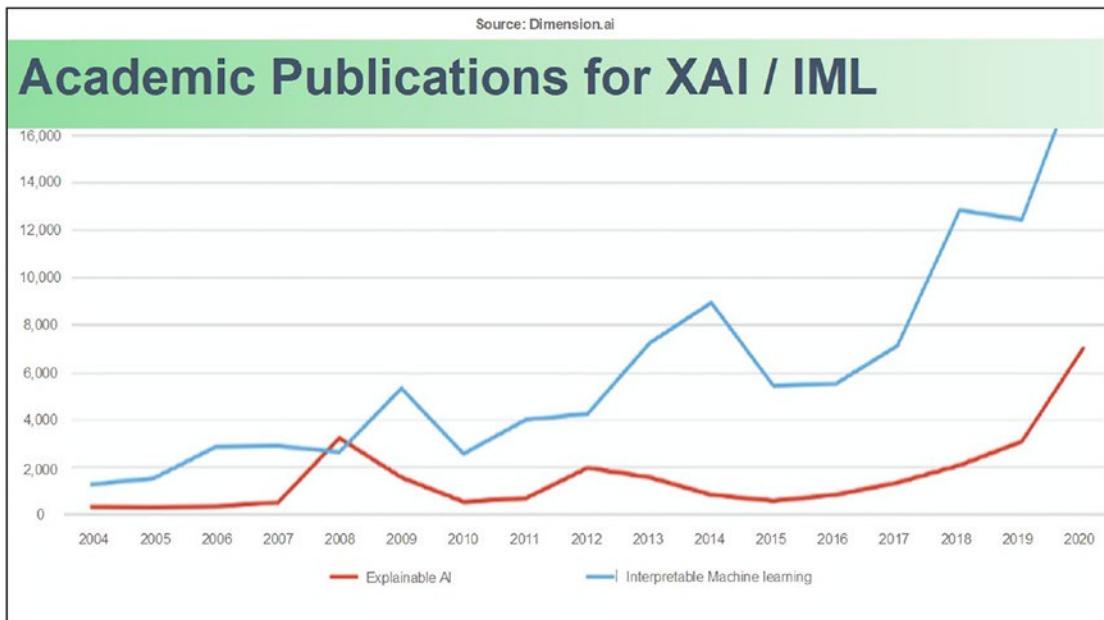


Figure 2-8. Academic publications related to interpretable ML and explainable AI

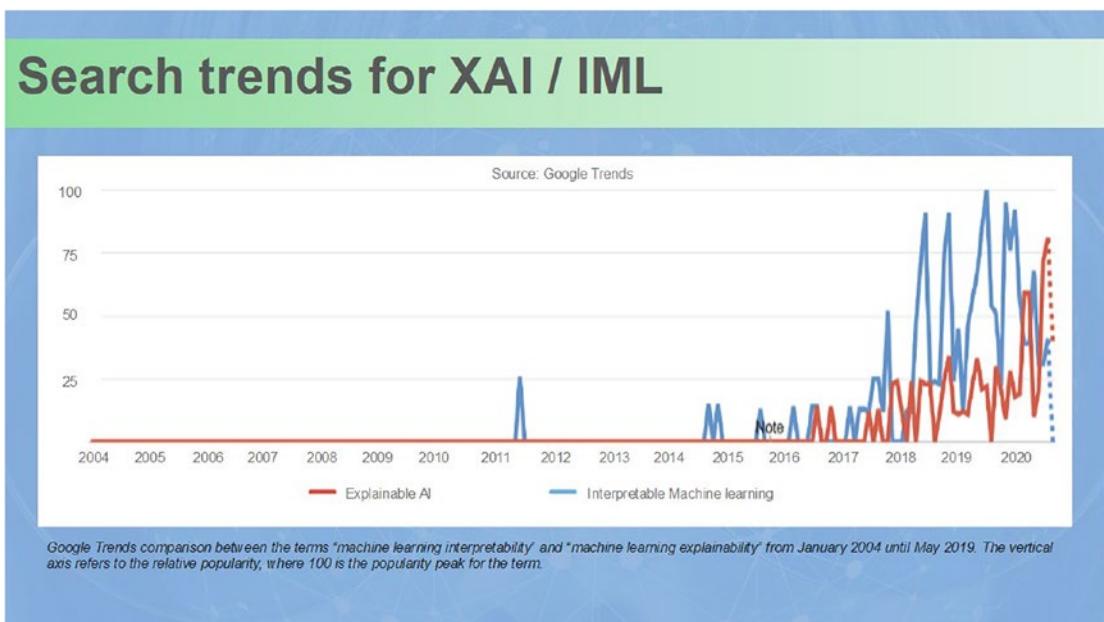


Figure 2-9. Google Trends for interpretable ML and explainable AI

The recent surge in interpretability research has led to quite a bit of confusion. It's unclear what it means to be interpretable and how to pick, evaluate, or discuss methods for producing interpretations of machine learning models. The European Parliament recently adopted the General Data Protection Regulation (GDPR), which became law in May 2018. An innovative aspect of the GDPR is the clauses on automated (algorithmic) individual decision-making, including profiling, which introduce, to some extent, a right of explanation for all individuals to get "meaningful explanations of the logic involved" when automated decisions take place. Despite multiple opinions among legal scholars regarding the real scope of these clauses, everybody agrees that the need to implement such a principle is urgent and that it represents today a huge open scientific challenge.

It is predicted that in the future, half of business ethics violations will occur through the improper use of big data analytics. Explanation technologies are an immense help to companies that aim to create safer, more trustable products. It also helps them better manage their liabilities. Likewise, the use of machine learning models in scientific research, for example, medicine, biology, and socioeconomic sciences, requires an explanation not only for trust and acceptance of results but also for the openness of scientific discovery and, therefore, the progress of research. Consequently, an explanation is at the heart of responsible, open data science across multiple industry sectors and scientific disciplines. Different scientific communities studied the matter of explaining machine learning decision models. However, each community addresses it from a special perspective and provides perspective to the explanation. Most of the works within the literature come from the machine learning and data mining communities. The first one is usually focused on describing how black boxes work. At the same time, the other is more curious about explaining the choices even without understanding the small print on how the opaque decision systems work in general.

Despite the very fact that interpretable machine learning has been a subject for quite a while and recently received much attention, today there are many ad hoc scattered results, and a scientific organization and classification of these methodologies is missing. With this book, we aim to unify such information with practical, understandable examples.

Summary

This chapter discussed interpretability. It started by defining interpretability and describing the motivation behind using interpretability methods. It discussed black-box models and why interpretability might be a great tool for understanding black-box models. It then looked at the research trends of interpretable ML and explainability. A lot of research is happening in explainability and interpretability. Google search trends indicate that the popularity of interpretable ML is increasing. You also learned about the research that is happening in interpretability. Overall, this chapter built the base for interpretability, highlighted its need, and established a case for interpretable models. In addition, it gave a glimpse of the surge in interest in interpretability, and mentioned various government laws and initiatives taken by large tech companies.

The next chapter looks at the taxonomy of interpretability and interpretability methods and their usage.

CHAPTER 3

Machine Learning Interpretability Taxonomy

The previous chapter explained why interpretability is important and the motivations behind building interpretable models. This chapter discusses the different themes under which different interpretable models can fit and where we should use specific methods. Figure 3-1 illustrates the different types.

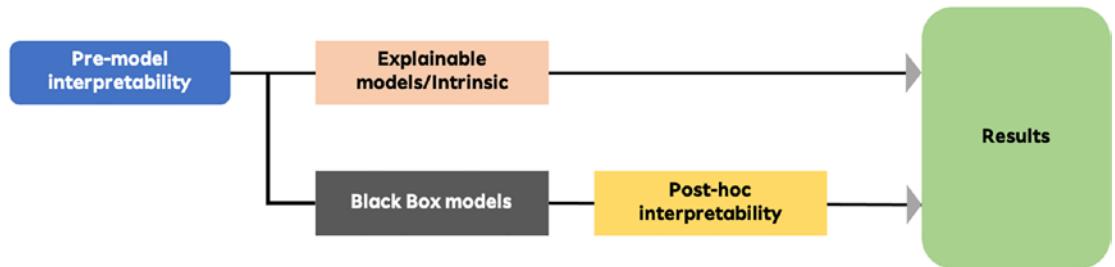


Figure 3-1. Different types of interpretability categorization based on where the algorithm is applied in the model pipeline

Interpretable machine learning techniques can generally be grouped into three categories.

- **Pre-model interpretability** uses interpretable techniques used before model building.
- **Intrinsic interpretability** uses explanations derived using the model structure.
- **Post hoc interpretability** uses explanations derived from methods outside the model structure generally run after the model has been built and predictions have been made using the model.

Pre-model interpretability is exploratory data analysis on a data set to understand the distribution of various features. It helps determine any relationships in different feature values and each feature with the dependent.

Intrinsic interpretability or explanations are computed using self-explanatory models that incorporate interpretability directly to their structures. The algorithms of this category include decision trees, rule-based models, linear models, and attention models. With the use of an intrinsic interpretable model, you might not achieve the same accuracy as black-box models; however, it becomes easy to understand the models' working because of their inherent structure. Intrinsic interpretable models can further be divided into global methods and local methods.

Global interpretability means users can understand how the model works globally by inspecting the structures and parameters of a complex model. In contrast, local interpretability examines an individual prediction of a model locally, figuring out why the model makes its decision.

After fitting a model on the data, the data scientist then analyses it to understand the model results. The process of analyzing the model using the interpretability method to extract various types of information is called *post hoc interpretability*. There are several post hoc interpretability methods that can be used in different forms on top of various models to understand the model's inner workings. Post hoc interpretability methods are implemented after the predictions are made from the model. The common types of input that go into these kinds of models involve training data, the black-box model itself, or prediction functions.

The diagram in Figure 3-2 shows how all interpretability models can be divided into different sections. Some sections focus on the separations of different techniques, while some sections focus on model-related sections.

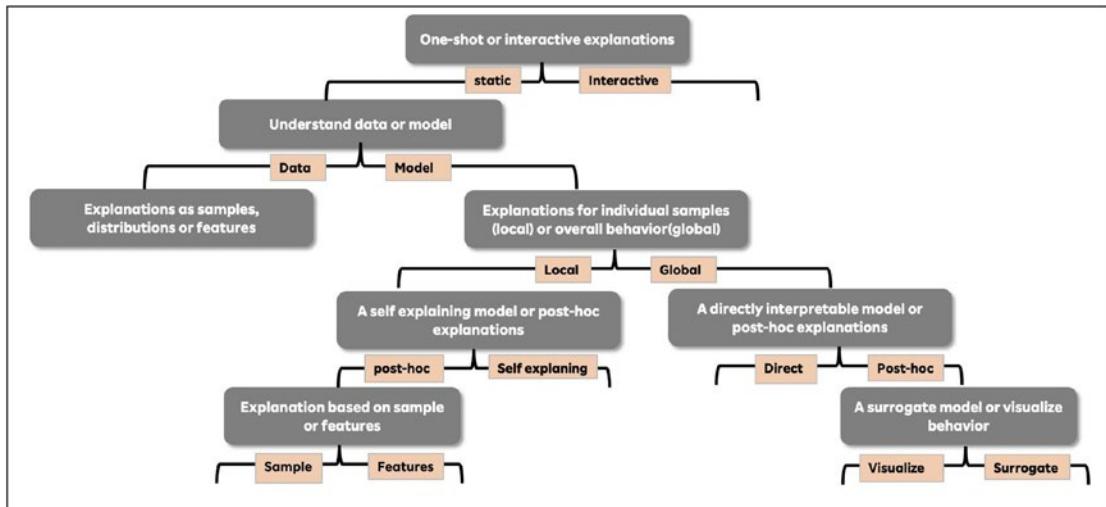


Figure 3-2. Categorization of interpretability techniques

Scope-related Types of Post hoc Model Interpretability

Based on their scope of usage, model interpretability techniques can be classified into either Global or Local methods.

Global Model Interpretability on a Holistic Level

A model can be called a *global interpretable model* if you can comprehend the whole model at once. To explain the overall model output, the interpretability methods under this umbrella require knowledge of the algorithm and data. Global interpretability means how the model makes its decisions, an overview of each of its components, including features, weights, and parameters. To give one example, let's say we come up with a metric to explain the overall error rate of the model averaged across all observations. This metric is called a *global* interpretation of the error rate of the model. In the example given at the start of the chapter, Ajay's boss wanted to know for the strategy team at an overall level which features impact or drive churn.

Global interpretability is very difficult to achieve in practice. Any model with several parameters or weights is difficult to comprehend or fit into the human memory. Additionally, we humans can only visualize three dimensions, and hence coming up

with a global interpretation solution of complex models becomes a challenging task. For models like linear and logistic, the interpretable parts are the coefficients; whereas for decision tree models, it is the split and leaf node predictions. However, sometimes models use highly modified, anonymous, or masked features. Hence it becomes difficult to interpret them from the model output. Keep in mind that interpreting a single weight in a linear model is interlocked with all other weights, which means that interpretability does not consider feature interaction.

Local Model Interpretability

The main concept behind local interpretability is to zoom in on a single instance and learn how the model is decided in this instance.

In the example, Ajay's boss wanted to utilize the power of local interpretability to understand the model's behavior for each customer. He would then derive insights from this and tell the operation team. We recently worked on a model interpretability exercise for an insurance firm and built a churn prediction model on their data. The churn model predicted which customers pay the premium for their policy in the upcoming month and which customers will default on their policy. We want to tell our readers how we utilized a model and its local interpretation in real-life scenarios.

The churn model was utilized by the operations team. The idea was to predict the probability for customers of whether they will pay the premium or not. The operation team would then group these probabilities into different buckets. Since the probabilities are in the range of 0 and 1, hence the following buckets were made. {0–0.3} were high responders, {0.3–0.5} were medium-high responders {0.6–0.8} were medium-low responders and {0.8–1} were low responders. Since we are predicting churn, hence high probability means customers will not pay the premium.

The operation team selected customers in each bucket and passed on the list to the customer care executives. The star customers care executives of the team were given the low responders' bucket because these executives historically showed more potential to convert customers to buy their policy. Within this list, we then provided why the customer is a low responder. The local interpretability result of various features was displayed on the screen as soon as the customer care executive dialed a number. The executive was able to see why the model predicted that this customer would not pay the premium, and the executive modified his conversation on those variables. After the project went live, we discussed it with the operations team, and they were very happy with the results. The churn percentage reduced by 13% after model implementation.

We were delighted and curious about how the executives benefited from the model outputs, so we had a live interaction with one of the executives. He told us sometimes, when they called a customer, the model stated that the customer would not likely pay the premium because they were not happy with the experience level of their agents. The executive would then assign a new executive to the customer, and the customer would then immediately pay the premium. Similar actions were taken differently for different customers. The executive earned more commission and was happy with the model implementation. Hence, you can see how local interpretability pinpoints the reason for a certain decision and influences actions.

Local interpretation can be made by building a surrogate model that is a simple interpretable model and can approximate a small region of interest in a complex black-box model. While not providing an optimal solution, this surrogate model is a reasonably good approximation while maintaining interpretability. The reasoning behind this is that locally, the prediction might only depend linearly or monotonously on some features rather than having a complex dependence on them. This means that local explanations can be more accurate than global explanations.

A Group of Predictions

To explain a group of predictions, there are essentially two possibilities: apply global methods and treat the group of predictions of interest as if it was the whole data set or apply local methods on each prediction individually, aggregating and joining these explanations afterward

Model-related Types of Post hoc Model Interpretability

Model interpretability methods can also be divided into two parts based on the type of support provided for a model: model-specific and model-agnostic.

Model-specific methods are limited to specific model methods because these methods utilize the internal model parameters of the function to further understand the working characteristics of the model. For instance, the interpretation of weights in a linear model is a model-specific interpretation since, by definition, the interpretation of intrinsically interpretable models is always model-specific. On the other hand,

model-agnostic methods can be applied to any ML model (black box or not) and are applied after the model has been trained (post hoc; post-model). These methods rely on analyzing pairs of feature input and output. These methods cannot access the model's inner workings, such as weights or structural information; otherwise, they would not be decoupled from the black-box model. Another property of these methods is that models are interpreted without sacrificing their predictive power, applied after training.

Result-related Types of Post hoc Model Interpretability

Another criterion that allows differentiating explanation methods is the result that each method produces or, in other words, the type of explanation that each method provides.

- **Feature summary:** Some explanation methods provide summary statistics for each feature; for example, a single number per feature, such as feature importance. Most of the feature summary statistics can be visualized as well. Some feature summaries are only meaningful if they are visualized, making no sense to present them in other ways (e.g., partial dependence plots are not intuitive if presented in tabular format).
- **Model internals:** This is the explanation output of all intrinsically interpretable models. Some methods' outputs are both model internals and summary statistics, such as the weights in linear models. Interpretability methods that output model internals is, by definition, model-specific.
- **Datapoint:** Some methods return datapoints (existent or not) to make a model interpretable. These are example-based methods. To be useful, explanation methods that output datapoints require that the datapoints themselves are meaningful and can be interpreted. This works well for images and texts but is less useful (e.g., tabular data with hundreds of features).
- **Surrogate intrinsically interpretable model:** Another way to interpret black-box models is to use an intrinsically interpretable model to approximate them (globally or locally). Thus, the interpretation of the surrogate model provides insights into the original model.

You could argue that there are other ways of providing interpretability, such as rule sets, question-answering, or even explanations in natural language. Despite this, the results account for most existing interpretability methods.

Categorizing Common Classes of Explainability Methods

Let's go over some of the categories created by combining all the methods as part of the interpretability taxonomy.

```
[static → model → local → post hoc → features]  
[static → model → global→ post hoc → visualize]  
[static→ model→ global→ post hoc → visualize]  
[static → model → local → post hoc →samples]  
[static → model → global→ post hoc → surrogate]  
[static → data → features]  
[static → model → local →self]  
[static → model → global →direct]
```

All the methods mentioned in this book belong to one of these categorizations. As we go into the details of each method, we mention which path the method belongs to. This section can be used as a reference point later to self-classify each method you learned about.

We now dive into more advanced topics of interpretability in further chapters. Now that you know the importance of interpretability and the different themes under which various methods can be classified, we believe you are ready to learn about the inner workings. The next few chapters discuss different methods and how interpretability sits in the ecosystem of predictive models. We focus on all three types of data—tabular, text, and image—and cover the methods across all themes.

At this point in the book, we think it is very important for you to take a step back and remember the major points discussed in Chapter 2. This is a good exercise to have some base before diving into more complex methodologies. To help our readers build a solid understanding of the first three chapters, we summarized what we wanted you to learn.

Summary

Machine learning is a subset of artificial intelligence that uses algorithms and statistical models for computers to perform specific tasks without human interaction.

Machine learning has grown leaps and bounds in recent years.

Supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning are different categories of machine learning algorithms.

To learn a machine learning algorithm, follow these simple guidelines.

- It is imperative to break down the algorithm into several components to understand it.
- You need to source algorithm descriptions from various sources and look for only verified sources.
- It becomes very easy to learn an algorithm using a description template. Create your template which suits your learning style.
- Once you are comfortable with simpler modules, expand the scope to incorporate more complex modules which require more coding.

Experimenting with an algorithm proves very useful to understand the inner working of the different modules.

Select an algorithm and then identify one question which comes to your mind regarding the working of the algorithm. Design an experiment around that question. Build some experiments and understand the result as to how different parameters affected the output of the algorithm.

A machine learning algorithm's accuracy measures how successful the model was to capture the patterns in training data and then use those learnings to predict unseen data.

However, accuracy alone is not enough for a model. In real-world scenarios, even if the underlying model is very complex with high accuracy metrics, the business teams sometimes find it difficult to adopt the models because accuracy does not explain why the model took a certain decision.

The black-box model is a term to describe models which have complex workings to compute an output typically spread over multiple steps. However, adopting these advanced techniques poses a few questions.

- Can we interpret a deep neural network?
- How about a random forest with 500 trees?

- Building a complex and dense machine learning model has the potential of reaching the desired accuracy, but does it make sense?
- Can you open the black-box model and explain how it arrived at the result?

Interpretability plays a major role in preventing business users and model builders from the curse of black-box models.

Interpretability is that the degree to which we can understand the explanation for a choice. Interpretability is that the degree to which a person can consistently predict the model's result.

Interpretability is motivated by the desire to make better decisions, eliminate bias, justify processes, and create repeatable operations.

Several industries are witnessing an increasing trend of leveraging ML for high stake prediction applications, which profoundly impacts human lives.

There has been a recent surge in interpretability research within both the industry and academia.

Interpretable machine learning techniques can generally be grouped into three categories.

- **Pre-model interpretability:** Interpretable techniques used before model building
- **Intrinsic interpretability:** Explanations derived using the model structure
- **Post hoc interpretability:** Explanations derived from methods outside of the model structure generally after the model has been built and predictions have been made using the model

Interpretability algorithms can be classified into global interpretable explanations and local interpretable explanations.

Another classification is in terms of whether the methods are model-specific or model-agnostic. Model-agnostic methods are not dependent on the model and produce explanations irrespective of the model's inner working; for this reason, these methods do not hamper the accuracy or predictive power to generate explanations.

Another criterion that allows differentiating explanation methods is the result that each method produces or, in other words, the type of explanation that each method provides—feature, summary, datapoints, model internals, and so forth.

CHAPTER 3 MACHINE LEARNING INTERPRETABILITY TAXONOMY

The next chapter focuses on the various properties of the interpretability methods. Few of the evaluation properties derive a close relationship with the taxonomy of the models. Hence it is important to keep the key points from this chapter in memory before going into the next chapter. These properties are very important to understand the creation of various methods from scratch and justify the underlying algorithms of various methods. The properties define the base for each interpretability method.

CHAPTER 4

Common Properties of Explanations Generated by Interpretability Methods

The last three chapters covered interpretability and the general taxonomy of interpretable methods to introduce interpretability definitions and provide an overview of different methods. Now from this chapter onwards, we slowly transition into more technical stuff about interpretability. In the upcoming chapters, you see a lot more formulas and mathematical concepts.

This chapter discusses the properties of the explanation generated by interpretability methods. Just as there are some evaluation metrics for black-box models such as accuracy, precision and recall, there is a need to evaluate the interpretability methods as the output of interpretability is used in a lot of actionable decisions. This chapter lists the different properties of several interpretable methods. Later chapters explain how you can use these properties to come up with some evaluation metrics to gauge and understand whether the explanations we produce for the models have some significance or not.

Before diving into the properties directly, you must have noticed that we have used the word *explanations* several times. Moreover, if you look at the heading of this chapter, it also says “common properties of the explanations.” Let’s discuss what an explanation is.

Explanation Defined

“An explanation is the answer to a why-question.”

The objective of an explanation is to make something clear, understandable, transparent, and interpretable. Interpretability is the end-goal that we want to achieve, and explanations are the tools to reach interpretability. An explanation relates the feature values or variables to their respective predictions so that we as humans can easily understand. This enhances the interpretability of the models.

Some explanation theories need to be studied. Some theories assume there is only one proper explanation, which means the correctness of the answer has nothing to do with whether the audience can understand it or not. Other types of theories argue that the explanations should take the listeners or users into consideration. Since different users have different knowledge bases, such theories naturally align with the assumption that something can have multiple explanations—a phenomenon called the *Rashomon effect*.

It is argued that the second type of theory is more appropriate than the first. The principle of interpretable ML is based on the theory that we are more concerned with making the audience understand the reason behind an ML model’s prediction. These explanations are related to knowledge acquisition and involve deriving an explanation by the process of inference, meaning the cause of the event is first identified and then subsets of the cause are selected as explanations.

Another objective of the explanations is that the listener receives enough information from the explainer to understand the causes of some event or decision. The explainer can be either a human or a machine.

In the end, we would like to say that explanations are highly subjective. Hence, any method that we produce to create explanations should be flexible to adapt to the needs of the human counterparts who will use those explanations. In other words, there is no such thing as a single explanation that solves all the interpretability problems. We should consider the domain, use cases, and audience for which we are trying to build the explanations.

Each situation needs to consider the problem domain, the use case, and the audience for which the explanation is aimed. Now since we understand what an explanation is when we refer to interpretable ML, let’s dive into the properties of these explanations.

Properties of Explanation Methods

Figure 4-1 highlights the different properties which form the building blocks of explanation methods. Each explanation method should satisfy one or more of the properties shown. Also, when selecting an explanation method for a use case, you should analyze which properties the method adopts and then decide to use the method.

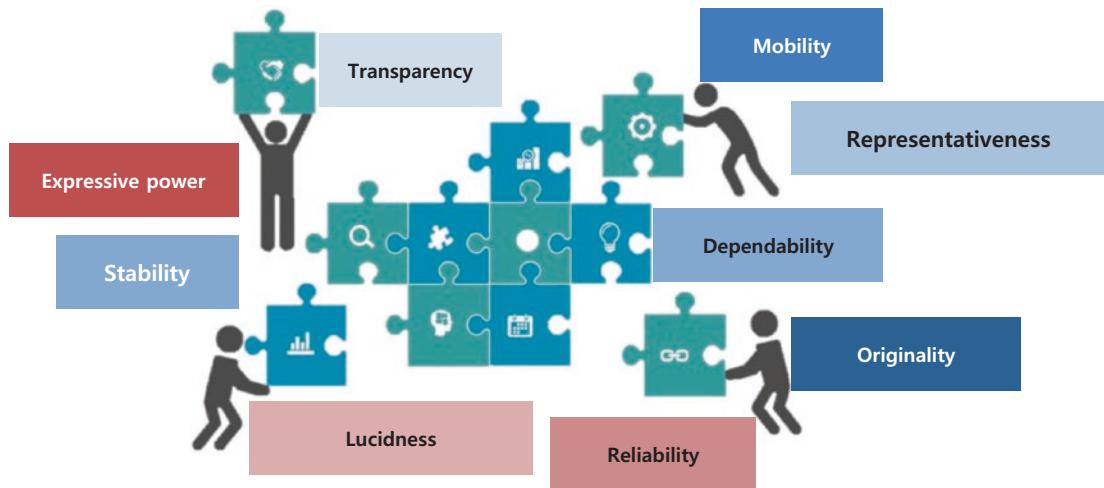


Figure 4-1. Common properties of interpretability methods

Properties of interpretable ML methods can assess and compare various explanation methods.

Template of Expression

One of the major properties of the interpretability methods is their expressive method, or how they express results. Results can be expressed in the form of rules, decision sets, natural languages, or charts.

Transparency

Model-specific explanation methods are highly translucent, whereas model-agnostic methods have zero transparency. Transparency means how much the explanation methods rely on the inner workings of the ML model, such as the model's parameters.

Mobility

Mobility is related to the variety of ML models to which a particular interpretable method can be applied. Model-agnostic methods are highly portable or mobile. They can be applied to many model frameworks as they utilize a common parameter of the model and then build the explanations themselves.

Algorithmic Feasibility

Algorithmic feasibility is related to the computational complexity of the method. Although some methods promise to give meaningful results, they are so complex that it takes a lot of time and resources to run them and get explanations. Computation time is of the main drawbacks of some of the very well-known methods. Apart from the time, some algorithmic properties need to be considered to define feasibility for the methods. Some methods include a data sampling process which means that the explanations might not be stable. This means that repeating the same instance and model with the same parameters can result in a different explanation.

Now that we have discussed the properties of the explanation methods, let's look at the explanations generated by these methods.

Properties of Individual Explanations

Certain properties apply to explanations rather than the whole method. Next, let's discuss the properties which are critical for evaluating the explanations generated by the methods. An explanation usually relates the feature values of an instance to its model prediction in a humanly understandable way.

Correctness

Correctness property is related to the accuracy of the explanation regarding the test data or unseen data. One concession we get when we evaluate an interpretability method is that if the accuracy of the black-box model is in itself, then you can accept lower accuracy in the explanations.

Loyalty

Loyalty is one of the essential properties of an explanation because low loyalty is essentially useless. By loyalty, we mean how well the explanations approximate the model predictions. High accuracy and high loyalty go hand in hand. However, some methods only provide local loyalty, which means they only approximate well to a single instance or group.

Dependability

Dependability is measured in the context of two different models trained on the same task and with similar output predictions. This property is related to how different the explanations are between them.

If the explanations are very similar, the explanations are highly dependent. However, this property is tricky since the two models could use different features but get similar predictions. In this specific case, high dependability is not desirable because the explanations should be very different, as the models use different relationships for their predictions. This property is desirable when the models rely on similar relationships; otherwise, explanations reflect different aspects of the data that the models rely on.

Resoluteness

Resoluteness represents the similarity of the explanations of similar instances. Stability compares the explanations between similar instances for a fixed model instead of dependability, which compares explanations between different models.

High stability represents the fact that small variations in the values of the model's features do not result in substantial changes in the explanations unless there is a great change in the prediction itself. Specific components of an explanation method like sampling methods can also be attributed to the lack of stability. However, stability is always desirable.

Lucidness

When we must judge how well humans understand explanations, we look at the lucidness property of the methods. Lucidness is most difficult to define and measure.

Interpretability is a subjective concept, where properties depend on the audience and the context in which the methods are used. It is a measure of how comprehensible the features are. Several modules within a method might make an explanation less understandable; for example, the transformation of features.

Reliability

Reliability is linked to the certainty of the ML model. Many ML models only provide prediction values and do not contain any statement on the model's confidence in the correctness of the prediction. The reliability property of explanations for interpretability helps instill confidence in the users of the model regarding the correctness of the model.

Significance

Significance measures how well the explanations reflect the importance of the features of their own parts. For example, if the explanations generate a decision rule list as an explanation for the model significance, which of the rule's conditions was the most important?

Originality

The originality property tests whether the prediction comes from the region in the feature space provided as input or is outside the bounds of the input feature. In case it is out of bounds, then the explanation may be inaccurate or useless. One way this can be checked is to find the data instance to be explained in the distribution of the training data.

Representativeness

Representativeness describes the coverage of the explanation (i.e., the number of instances covered by the explanation). Explanations can cover the whole data of the model; for example, the interpretation of weights in a linear regression model. On the other hand, some methods may cover only a single instance.

We talked about the properties of explanations from a technical perspective. However, we would like to remind you that interpretability is all about making models human-friendly. Hence, the next section discusses which explanation properties are desirable from a human-friendly perspective.

Human-Friendly Explanations

Explanations always are not presented in a way that makes them easily understandable. Research was conducted on publications of explanations. Most of the existing work in interpretability uses the intuition of the method developer or researcher on what they think is human-friendly. However, there should be a standardized format for making explanations human-friendly. Let's look at some of the desirable properties of human-friendly explanations.

Contrastiveness

Humans are a curious species. We generally ask why. However, we don't stop at why. We tend to dig deeper into the why, and then the next question is why not? Hence humans generally ask why a certain prediction was not made instead of asking why a prediction was made. This is the tendency to think about counterfactuals (to think in opposite terms of the stated facts). Sometimes stakeholders are not interested in factors that led to a particular explanation. Instead, they find more value in determining what factors need to be changed in the input so that the model also changes its predictions.

Selectivity

Explanation methods should be selective. They should provide only relevant information about the model and its features. The methods should only explain the main cause of the event around which model is built. Since the objective is to simplify the complex black-box models, producing complex explanations that result in confusion and waste time and resources. A lower number of explanations ensure that they are easy to comprehend. The Rashomon effect describes situations in which different causes can explain an event. Since humans prefer to select some of the causes, the selected causes may vary from person to person.

Social

Another important property of the explanations on being human-friendly is that they are a part of the social interaction between the explainer and the listener. While picking up a suitable explanation, you should keep this factor in mind and only pick relevant ones for the ecosystem of the model and the target audience. The best explanation should always vary between domain and use cases.

Focus on the Abnormal

One of the important aspects or properties of the explanations is to include the abnormal cases. This is essential for giving complete information to the users. This enables users to understand the model and why certain decisions were taken completely. Eliminating these abnormal cases might result in the generalization of explanations. We should ensure that abnormal cases should be included in the explanations even if they produce the same influence on the model as the more frequent explanations.

Truthful

Good explanations are proven to be true in the real world. This does not mean that the whole truth must be in the explanation, as it would interfere with the explanation being selected or not, and selectivity is a more important characteristic than truthfulness. With respect to ML interpretability, this means that an explanation must make sense (plausible) and be suitable to predictions of other instances.

Consistent with Prior Beliefs

People tend to ignore information that is inconsistent with their prior beliefs. This effect is called *confirmation bias*. The set of beliefs varies subjectively from person to person, but there are also group-based prior beliefs, which are mainly cultural, such as political worldviews. Notwithstanding, it is a trade-off with truthfulness, as prior knowledge is often not generally applicable and only valid in a specific knowledge domain. Honegger argues that it would be counterproductive for an explanation to be simultaneously truthful and consistent with prior beliefs.

General and Probable

A cause that can explain a good number of events is very general and could, thus, be considered a good explanation. This seems to contradict the claim that abnormal causes make good explanations. However, abnormal causes are, by definition, rare in the given scenario, which means that, in the absence of an abnormal cause, a general explanation can be considered a good explanation. Regarding ML interpretability, generality can easily be measured by the feature's support, which is the ratio between the number of instances to which the explanation applies and the total number of instances. These properties are fundamental to interpretability in machine learning, although some might be more relevant than others depending on the context. No matter how correct an explanation is, if it is not reasonable or appealing to humans (human-friendly), the value of ML interpretability vanishes.

Summary

This chapter briefly discussed three important interpretability concepts: explanations, the *properties of interpretable methods and explanations*, and *human-friendly explanations* and their properties. These topics are important to the theory behind interpretability and how it should be perceived in real-life scenarios. The next few chapters explain these properties in detail. These properties are mostly used in the assessment of interpretability methods. They help us understand human factors in interpretability and how humans perceive the usage of interpretability techniques.

CHAPTER 5

Human Factors in Model Interpretability

The previous chapters discussed properties and assessment criteria for interpretability methods; however, keep in mind that explanations are designed for humans. Hence, the human factor in designing explanations is very important.

Few main components of an explanation system and its implementation involve goals, strategies, and processes. This chapter covers all of them from the perspective of humans. We have covered some examples quoted by actual practitioners of explainability.¹

Like any machine learning exercise that needs to impact the business, interpretability also requires cooperation and brainstorming between people in various roles within a company. This builds trust in the explainable systems. This trust is not limited to trust between people and models, but regular interaction also establishes trust between people within the organization.

Let start by talking about interpretability roles.

Interpretability Roles

Stakeholders of an interpretability exercise can be divided into three main categories: technical expertise builders, domain knowledge reviewers, and model consumers.

¹Please refer to “Human Factors in Model Interpretability: Industry Practices, Challenges, and Needs,” by Sungsoo Ray Hong, Enrico Bertini, New York University, and Jessica Hullman, Northwestern University (<https://arxiv.org/abs/2004.11440v2>).

Technical Expertise Builders

These roles generally consist of data scientists or ML engineers within the organization. These individuals develop, design, or test models and integrate systems within the organization's data infrastructure.

Domain Knowledge Reviewers

Reviewer is an interesting categorization. These individuals have the domain knowledge and sit right next to technical expertise builders in terms of importance. They work with experts to give them feedback about models and ensure that models meet the desired goals and behave as expected. These individuals may or may not have the technical knowledge about ML. Reviewers provide different perspectives to fix potential issues. The reviewer category can be very broad; hence, breaking it down into further roles is beneficial.

- **Domain experts:** Domain experts have deep knowledge of what real-world phenomena the models need to capture. For example, doctors in the healthcare product or analysts in credit evaluation models.
- **Product managers:** Product managers are responsible for the development of the actual product the organization needs. Requirements and business goals are the major communication agendas for a product manager, and they also ensure that all goals are met in time.
- **Auditors:** Auditors examine models from the legal perspective. Their goal is to make sure the model satisfies legal requirements from a compliance standpoint.

Stakeholders or End Users

Stakeholders are the end users of the information and decisions that the models take. This group consists of individuals like

- doctors working with mortality and readmission models
- bank representatives who use loan approval models to take whether a loan should be granted or not

- engineers who must predict when machinery will fail or what the output patterns could be
- a biologist who might want to understand the effects of a drug

These individuals may or may not belong to the company to which the technical experts or reviewers may belong. This depends on the business model of the organization. For services companies that offer ML services to other organizations, stakeholders generally belong to the client organization. For companies that do in-house ML consulting, stakeholders belong to different teams within the same organization.

One important aspect to keep in mind regarding these roles is that they do not necessarily define a single individual. One individual can cover multiple roles, and a single role might be valid for one individual or a whole team.

Interpretability Stages

At different stages of the model and product development, interpretability plays a major role.

Let's talk about various stages from a human perspective and how interpretability fits into such stages.

Ideation and Conceptualization Stage

Interpretability plays a major role even before the actual model is developed. Individuals spend a lot of time before model building to explore strategies and metrics.

Technical experts may involve domain reviewers to assist in the design phase of feature sets to choose features that make sense from the point of view of domain knowledge. For instance, finance professionals know about the importance of having the perspective of business analysts early on in defining features for credit evaluation models.

Similarly, some technical experts and auditors work together (especially in heavily regulated industries) in the early stages of the process to define a framework that explains each feature, how it is engineered, and why using it does not risk a legal violation. A data engineer in a major bank explained that designing features with the compliance team makes it easier to gain their trust later in the verification and validation stage.

Building and Validation Stage

In this stage, the technical experts build and validate models. They deploy various interpretability techniques to understand why the model is giving a certain output. This is where the amount used in the taxonomy of model interpretability is maximum. The technical experts pick an interpretable method to explain a particular observation or the overall model behavior. Also, choosing a model-specific or a model-agnostic method is done while building the model and the validation steps.

Complementing this activity, technical experts identify edge cases, often called *anomalies* or *corner cases*—that is, cases for which the model is particularly uncertain or may behave unexpectedly. Even if limited in scope and completeness, reasoning around cases is how experts often interpret and test a black-box model.

Further, the experts also investigate the features used in the model as a part of the interpretability lens to better understand the model. Experts mostly look at how sustainable and plausible a model is by looking into what features drive most decisions. This is typically done by ranking features by importance. In such scenarios inspecting less critical features can be as beneficial as inspecting more important features.

Information about feature importance is commonly shared with other stakeholders and helps them reason how the model behaves. Technical experts also stress the fact that comparison of multiple models becomes very easy with interpretability. Interpretability is suitable for understanding one single model and can develop an evolving set of models that can be compared and refined. Although, for many users, model interpretability is the preferred model comparison method, there is no established methodology for doing so.

In this phase, there are two interpretability needs.

- **Gain confidence.** Technical experts need to gain confidence in the reliability and validity of the models they build.
- **Gain trust from stakeholders.** Model stakeholders sometimes hesitate to use the model for final decisions when the models are black box, and the logic is too complex. It becomes difficult for them to establish trust. This problem is severe in high-stakes environments where even a single mistake can have very costly consequences.

A senior data scientist working on a model for intensive care units (ICUs) remarked, “I don’t think we can get away with providing a black-box model. I don’t think that’s going to work as much as people say. If you don’t explain it to them (model consumers), I don’t think they will ever want to use it.”

In addition to model analysis and validation needs, model technical experts have a strong need to communicate to other stakeholders how models behave, how much they can trust them, and under what conditions they may fail. The quality of communication matters; it encourages insights that can be used to improve models and obtain organizational trust. Communication between different stakeholders is often iterative and may motivate further modeling to capture insights about how a model is doing.

One of the strongest needs model technical experts describe is devising methods and tools to interact and communicate with model reviewers. Involving reviewers in the validation process using language and representations they can understand is one of the main challenges.

Many people also believe in a “curse of knowledge;” model technical experts often had difficulties identifying what other stakeholders don’t know. Even if they grasped stakeholders’ knowledge level, determining how to deliver the insights seemed difficult, even with visualizations.

Deployment, Maintenance, and Use Stage

Once the technical experts finalize the model after several reviews and enhancements, deploy the models by integrating them into the actual data feed. In the integration phase, the interpretability issues are not limited to understanding the model behavior but also extend to the whole infrastructure around the model.

There might be a bit of confusion around the goal of interpretability as to whether it is for understanding how the model works or how a whole model pipeline works. However, the ecosystem in which models exist is as important as the model itself. In these scenarios, interpretability issues may arise due to the black-box nature of the model and the complex pipelines that are part of the software. The complexity in the flow of information within a model framework can also cause interpretability issues.

In the life cycle of an ML system, interpretability challenges arise after a model has been put in production. Some of the relevant problems are deciphering how to check instances/prediction patterns that do not reflect the state-of-the-art behavior and checking why mode has made an unacceptable error while supporting high-stakes decisions. Such problems are detected by the team which is responsible for monitoring the model. Once the error is identified, it is classified whether we need an interpretability method to determine the cause of the error or not. This type of “root cause analysis” can be particularly challenging.

CHAPTER 5 HUMAN FACTORS IN MODEL INTERPRETABILITY

Data scientists are often required to build a better version of ML models as a new version with learnings from previous versions might improve performance. In such cases, model comparison becomes a part of interpretability. Comparison between different models often requires the support of tools in scenarios where multiple models are deployed in complex systems. Since model changes can affect interpretability, model technical experts need to closely watch the model's parameters. Sometimes the experts keep the model inputs constant to understand the effect of the changes on the interpretability of different models.

There were two main situations where an explanation was crucial. In high-stakes situations, explanations play a very major role. For example, when physicians use models to support making decisions about patients, they need to understand how the model "reasons" to reconcile it with their own knowledge.

The senior data scientist of the same firm remarked that explaining a prediction simply is not enough sometimes, and the doctors or physicians are inclined to understand what information about patients can be manipulated to make the model behave differently. The explanations can be used as evidence to support a decision when the users of the model agree with the model's decision, or they can find new insights in scenarios where the model does not agree with the human's idea of prediction. In scenarios where a human disagrees with the model's prediction, interpretability plays a very important role. The whole idea behind the ML model is that the models should help humans become more intelligent and efficient. But when suddenly the predictions which may be better than what humans can make cannot be explained to humans, it becomes for the humans to learn these scenarios and, in turn, improve their own predictions or knowledge. Sometimes, when a model's decision goes against the desired goal in a real-life scenario, the explanations become crucial.

A good example of such a scenario can be a case in which banks use models to support representatives meeting with their clients.

The AI head at a bank responsible for business credit evaluation mentioned two main situations in which explanations are crucial in the organization. First, when a customer's loan is denied, the representative needs to justify (the "right to an explanation" is part of many countries' legal code). Second, representatives have a strong interest in granting more loans; therefore, they are always seeking explanations they can act on, such as those that suggest how to turn declined loans into accepted ones.

Interpretability Goals

The previous section touched upon specific goals various stakeholders may have at different stages of model development. This section summarizes the set of goals we encountered to connect goals more explicitly to roles and processes. We identified three broad classes of interpretability goals.

Interpretability for Model Validation and Improvement

One of the most important activities during model development is identifying issues with the model and finding solutions to fix them. The model technical experts and reviewers should fulfill this goal during the model development journey. However, they also sometimes need inputs to effectively achieve this goal. For example, auditors and legal teams may help experts and reviewers to identify a stringent set of legal requirements while fixing any issues. One of the ways to validate models is to check up on aggregate statistics of the model's outputs; however, we need interpretability tools and methods on top of it.

Sometimes models can learn non-essential and non-meaningful relationships between input and output with high accuracy. This might make the involvement of domain experts necessary for the proper interpretation of results. Identifying irrelevant correlations and false implications of causation may require the involvement of experts who understand the model and its mechanisms. This process is characterized by contrasting model behavior with the stakeholders' perceptions of the actual meaning of the data and model predictions.

From a legal standpoint, transparency is needed to make sure models are not violating legal regulations such as the Fair Trading Act or General Data Protection Regulation, which aim to prevent discriminatory practices. When a model makes a mistake, it is crucial to explain why a given decision was made. Compliance is one of the main reasons why interpretability is needed.

Interpretability for Decision-Making and Knowledge Discovery

The whole idea behind interpretability is to assist with decision-making. For high-stakes problems, decision-makers cannot trust or use predictions that do not provide a rationale for their recommendations. Similarly, explanations need to match the expert's needs and the constraints of the domain. One of the important themes around interpretability is actionability.

Actionability is thus by far the most important emerging requirement of interpretable systems. Every interpretability method comes up with certain explanations; however, not all explanations are actionable. Some very common methods give good explanations and may only increase the knowledge about the model and its behavior. However, one of the main advantages around interpretability should be its ability to provide actions or recommendations. Model experts need to have multiple iterations to come up with actionable features. This sometimes also requires multiple discussions with reviewers and stakeholders.

A potential of a tool to explain its decisions serves as a catalyst for learning or knowledge discovery. For example, in a decision support context, model interpretability becomes critical when a decision-maker and a model disagree. That learning is an important result of interpretability in these situations. Without explaining why a model makes a prediction, there is no way a human can reconsider a decision or learn from a model. There are cases in which models are not necessarily built for decision-making or automation but primarily as a vehicle to generate insights about the phenomena described by the data and understand KPI (key performance indicators) in the future. The goal is to understand which features drive performance rather than to predict future performance.

Based on the system feature(s) used in the past, user retention models can show the likelihood that a user will return to the service.

Interpretability to Gain Confidence and Obtain Trust

Every model expert agrees on the need to gain trust in a model. Debugging is essential to gain confidence in the fact that the model is working properly. This individual-level trust is typically the focus of ML interpretability research to date. Even in cases where customers or stakeholders do not require transparency, data scientists should

thoroughly investigate their models to build understanding and trust as part of necessary due diligence. However, persuading others that a model can be fully trusted often takes precedence in the interpretability work. Lack of trust in the enterprise of model building, or the work of the team developing those models, could prevent progress in the use of ML in the organization. Hence, model technical experts need to gain trust in their models and procedures and may rely on interpretability to persuade others in different roles in the organization that their work provides value.

Being able to concisely and effectively explain how a model works and why it should be trusted is not easy, especially because some stakeholders may need to understand this at a high level of abstraction and with little technical detail. Visualization methods can play a major role in the communication stage of interpretability. The high priority on building trust as a modeling goal for model experts means they perceive the need to build transparent models or use model extraction methods to explain to others how a given model works.

Human-Friendly Themes Characterizing Interpretability Work

Research on interpretability to produce new tools or insights has primarily focused on the interaction between an individual and a model. Interpretability is described as a property of the model class based on its relation to human expectations or constraints, such as monotonicity or additivity, or human-driven constraints that come from domain knowledge or the presentation of a specific model's results that enables a human to understand why a label was applied to an instance (e.g., LIME or SHAP).

We summarize four themes that are different from the technical characterization we mentioned in the previous chapters. These characterizations are more human-centric and talk about how humans can work together with interpretability to make models more useful.

Interpretability Is Cooperative

Interpretability in an organization means collaboration and coordinating of values and knowledge between stakeholder roles. This theme is defined majorly by mentions of discussion with other stakeholder groups, such as domain experts. These discussions

frequently occur during ideation and model building, and validation, but also after deployment. Collaboration around interpretability is important for improving business reasoning and convincing stakeholders that the models that have been built will bring value.

Interpretability's role in building and maintaining trust between people in an organization is another clear way its social nature was evident.

People respond to use cases and anecdotes way better than they respond to math. Hence, the proof and the need for interpretability should be adopted when communicating to a client. It's usually to explain the operation of the system and then give some answer.

When public relations, tech transfer, or other teams further from model development gained trust through interpretability, limitations or constraints on models tended to be more easily accepted with a model.

The internal trust frees up teams from constraints on the types of models they could use or the level of monitoring required, such that model interpretability was seen as a means to obtaining a certain status.

Sometimes interpretability's importance was not that it could explain any decision, but that the act of including it alone signaled a "due diligence" that other stakeholders or end users found comforting.

Interpretability Is Process

Existing characterizations of interpretability, such as measuring mismatches between a human mental model and an ML model rarely comment on the potential for interpretability to occur in time. Interpretability is a persistent concern. While interpretability might manifest differently across stages (e.g., as feature selection early in the pipeline vs. identification of a model's consequences to existing decision pipelines after deployment), it is naturally associated with a diverse set of practices and intents that might evolve or co-occur but which all contributed to the understanding of what interpretability means.

Interpretability can foster "dialogue" between a human user and a model, which is perceived as necessary to maintain for learning and continued use during the model's lifespan.

Interpretability Is a Mental Model Comparison

The types of learning and sense-making that interpretability was perceived to bring about in an organization often occurred through comparisons between mental models held by different stakeholders. This contrasts with interpretability definitions that frame it as determinable from interactions between an individual mental model and the ML model. While an ML model to human comparison was implicit in many participants' descriptions around interpretability, that the ML expert's own understanding of the problem had a mediating influence on how they pursued interpretability was apparent.

Considerable effort is required toward understanding end users' mental models to inform their work. Reflecting on aspects of human decisions refines and puts in perspective interpretability goals.

Interpretability Is Context-Dependent

Interpretability solutions built by the teams are strongly shaped by the needs and use cases of the user groups for which they were intended. Impacts of interpretability work are intimately related to decision-making infrastructures that could be complex.

Design Opportunities for Interpretability Challenges

This section discusses the challenges that come into play when implementing interpretability from a human point of view.

Identifying, Representing, and Integrating Human Expectations

To identify and integrate human expectations of interpretability, it is very important to understand and identify the edge cases. This edge case identification can happen by a group of people using the model or by an individual

The following are some of the benefits of integrating interpretability in a model-building exercise.

- It helps humans articulate the expectations they have with predictions in interpretability tasks.

- It helps humans effectively recognize gaps between what they expect from a system and the actual representations.
- It helps humans debug/change model behavior based on their findings.

Data scientists often engage in such collaborative work. Having access to stakeholders' responses to other different types of models is an important component of model development and building stage.

One beneficial approach can be developing interactive applications that stakeholders or end users can use to efficiently provide feedback on the model usage. Researchers have developed platforms to find errors made by predictive models to gather feedback from a group of people. Interactive applications and approaches can enable domain experts and ML experts to engage with models and share knowledge about decision processes.

Communicating and Summarizing Model Behavior

Since the models are cooperative and social in the building phase, a lot of knowledge transfer happens in model hand-offs. To identify the model bugs that occur after deployment, it is very beneficial to have visualizations that can capture the relationship between the model and those receiving the models. In short, a good summary of model behavior provides a great deal of relief to stakeholders and end users to understand the working of the model and enable developers to solve model bugs quickly.

These can be developed along with the algorithms which identify the edge cases.

One important concern is to improve approaches where the output does not agree with human behavior or perception.

Scalable and Integrable Interpretability Tools

For many interpretability methods, best practices or guides regarding the integration of interpretability tools do not exist because they were developed in academic settings, and the industry needs to catch up on the trend of the interpretability usage for production-level model jobs.

It is difficult for tools to integrate with existing platforms because they cannot be adapted to their specific environment. In some cases, the problem is that the interpretability tools cannot adapt to the data size.

Another challenge is that interpretability can be a great tool to compare models; however, not many methods provide insights on model comparison. Cases for model comparison include comparing parameters, selecting features, timestamps, and more.

Post-Deployment Support

In the post-training phase, better tools are needed to monitor models once they have been deployed.

Visualization tools are a great way to monitor model behavior. Automated anomaly detection may be useful in addition to identifying potentially troubling behaviors, given an appropriate specification of critical expectations that should hold, such as one learned from domain experts.

Summary

This chapter described different interpretability roles. We described how each role works in a modeling ecosystem and contributes to explainability. Next, we described the various stages of interpretability or explainability exercises. We talked about how each role is connected to different stages of building and applying explainability methods. Finally, we went over the different goals of interpretability methods. We talked about how these goals can be aligned with both roles and stages of method development and application. The chapter ended by discussing various method themes and how they can be defined using a human-centric approach.

The next chapter describes how to assess various explainability approaches based on several properties and factors. Chapter 6 sets a base for understanding the primary requirement for building or choosing an interpretability or explainability method for your modeling tasks.

CHAPTER 6

Explainability Facts: A Framework for Systematic Assessment of Explainable Approaches

In the previous chapters, you learned about explanations' properties and the human perception of explanations methods. From this chapter onward, we slowly transition into the framework of interpretable methods and explanations. Explainable systems can be assessed along five key dimensions.

- Functional
- Operational
- Usability
- Safety
- Validation

Using the knowledge from this chapter, you can compare various interpretability methods, understand their capabilities, and identify their discrepancies, theoretical qualities, and implementation properties. This chapter introduces a framework that assesses the methods based on functional and operational requirements. We think it is very important for you to understand the properties and the procedure of designing

CHAPTER 6 EXPLAINABILITY FACTS: A FRAMEWORK FOR SYSTEMATIC ASSESSMENT OF EXPLAINABLE APPROACHES

the framework of interpretability methods to thoroughly understand each method later in the book. Every explainability method designed for predictive systems should be accompanied by a framework that assesses its functional and operational requirements. The quality of explanations should be evaluated based on usability criteria to understand it from a user perspective. Their security, privacy, and vulnerabilities should also be kept in mind before the method is designed. Hence, a standardized list of explainability properties spanning all five dimensions facilitates easy evaluation and comparison of different explainability approaches.

Explainability Facts List Dimensions

The facts list of explainability is supposed to evaluate explainable systems across multiple dimensions.

The functional requirement considers algorithms parameters such as

- type of problem the explanations apply to
- whether the explanations are designed for data, model, or predictions
- which technique uses the explanations (i.e., local vs. global)
- the relation to the predictive system (i.e., post hoc or intrinsic).

This was covered in Chapter 3.

The operation requirements include interaction with the end user, accuracy vs. interpretability trade-offs, and user background knowledge to fully utilize the explainability power.

The usability requirement includes listing the properties of the explanations, making them more human-friendly and natural so that the output is easily understandable.

Some of these properties are what we talked about in Chapter 4.

Robustness and security of an explainable approach are the two safety requirements. Safety requirements include robustness and security of an explainability approach. For example, it includes how much information is leaked to the explainability approach of the underlying model. Also, it checks whether the explanation from a single data point is constant across different models by using the same method.

Let's now discuss each of these requirements in detail.

Functional Requirements

The functional requirements for designing an explainability method closely relate to the taxonomy of the interpretability methods discussed in Chapter 3. A data scientist can use the list of functional requirements (F1-F9) to identify and deploy the most suitable algorithm for a particular use case. These are discussed in the following sections.

F1: Problem Supervision Level

Explainability approaches may be relevant to supervised problems and apply to unsupervised learning methods and some semi-supervised learning methods.

F2: Problem Type

There are different types of problems in machine learning for which explanations can be designed. For example, we have a classification (binary, multiclass, multilabel, etc.), regression, clustering, recommendation algorithms such as ranking, and collaborative filtering. Once you define and identify a problem based on the methods mentioned or other methods, you can easily identify explainability methods suitable to the requirements.

F3: Explanation Target

Any machine learning pipeline has three major components: data points, models, and predictions.

Explaining the data points does not come under the umbrella of machine learning explanations or interpretability because there are other methods, such as summary statistics, class ratio, and feature correlations. The target of explainability methods is generally the underlying model and its predictions. By understanding the model's behavior and the predictions, we can easily decide on which explainability method to choose. The explanation of the model is based on its general functionality and conceptual behavior.

F4: Explanation Breadth/Scope

The scope is one of the major assessment criteria for picking or designing any explainable model. The scope is similar to the taxonomy scope discussed in Chapter 3. Based on the requirements, you can define the scope as local (any single explanation) or global (the overall data or a group of explanations within the data).

F5: Computational Complexity

Another important assessment criteria are regarding the computational complexity of the modules in the explanation algorithm. The complexity can be measured around metrics such as the time to generate results and the memory required to compute the explanations. This should generally be coupled with the explanation scope mentioned. Designers should consider whether the time and computational resources increase exponentially with an increase in scope from local to global. Various optimization frameworks should be put in place to compute explanations on optimized time and resources.

F6: Applicable Model Class

Model class assessment is very important for designing explainability algorithms. This is also closely related to the scope we discussed in Chapter 3. Model class means determining whether the methods apply to a specific model or applicable across all models (i.e., model class-specific methods or model-agnostic methods).

F7: Relation to the Predictive System

Two main relations exist between explainable systems and modeling pipelines. The explainable systems can be either post hoc. They are designed to work on top of existing models and utilize the model file or predict the model functions; for example, LIME or SHAP techniques discussed in upcoming chapters. In intrinsic techniques model itself is used for both predicting and explaining (e.g., explaining a linear regression with its feature weights). Examples of post hoc methods are local surrogate explainers.

Another additional relationship can be in the form of a surrogate system that mimics the patterns of a complex model using a simpler model. Surrogates need to be carefully designed to not introduce the same level of complexity as the original model.

F8: Compatible Feature Types

Every method should be designed keeping in mind the guidelines for different types of features. The compatible feature types might be categorical, numerical, or ordinal. Algorithms with optimization as their backbone do not work well with categorical features. Some selected model implementations require categorical features to be preprocessed, for example, one-hot encoding, thus making them incompatible with some explainability approaches. Every method should describe the feature types to judge whether they belong to a particular hierarchy and how they are helpful to the model.

F9: Caveats and Assumptions

In the end, any aspects of explainability that do not lie within the bounds of the categories mentioned earlier should be included in a separate functional class. Assumptions of feature independence, effects of correlated features on explanation quality, or simple requirements like feature normalization when explaining weights should be appropriately segmented into categories.

Operational Requirements

The operational requirements (O1–O9) are another important aspect to consider after functional requirements and are discussed in the following sections. These characterize how explainable systems interact with users and what is expected. These are important from a deployment and automation point of view.

01: Explanation Family

The explanation family is the first assessment that should be done for explanation methods in light of their operational requirements. While assessing the explanation methods, we should consider the family to which they belong, and there can be multiple types as described next.

- Explanations can belong to a family of the relationship between model internals, or relationship between features, relations between features and predictors; for example, some explanations can be based on feature importance of various features.

- Explanations can also belong to a family of contrasts and differences (using examples); for example, prototypes and criticisms (similarities and dissimilarities) and class-contrastive counterfactual statements.
- Explanations can belong to a type that represents a full causal model.

02: Explanatory Medium

Different explanation systems use different mediums to express the working of the model they are trying to explain. The most common medium for an explanations system to express results is visualizations, in forms of text or a mix of both. Some methods may also represent a summary of models in the form of coefficients, summary statistics, and plots like PDP (partial dependence plots (see Chapter 9)) and ICE (individual conditional expectations (see Chapter 9)). In text form, the explanations have a variable and a simple text to explain the variable's meaning in the model's context. An explanation system that has a framework that is well defined can provide the listener an opportunity to argue against it or accept it with complete knowledge.

Sometimes a mixture of a plot and natural language representations may become necessary because visualizations are confined to three dimensions, so including texts may prevent explanation systems from the curse of dimensionality.

03: System Interaction

System interaction plays a major role in determining how the users interact with the explainable systems. The communication framework can either be dynamic or static (i.e., with or without feedback). In a static system interaction, the explainable method gives an output based on predefined protocol by the technical expert. In such a case, the output may not always satisfy the user's expectations. One example can always be to output the most significant feature when a user may be interested in a different set of features. The other type of system interaction can be dynamic interaction. This enables the user to explore all details about a particular explanation. Examples of this kind of system interaction are web interfaces. The creator of such systems should clearly indicate how feedback impacts the explainable system.

04: Explanation Domain

The explanation domain can be very different from the domain of the model. For example, a model may take image data as input; however, the explanation may be a text instead of saliency maps laid on top of the image. Similarly, the model can be a tabular data model; however, the explanation system output might be a rule-based text paragraph. Assessing any method explanation domain is very important to consider as it is crucial in defining alignment between the model builder and stakeholders.

05: Data and Model Transparency

Another requirement on the operational assessment of the explainable methods is the underlying transparency requirement. Sometimes the data that goes into the model is transformed. Also, before using the explanation system, users need to assess their knowledge on the inner workings of the model. Users should also check whether the features are human understandable or not. For example, when predicting room temperature, you can either have a simple room temperature variable in the output or a squared sum of the room temperature and the room height. When the input is not easily understandable, the system designer may want to provide a list of data transformations.

06: Explanation Audience

The audience for any explanation method may vary from a domain expert to a person who is unaware of the model and explainability domain. There can be two types of audiences within domain experts: one with AI/ML knowledge and the other with only the business domain knowledge. Therefore, being vigilant about the level and type of background knowledge of the audience comprehending a method becomes very important. Furthermore, the transparency of the features (or the model) should be considered with respect to the intended audience. For example, consider a system that explains its predictions using natural language sentences. Given the language skills of the recipient, the system can use sentences of varying grammatical and vocabulary complexity to facilitate easier comprehension; for example, explaining a disease diagnosis to doctors as opposed to patients or their families.

07: Function of the Explanation

One of the operational requirements for proper assessment of explainability approaches is that every method should be accompanied by a list of its practical applications.

Most of the methods are designed for transparency (i.e., explaining a component of the ML pipeline to an end user). Different components can be whether this method is to support systems or compare models. It is important to provide the user with the validated deployment context to prevent its misuse, leading to unintentional harm when deployed in high-risk applications or autonomous systems.

08: Causality vs. Actionability

Explanations are generally not of causal nature. This property needs to be communicated to the users so that they don't make incorrect conclusions. For actionable and not causal explanations, if the users tend to assume that the insights are causal, it might result in very inaccurate insights, as in popular methods like SHAP. Similarly, explanations derived from a full causal model should be advertised as causal and used to their full potential.

09: Trust vs. Performance

One important assessment criterion for explainability approaches revolves around performance. An elaborate discussion on performance vs. explainability trade-offs is very important. Majorly the job of explainability is to build the trust of users in predictive systems. However, sometimes there is a decrease in performance in quest of making the models more explainable. Users need to choose methods that do not drastically decrease accuracy or other performance-related metrics.

Usability Requirements

Here, we discuss the properties of explanations that are important from a listener's point of view. Many of these are grounded in social science research; hence, whenever applicable, make algorithmic explanations feel more natural to the end users, regardless of their background knowledge and prior experience with this type of system or technology. Also, we may find a lot of similarities between the usability aspect of explanations and the properties of explanations discussed in Chapter 4.

U1: Soundness

This property measures the truthfulness of an explanation. This property is not valid if the explanation is intrinsic because the prediction and explanation are derived from the same model. In post hoc explanations, this property measures the amount of error introduced due to the explanation method.

This can be done by calculating a selected performance metric between the outputs of predictive and explanatory models. A high value of any selected metric means that explanation is consistent with the underlying predictions.

U2: Completeness

Explanations should generalize well beyond a certain point. Completeness measures how well the explanations cover the underlying predictive models. It is generally checked by measuring the correctness of explanations across multiple data points.

U3: Contextfullness

Explanations should come with all necessary conditions so that the users understand the limitations correctly. Contextfullness makes a local explanation explicitly local or indicates that it can be treated as global despite being derived for a single prediction. A specific case of this property, called *representativeness*, measures the number of instances a single explanation covers in a data set. Another aspect of contextfullness is the degree of importance for each factor contributing to an explanation. For example, if three causes support an explanation, how important are they individually.

U4: Interactiveness

Given an audience's wide range of experience and background knowledge, one explanation cannot satisfy a wide range of expectations. Therefore, to improve the overall user experience, the explanation process should be controllable. For example, it should be reversible (in case the user inputs a wrong answer), respect the user's preferences and feedback, be social (bidirectional communication is preferred to one-way information offloading), allow adjustment to the granularity of an explanation, and be interactive. This means that, whenever possible, the users should be able to customize the explanation that they get to suit their needs. For example, if the system

explains its decisions with counterfactual statements and a foil used in such a statement does not contain information that the users are interested in, they should be able to request an explanation that contains this foil (if one exists).

U5: Actionability

Actionability is a very important concept in the properties of the explanations. All explanations should be prepared in such a way that users can understand the reasons behind the decisions. Also, the output should be so that some desirable actions can be taken based on the explanations. For example, in terms of counterfactual explanations for a banking loan problem, the number of active loans makes more sense than the customer's age.

U6: Novelty

Providing users with a mundane or expected explanation should be avoided. Explanations should contain surprising or abnormal characteristics that have a low probability of happening (e.g., a rare feature value) to point the user's attention in an interesting. However, this objective requires balancing the trade-off between coherence with the listener's mental model, novelty, and overall plausibility. For example, consider an ML system where explanations help to better understand a given phenomenon. In this scenario, providing the users with explanations that highlight relations that they are already aware of should be avoided

U7: Complexity

The complexity of explanations should be tuned to the users who are using the explanations. Suppose the system does not allow for explanation complexity to be adjusted by the user. In that case, it should be as simple as possible by default (unless the listener explicitly asks for a more complex one). For example, an automated medical diagnosis explanation should use observable symptoms rather than the underlying biological processes responsible for the condition.

U8: Personalization

Adjusting an explanation to users requires the explainability technique to model their background knowledge and mental model. This is particularly important when adjusting the complexity of an explanation, its novelty, and its coherence. An explanation can be personalized online via interaction or offline by incorporating the necessary information into the model (e.g., parameterization) or data. Personalizing an explanation is related to another rule of cooperative communication: the maxim of relation. According to this rule, communication should only relay relevant and necessary information at any given point in time. Therefore, an explainability system must “know” what the user knows and expects to determine the content of the explanation.

Safety Requirements

Explainability methods reveal partial information about the data set used to train predictive models, the model’s internal mechanics or parameters, and their prediction boundaries (see safety requirements S1–S3 in the following sections). Therefore, we should consider the effect of explainability on robustness, security, and privacy aspects of predictive systems, which they are built on top of and the robustness of explanations themselves.

S1: Information Leakage

Every explainability approach should be accompanied by a critical evaluation of its privacy and security implications and a discussion about mitigating these factors. It is important to consider how much information an explanation reveals about the underlying model and its training data. For example, consider a counterfactual explanation applied to a logical machine learning model; given that this model family applies precise thresholds to data features, this type of explanation is likely to leak them. On the other hand, explanations for a k-nearest neighbors’ model can reveal training data points, and for a support vector machine, these could be data points on the support vectors.

S2: Explanation Misuse

With information leakage in mind, one can ask how many explanations and different data points it takes to gather enough insight to steal or game the underlying predictive model. This can be a major concern, especially if the predictive model is a trade secret. Furthermore, adversaries can use explanations to game a model; consider a case where a malicious user could find a bug in the model by inspecting its explanations, hence is now able to take advantage of it. This observation indicates a close relation between explanations and adversarial attacks.

S3: Explanation Invariance

Given a phenomenon modeled by a predictive system, the data we gather is a way to quantify its observed effects. The objective of a predictive system should be to elicit insights about the underlying phenomenon, and the explanations are a medium to foster their understanding in a human-comprehensible context. Ideally, explanations should be based on a property of the underlying phenomenon rather than an artifact dependent on a predictive model. An explainability approach should provide the same explanation given the same inputs (model and/or data point). This could be measured by investigating the variance of an explanation over multiple executions of an explainability algorithm. Ideally, explanations produced by one method should be comparable to those produced using another explainability technique (given fixed training data).

Validation Requirements

Finally, explainable systems should be validated with *user studies* or *synthetic experiments* in a setting similar to the intended deployment scenario. A well-designed user study could also provide a clear answer to some of the (qualitative) explanation properties listed in the previous sections. For example, it can evaluate the effectiveness of an explanation for a particular audience, assess the background knowledge necessary to benefit from an explanation, or check the level of technical skills required to use it, because not everyone will be comfortable with a particular explanatory medium. Standardized user studies are not a silver bullet. However, a protocol (such as randomized controlled trials in medical sciences) should be in place, given that they are the most acceptable approach to validate the explainability powers of a new method.

CHAPTER 6 EXPLAINABILITY FACTS: A FRAMEWORK FOR SYSTEMATIC ASSESSMENT OF EXPLAINABLE APPROACHES

A different set of, mostly synthetic, validation approaches was proposed using simulated data with known characteristics to validate the correctness of explanations and testing stability and consistency of explanations

- **Simulability** measures how well a human can re-create or repeat (simulate) the computational process based on the provided explanations of a system.
- **Algorithmic transparency** measures the extent to which a human can fully understand a predictive algorithm: its training procedure, the provenance of its parameters, and the process governing its predictions.
- **Decomposability** quantifies the ability of the listener to comprehend individual parts (and their functionality) of a predictive model: understanding features of the input, parameters of the model (e.g., a monotonic relationship of one of the features), and the model's output.

Multiple questions arise when developing an explainability approach for a predictive system and evaluating it based on the list of properties. Are they all equally important? Are they compatible with each other, or are some of them at odds? In practice, many of them cannot be achieved at the same time, and their importance often depends on the application area. Moreover, while selected classes of explainability methods (e.g., counterfactuals) may be flexible enough to comply with most of the properties in theory, some of them can be lost in implementation due to algorithmic choices.

While functional and operational requirements are properties of a specific explanation methodology and its implementation, the usability desiderata are general properties. Any approach should aim to satisfy all of them. For example, making an explainability technique model-agnostic forces it to be a post hoc (or a mimic) approach and prevents it from taking advantage of specifics of particular model implementation. Furthermore, such approaches create an extra layer of complexity on top of the predictive model (as opposed to ante hoc techniques) that can be detrimental to the fidelity of the explanation: a trade-off between completeness and soundness that is common to model-agnostic approaches. Simpler explanations—that is, have fewer causes and are more general and coherent—are usually more appealing to humans. However, depending on the application and the target audience, this may not always be desirable.

CHAPTER 6 EXPLAINABILITY FACTS: A FRAMEWORK FOR SYSTEMATIC ASSESSMENT OF EXPLAINABLE APPROACHES

Some of these properties can be employed to achieve more than one goal. For example, completeness can be partially achieved by having contextual explanations. If an explainability system is not inherently interactive, this requirement can be achieved by deploying it within an interactive platform such as a dialogue system for explanations delivered in natural language or an interactive web page for visualizations. Actionability is usually data set-specific and can be achieved by manually annotating actionable features and ordering the time-sensitive ones.

Summary

This ends the chapter on the assessment of explainability methods based on multiple criteria. This information is useful when you learn more about different methods. Starting in Chapter 8, you can map these properties and assessment criteria to the different methods. It is a great exercise to see which methods fulfill some or all the properties and evaluate the different methods based on their use cases.

CHAPTER 7

Interpretable ML and Explainable ML Differences

We have studied so far about interpretability and its properties. Before we go into the technical section of model interpretability, it is beneficial to understand the basic stuff in the right manner. Many terms are used loosely in different articles. Some common methods of interpretability are SHAP and LIME. Some people, however, loosely mention them under interpretable ML. This chapter highlights the differences between interpretable ML and explainable ML.

Interpretable ML and Explainable ML Basics

Let's start with the basics. Any model is said to be interpretable if humans can understand it on their own. The model summary statistics and parameters are sufficient to explain why the model made a certain prediction or decision.

Let's take the example of a decision tree (see Figure 7-1). The objective of this problem was to predict whether someone would pay a car loan or not. People who would default are tagged as Yes, and the remaining are tagged as No.

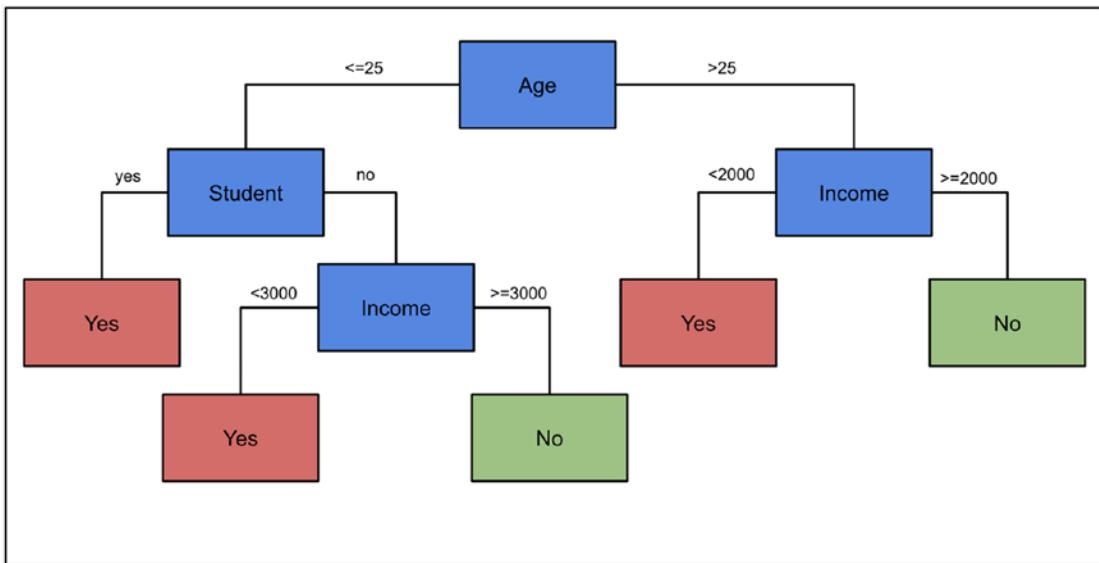


Figure 7-1. A simple decision tree model showing how interpretable models work

Now let's say that a 29-year-old woman with a monthly income of \$3000 applies for the loan. Based on this model, she was granted a loan through an automated underwriting system. We can now examine the nodes of the tree to understand why the loan was granted.

The first split happens at the variable age. Since the candidate is 29 years old, hence we go down the right-hand side node. The next split happens at income. Since she has an income of \$3000, we again go down the right side and arrive at the No leaf node.

Hence, according to the model, the student will not default on a loan.

Analyzing the Decision Tree

Using a simple linear regression model with the following equation, let's predict the maximum loan size by building a model based on a person's age and income. Y is the loan size for an individual.

$$Y = 100 * \text{age} + 10 * \text{income} + 200$$

From the equation, you can see that for every year of a person's age, the predicted maximum loan increases by \$100. Also, loan size increases as income increases.

For this example (29 years old, \$3,000 income), the maximum loan size is predicted to be \$33,100.

We can look at the model's parameters and understand why exactly a certain prediction was made. This is possible because these models are fairly simple. The equation has three parameters. But when the models become more complicated, it becomes difficult to understand them similarly.

A random forest is a complex model. It consists of multiple individual trees, and the final prediction is made by considering the output of each tree. To understand the decision made by a random forest, we must simultaneously understand the decision of each tree within the overall model. Even with a small number of trees, this is a complicated exercise.

Now imagine the scale of complexity when we talk about a neural network model. For example, a convolutional neural network AlexNet which is typically used for image recognition has 62,378,344 parameters. A simple regression model, however, has only three parameters. It is simply not possible for a human to understand a model of such scale and complexity. Certain techniques such as feature importance scores that determine how well each feature can predict the target variable can be useful to understand the model behavior. The higher the score, the more is the influence of the features on the model's prediction.

Some techniques decompose the individual predictions. So unlike feature importance which works mostly at a global level. On the whole data set, these techniques explain how a particular feature affects the behavior of a single instance. LIME and SHAP are such methods. Additionally, there are methods like DeepLift, which produce explanations for neural network models.

You should be very careful while using these techniques and should always focus on building domain knowledge also. To clear any confusion, you can use multiple techniques separately or in combinations.

This section discussed models as either being interpretable or being explainable using certain techniques. However, such a clear separation between methods may or may not exist.

Digging Deeper

The interpretability of models can be laid out on a spectrum, with some techniques being more interpretable than others. Interpretability is the degree to which a model can be understood in human terms. Hence, one model is more interpretable than another if it is easier for a human to understand how it makes predictions than the other model.

Figure 7-2 shows the interpretability spectrum. A convolutional neural network is less interpretable than a random forest, which is less interpretable than a decision tree.

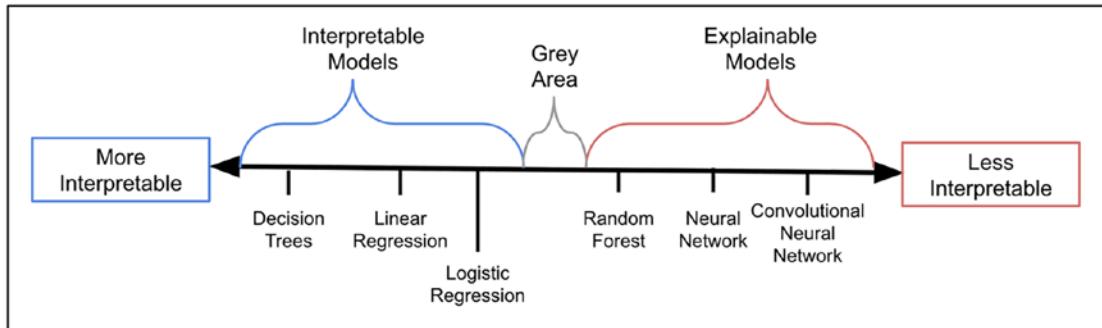


Figure 7-2. The spectrum of interpretability shows the difference between interpretable mL and explainable ML

Interpretable models lie to the left of the spectrum, whereas the explainable models lie to the right of the spectrum. However, there is also a gray area where you would find that people would disagree on the classification.

This linear regression model is interpretable as it was easy to understand by a human. We only had two simple features (i.e., age and income); however, if we included more features, the model would incline more toward the complicated side than the explainable side.

In the machine learning and model building domain, interpretable ML and explainable ML are generally used interchangeably. Although these terms are very close to each other, there is still a fundamental difference that we should keep in mind.

Interpretable ML is studying about or using systems or models which can be explained based on the characteristics of the model itself. When referring to the taxonomy of models, it is a term that comes very close or is similar to intrinsic interpretability. Interpretable ML consists of models which are inherently interpretable. This chapter looks at a linear model and decision trees (see Figure 7-1). These are interpretable models since we can explain why the model took a certain decision using the model itself. It's being able to look at an algorithm and saying, *yep, I can see what's happening here.*

Explainable ML, however, is the extent to which we can explain the internal mechanics of machine or deep learning models using techniques that may or may not be dependent on the model itself. In simpler terms, one can think of it in a way that explainable ML is a term that can be used for all black model models that need a method

to explain what is going on inside the model. Basically, explainable ML lends a helping hand in understanding why the model took a certain decision.

Let's take the example of a sport to explain the difference. Let's say you are watching a cricket match, and the player hits a ball for a boundary. You take all this data and build a model around it to predict whether a shot results in a boundary or not. As soon as a ball approaches the batsman, the model predicts whether the shot is the boundary. In this case, interpretable ML is when you understand why the model predicted a boundary. You might see that the batsman took a high stance and bent forward in an attacking position, and the ball was pitched in the right zone, and hence the batsman was able to take a good shot. Explainable ML seeks to understand the chemistry behind the shot. It explains all the factors as to why the batsman was able to score the boundary. It demystifies complex interactions among the ball pitch and bat quality, and the player's mood, historical performance, or favorite hitting zone.

Understanding *how* something happens is interpretable ML, whereas understanding *why* something happens is explainable ML. Explainable ML finds its use in scenarios of black-box models.

After learning about the differences between explainable ML and interpretable ML, we think it is good for you to understand the challenges governing both and why there is a need for both in separate scenarios. Let's dive into the key issues, starting with explainable machine learning.

Key Issues with Explainable ML

Recent work on the explainability of black boxes might contain critical issues that have generally gone unnoticed but that can have a lasting negative impact on the widespread use of machine learning models.

Trade-offs Between Accuracy and Interpretability

More complex models are generally more accurate, meaning sometimes it is necessary to have a complex black-box model for superior predictive performance. This can be avoided by having well-mannered structured data with naturally meaningful features. When we work on structured data problems with meaningful features, there is sometimes no major difference in performance between more complex classifiers like deep neural networks, boosted decision trees, random forests, and much simpler

classifiers like logistic regression and decision lists after the operations of preprocessing. Some people believe that it is unfair to compare an interpretable CART model with a more advanced neural network model and say that interpretability can only exist by compromising accuracy. However, in the current scenario, we see that to practice explainable ML, you need methods built on top of complex models to achieve accuracy.

Beware of the Unfaithful

Sometimes explainable ML methods provide explanations that are not faithful to what the original model computes. Explanations are not always truthful to how the original model predicts or behaves. An inaccurate explanation model is very dangerous because users will find it hard to trust such a model. The terminology “explanation” sometimes in itself is very misleading because explanation models do not always attempt to mimic the calculations made by the original model. There can be instances where an explanation model built on top of a complex model can generate the same behavior of predictions using a completely different set of features. Hence it is not always the case where an explanation model serves its purpose.

Not Enough Detail

Explanations sometimes do not provide enough details about what is happening inside the black-box model.

Even if the explanation model is correct, it is sometimes possible that the explanations do not provide a full picture of the actual complex model. Recently, data scientists tend to only care about the correct label. Hence, many explainable ML methods focus majorly on the insights for the correct label or positive label. It should be noted that there might be a lot of information hidden in the negative label also, and not always the negative label is the opposite of positive label in terms of explanations. A single variable can tell very different stories about positive and negative labels separately. Thus, one of the disadvantages of explainable ML is that it is purely used to explain the positive label.

Key Issues with Interpretable ML

There are many cases where black boxes with explanations are preferred over interpretable models, even for high-stakes decisions. Interpretable models might sometimes have computational problems or problems with the training of researchers and the availability of code. The following sections highlight some of the major issues with interpretable ML.

Profits vs. Losses

Companies can make profits from the trademarks of the prediction generated by their black-box models, but interpretable ML can result in losses. Interpretable ML can result in losses for companies who try to earn money through good-performing in-house-built black-box algorithms. Interpretable ML does not require explainable methods. It is easy to understand the mechanism behind the model. Thus, it destroys the use case for using complex high accuracy black box models.

A recidivism tool¹ for risk prediction is widely used in the US judiciary system for checking or predicting which convicts can be arrested again after their release. The model's output is simple in terms of if-then-else rules, which bases age and the number of past crimes to predict the likelihood of a person committing another crime that leads to jail. A simple interpretable ML model might be as accurate as this. However, the company behind this model has made this a proprietary software sold to the government. This model is equally accurate for recidivism prediction as to the simple three rule interpretable machine learning model involving only age and number of past crimes. However, it was sold as proprietary software to the judicial system.

In medicine, there is a trend toward blind acceptance of black-box models, which opens the door for companies to sell more models to hospitals.

The examples show that there is a problem with the business model for machine learning. The companies that profit from black-box models for high-stakes decisions are not entirely responsible for the quality of individual predictions or explanations of those predictions because no one knows what is happening inside the model and how the results are generated. The best we can do is apply an explainable method and deduce the mechanism. For this reason, interpretable ML is not very popular and is not

¹<https://www.propublica.org/article/how-we-analyzed-the-compas-recidivism-algorithm>

encouraged. The argument for favoring the black box instead of interpretable ML is that the black box prevents reverse engineering.

Efforts to Construct

Interpretable models entail significant effort to construct in terms of both computation and domain expertise.

For constructing interpretable models that can give results as good as black-box models, you need to have a high degree of domain expertise and computation time. As soon as we increase the number of features inside a traditional interpretable ML model, the computation time increases drastically. Also, the model builder should have constant contact with the model reviewer to check if the results are making sense and are aligned with the business objectives; otherwise, the output of the interpretable model is as good as a black-box model. This results in extensive building time for a good model. Interpretable models usually have a set of application-specific constraints. Solving constraint problems becomes difficult in practice.

Hidden Patterns

Many researchers and scientists find it challenging to construct interpretable models as they find that the black box can uncover various hidden patterns in the data. A transparent model may sometimes be unable to uncover such a pattern in the data. An interpretable model can only uncover a pattern in the data if researchers spend a considerable amount of time and expertise in creating a state-of-the-art interpretable method. This requires a lot of domain expertise to start with.

Now that you have studied the challenges of interpretable ML vs. explainable ML, it is beneficial to dive into another aspect of explainable ML. The next section discusses the differences between *modeling to explain* vs. *modeling to predict*.

Explanatory and Predictive Modeling

Explanatory modeling, as we all must be familiar with by now, is done to uncover hidden patterns in the model. The purpose of explanatory modeling is to understand everything about the model, from propensity scores to causal relationships, feature importance,

counterfactual reasoning, and much more. In summary, explanatory modeling refers to applying statistical models to data for testing causal hypotheses about theoretical constructs.

Predictive modeling is the process of creating models using different algorithms and preprocessing techniques on the input data. The resultant model can predict new observations with high accuracy. The only objective of this type of modeling is to create a prediction that can work as well on unseen data as it worked on the training data.

There is no restriction to the technique used for predictive modeling; for example, you can use a Bayesian or frequentist, parametric or nonparametric, or data mining algorithm or statistical model.

Explaining or Predicting: The Key Differences Between Two Choices

The ability to explain a phenomenon at a conceptual level is very different from generating predictions at a measurable level. This disparity is created by the operationalization of theories into statistical models and measurable data.

To convey this difference properly, consider a theory that X causes Y, and let's describe it using the function F, such that $Y = F(X)$.

F can be considered a model with two dimensions of inputs and outputs, where X is the input construct and Y is the output. F can be any model with the behavior of optimization or prediction.

Because F is usually not sufficiently detailed to lead to a single f, a set of f-models is often considered.

In explanatory modeling, the objective is to match f and F as closely as possible for the statistical inference and prove theoretical hypotheses. The X and Y data are tools for estimating f, which tests the causal hypotheses. The objective of explanatory modeling is to understand the relationship between X and Y and how changes in X govern changes in Y. Another objective is to understand how the f mechanisms work while taking input of X and producing an output of Y.

In contrast, in predictive modeling, the X, Y, and f entities are combined to create good predictions of new Y values. Even if the underlying causal relationship is $Y = F(X)$, a function other than $f(X)$ and data other than X might be preferable for prediction.

Four primary aspects can explain the differences.

- **causation–association:** In explanatory modeling, f represents an underlying causal function, and X is assumed to cause Y . In predictive modeling, f captures the association between X and Y .
- **theory–data:** In explanatory modeling, f is carefully constructed based on F in a fashion that supports interpreting the estimated relationship between X and Y and testing the causal hypotheses. In predictive modeling, f is often constructed from the data. Direct interpretability in the relationship between X and Y is not required, although sometimes transparency is desirable.
- **retrospective–prospective:** Explanatory modeling is retrospective (i.e., the model has some general insights about the data). Predictive modeling, however, is forward-looking (i.e., the model function predicts new observations or future observations).
- **bias–variance:** The expected prediction error for a new observation with value x , using a quadratic loss function, is given as follows.

$$\begin{aligned} \text{EPE} &= E\{Y - \hat{f}(x)\}^2 \\ &= E\{Y - f(x)\}^2 + \{E(\hat{f}(x)) - f(x)\}^2 \\ &\quad + E\{\hat{f}(x) - E(\hat{f}(x))\}^2 \\ &= \text{Var}(Y) + \text{Bias}^2 + \text{Var}(\hat{f}(x)). \end{aligned}$$

Bias is the result of incorrectly specifying the statistical model f . In explanatory modeling, the focus is on minimizing bias to obtain the most accurate representation of the underlying theory. In contrast, predictive modeling seeks to minimize the combination of bias and estimation variance, occasionally sacrificing theoretical accuracy for improved empirical precision.

To conduct proper modeling, you should decide beforehand whether the use case requires them to build a model to explain or predict. Based on the choice, there are many differences in certain parameters that should not be overlooked. The following describes two of them.

- **Choice of variables:** In explanatory modeling, the variable choice is based on the role of the construct in the theoretical causal structure

and on the operationalization itself. A certain terminology related to different roles exists in various fields; for example, in social sciences, they are the antecedent, consequent, mediator, and moderator variables. In pharmacology and medical sciences, they are treatment and control variables, and in the field of epidemiology, they are exposure and confounding variables. In predictive modeling, there is no need to dive into the exact role of each variable in terms of underlying causal structure. The criteria for choosing predictors is the quality of the association between the predictor variables and the response variable. In addition, data quality and data availability play a major role.

- **Choice of methods:** Explanatory modeling requires popular statistical models that are self-explanatory. In scenarios where interpretable models are not possible, you need to carefully choose methods from explainable ML to define the model's inner mechanisms. In predictive modeling, the top priority is generating accurate predictions. Some black box models are very good at generating high-quality predictions. In predictive modeling scenarios, transparency might not be that important. It is good to try a certain number of methods in predictive modeling and then choose the method that gives the best accuracy.

Validation, Model Evaluation, and Model Selection

Choosing the final model among a set of models, validating it, and evaluating its performance, differ markedly in explanatory and predictive modeling. Although the process is iterative, we separate it into three components for ease of exposition.

Validation

In explanatory modeling, validation methods that are generally used are model specification tests. Such tests construct validation techniques such as reliability and validity measures of survey questions and factor analysis. In predictive modeling, one of the major concerns while building a good model is a generalization of the data,

leading to overfitting. Hence validation consists of steps where we compare the model's performance across training and holdouts data sets to be certain that the model is not generalizing on a specific data set.

Model Selection

In explanatory modeling, the models are compared in terms of explanatory power, hence the popularity of nested models, which are easily compared. Stepwise-type methods, which use overall F statistics to include and/or exclude variables, might appear suitable for achieving high explanatory power. However, optimizing explanatory power in this fashion conceptually contradicts the validation step, where variable inclusion/exclusion and the statistical model structure are carefully designed to represent the theoretical model. In contrast to explanatory power, statistical significance plays a minor or no role in assessing predictive performance. It is sometimes the case that removing inputs with small coefficients, even if they are statistically significant, improves prediction accuracy. Stepwise-type algorithms are useful in predictive modeling as long as the selection criteria rely on predictive power rather than explanatory power.

Model Use and Reporting Explanatory Models

In the context of scientific research derives "statistical conclusions" using inference, which in turn are translated into scientific conclusions regarding F, X, Y, and the causal hypotheses. With a focus on theory, causality, bias, and retrospective analysis, explanatory studies aim to test or compare existing causal theories. Accordingly, the statistical section of explanatory scientific papers is dominated by statistical inference. In predictive modeling, the model function generates predictions for new data. We note that generating predictions from the model can range in the level of difficulty, depending on the complexity of the model and the type of prediction generated. For example, generating a complete predictive distribution is easier using a Bayesian approach than the predictive likelihood approach.

We must note the differences between interpretable ML and explainable ML. At many places in print and discussion forums, we might find that these terms are used interchangeably. Since this field is of model interpretability or model explainability is an evolving field, there is a strict difference available. Interpretable ML generally refers to the methods or algorithms that are transparent and can be easily understood.

Explainable ML refers to the methods or algorithms applied on top of black-box algorithms to understand the reasons behind the predictions made by the black-box model. Explanatory modeling is done to understand the mechanisms behind the complex models. It seeks to find what is happening inside the algorithms, and how the predictions are being made. It is done when we want to understand the complex relationships and derive actions and insights from these relationships between features and predictors of the model. Predictive modeling, however, is done where we want to predict on unseen data. In such cases, we are not interested in defining the relationships between features or components of the models.

With this, we come to the end of the second section of this book. This chapter focused on the properties of explanation methods and how humans understand the methods and the explanations. We also understood the key differences between some very commonly used terms. The following quickly summarizes the fundamental concepts of this chapter.

Summary

This chapter defined the differences between interpreting and explaining. We feel it is very important for you to understand the difference. We highlighted how interpretable ML, explainable ML, and interpretability have differences in their basic concepts. We discussed the drawbacks of interpretable ML and explainable ML and recommended considerations for choosing a method. Once the differences between explaining and interpreting were clear, we moved on to explain the difference between modeling for prediction vs. modeling for explanations. You will be able to relate this when you work on problems with different requirements.

The next chapter explains the various frameworks to which each method belongs. You will be able to relate different frameworks to different properties and categorizations that we have studied in all previous chapters.

CHAPTER 8

The Framework of Model Explanations

The last few chapters introduced the machine learning domain, the importance of interpretability, and the taxonomy of interpretability methods. The previous chapters introduced machine learning interpretability and focused on general overview and definitions. This chapter discusses the different frameworks of model explanations. Because there are a variety of methods under each category, please refer to the upcoming chapters for a detailed review of how each method works, along with some working versions of the code.

From this chapter onward, we dive into more complex technical details about interpretability methods. We talk a lot about the math involved in different methods and how the theory of interpretability is structured. We recommend you use a pen and paper to write down some of the key mathematical formulas discussed. It is beneficial to create a learning template, as mentioned in Chapter 1.

Data Sets at a Glance

Machine learning is a very diverse topic and includes all sorts of data sets for applications. We have tabular data, text data, and image data. There are hundreds of models across all these data types and several interpretability methods. Yet, one common thing across all methods is how these methods are built on top of these models. Our focus is mainly on the frameworks of post hoc interpretability techniques as model intrinsic methods rely on the framework of the model itself. We first briefly discuss the different frameworks across different data sets and then explain each framework.

The following is a brief introduction to the different types of data sets.

- **Tabular data:** Tabular data for machine learning is the most common form of a data set and is highly prevalent in all business domains. With the revolution of big data in the industry, companies have set up technical ecosystems to store vast amounts of tabular data. These data sets range from transaction data sets to personal information data sets. Tabular data can run some prevalent supervised machine learning algorithms like black-box classification algorithms and unsupervised algorithms such as clustering. Companies are heavily dependent on tabular data sets for various operations that are not limited to analytics and data.
- **Text data:** Text data has also grown in popularity after the surge in e-commerce companies. One major requirement came out from e-commerce businesses to evaluate customer reviews which were mostly written in free text form. Soon, many algorithms came up to study text data ranging from sentiment analysis to named entity recognition to categorization and so forth.
- **Image data:** Computer vision gained popularity slowly after some cutting-edge research in the machine learning domain. Companies started using these algorithms to analyze video feeds, images, and live videos. A typical image data in the retail domain might consist of pictures of items that need to be categorized or broken down into different components.

With so many algorithms present to analyze these different types of data sets, there is a need to have different types of interpretability algorithms to understand the decision made by different models.

This chapter aims to categorize explanation methods concerning the type of explanation returned and present the most widely adopted quantitative evaluation measures to validate explanations under different aspects and benchmark the explainers adopting these measures. The objective is to guide that maps a black-box model to a set of compatible explanation methods.

Types of Frameworks for Tabular Data

This section looks at the various frameworks for generating explanations for a tabular data set.

- **Rule-based (RB):** A framework that converts the tabular data into a set of rules post a modeling exercise on the data. In a classification problem, each class has a set of rules to define the behavior of that class.
- **Feature importance (FI):** A vector containing a value for each feature. Each value indicates the importance of the feature for the classification.
- **Prototypes (PR):** The user is provided with a series of examples that characterize a class of the black box. A prototype is a data instance that is representative of all the data.
- **Counterfactuals (CF):** A counterfactual explanation of a prediction describes the smallest change to the feature values that changes the prediction to a predefined output.

Feature Importance (FI)

In this era of readily available internet, data is created and captured with ease at an increasing rate. This has led to the creation of data sets with thousands of characteristics associated with each data point. One example is tracking user behavior on a retail website or detailed sensor information from an AI system mounted on a car.

Because of such large data sets, powerful machine learning methods such as deep learning and gradient boosting can achieve a very strong predictive performance in most tasks. However, since these models are black-box models, it is very difficult for humans to understand how the input features worked together to construct the predictions.

We have already spoken about these challenges in previous chapters. This problem exaggerates many-fold in large dimensional data sets. Sometimes, it is often the case that most or all the predictive performance can be achieved with a small subset of the features. This results in unnecessary computational complexity, more time spent training models, and lower performance out-of-sample, as an algorithm might not correctly detect the features driving the signal when exposed to many noisy features.

Unfortunately, we usually cannot directly determine such a subset of features with intuition alone. Thus, the research has focused on developing data-driven methods that identify relevant features, allowing us to discard those that do not provide additional value to the prediction. One of the most used approaches for feature selection is based on assessing the variable importance of a machine learning model, which attempts to quantify the relative importance of each feature for predicting the target variable. The variable importance is calculated by measuring the improvement in the model's performance due to the feature compared to the other features. Once you identify the most important features, you can remove the features of little or no importance. Variable importance is also an approach for model interpretability, whereby you attempt to understand how the model makes predictions by assessing which features are declared important during model training.

However, keep in mind that any method of feature importance is only useful if it is accurate. If some feature importance method overestimates the importance of irrelevant features, it may prevent the removal of unimportant features. On the other hand, if the importance of relevant features is underestimated, we might discard important variables and negatively impact the performance of the final model. Therefore, it is critical to understand how variable importance behaves, particularly whether the variable importance exhibits different characteristics when applied to different machine learning models. There are several frameworks within the variable importance domain of interpretability. The following sections explain each one.

Predictive Power of Feature Subsets

Consider a supervised learning task where the following elements are present.

- f model to predict
- response variable Y
- an input X, where X consists of independent features (X_1, X_2, \dots, X_d).

X denotes random variables, and x denotes values.

Feature importance can be defined as providing or computing how much predictive power a feature provides to the model.

Model f is trained using all the features; however, the performance can also be analyzed with a subset of features. For example,

$$X_S \equiv \{X_i \mid i \in S\} \text{ for different } S \subseteq D, \text{ where } D \equiv \{1, \dots, d\}.$$

Important features are those which, when removed, result in the decline of f 's performance

The restricted model f_S is defined as

$$f_S(xs) = E[f(X)|X_S = xs]$$

So that missing features X_{S^c} are marginalized out using the conditional distribution¹ $p(X_{S^c}|X_S = xs)$.

Two special cases are $S = \emptyset$ and $S = D$, which respectively correspond to the mean prediction $f_\emptyset(x\emptyset) = E[f(X)]$ and the full model prediction $f_D(x) = f(x)$.

Using this convention to accommodate subsets of features, the method can now measure how much f 's performance degrades when removed. Given a loss function, the population risk for f_S is defined as $E[l(f_S(X_S), Y)]$, where the expectation is taken over the data distribution $p(X, Y)$. To define predictive power as a quantity that increases with model accuracy, consider the reduction in risk over the mean prediction and define the function $v_f : P(D) \rightarrow R$ as follows.

$$v_f(S) = \underbrace{E[l(f_\emptyset(X\emptyset), Y)]}_{\text{Mean prediction.}} - \underbrace{E[l(f_S(X_S), Y)]}_{\text{Using feature } X_S}$$

The domain is the power set $P(D)$, the left term is the loss achieved with the mean prediction $E[f(X)]$, and the right term is the loss achieved using the features X_S . The function $v_f(S)$ quantifies the amount of predictive power f derives from the features X_S , and we generally expect that including more features in S makes $v_f(S)$ larger. While v_f provides a model-based notion of predictive power, the method also introduces a notion of universal predictive power. For this, the function is defined as $v : P(D) \rightarrow R$ as the reduction in risk from X_S when using an optimal model.

$$v(S) = \underbrace{\min_{\hat{y}} \mathbb{E}[\ell(\hat{y}, Y)]}_{\text{Optimal constant } \hat{y}} - \underbrace{\min_g \mathbb{E}[\ell(g(X_S), Y)]}_{\text{Optimal model using } X_S}.$$

¹<https://www.statisticshowto.com/conditional-distribution/>

The left term is the loss from an optimal constant prediction \hat{y} , and the right term is the loss for an optimal model g from the class of all functions (e.g., the Bayes classifier). Intuitively, v represents the maximum predictive power that could hypothetically be derived from X_S . Since f is typically trained using empirical risk minimization, the model-based predictive power v_f provides an approximation to, and the two coincide in certain cases where f is optimal.

Additive Importance Measures

In certain straightforward cases, features contribute predictive power in an additive manner. This means that

$$v(S \cup \{i\}) - v(S) = v(T \cup \{i\}) - v(T)$$

for all subsets S, T such that $i \notin S, T$. In these situations, X_i 's importance can be defined as the predictive power it contributes, or $\emptyset_i = v(\{i\}) - v(\emptyset)$.

However, a feature's contribution is not additive because it depends on which X_S features are already present.

A class of additive importance measures that includes any method whose scores $\emptyset_1, \dots, \emptyset_d$ can be understood as performance gains associated with each feature. This framework lets you unify numerous methods that explicitly or implicitly define feature importance in predictive power. This class of methods is defined as follows.

Additive importance measures are methods that assign importance scores $\emptyset_i \in R$ to features $i = 1, \dots, d$ and for which there exists a constant $\emptyset_0 \in R$ so that the following additive function provides a proxy for the predictive power of feature subsets; that is, $u(S) \approx v(S)$.

$$u(S) = \emptyset_0 + \sum_{i \in S} \emptyset_i$$

For methods in this class, $u(S)$ approximates $v(S)$ up to a constant value \emptyset_0 by summing the values \emptyset_i for each included feature $i \in S$. Each \emptyset_i can be viewed as the performance gain associated with X_i , which provides a measure of its importance for the prediction task.

Although these definitions focus on methods that approximate the universal predictive power v , they also include methods that approximate model-based predictive power v_f because they implicitly approximate v . The function v exhibits non-additive

behavior for most prediction problems, so the proxy u often cannot perfectly represent each feature's contribution to the predictive power. Although a crude approximation may provide users with some insight, closer approximations give a more accurate sense of each feature's importance.

Removal-based Explanations for Feature Importance

Removal-based methods belong to the feature importance domain of explainability and are based on the removal of feature to quantify the feature's influence. This framework helps model experts better understand model explanation tools, and that offers a strong theoretical foundation upon which a lot of explainability research can be based. The framework integrates classes of methods that were previously considered separate, including local and global approaches and feature attribution and feature selection methods. Figure 8-1 is a simple illustration describing removal-based explanations (see <https://arxiv.org/pdf/2011.14878.pdf>).

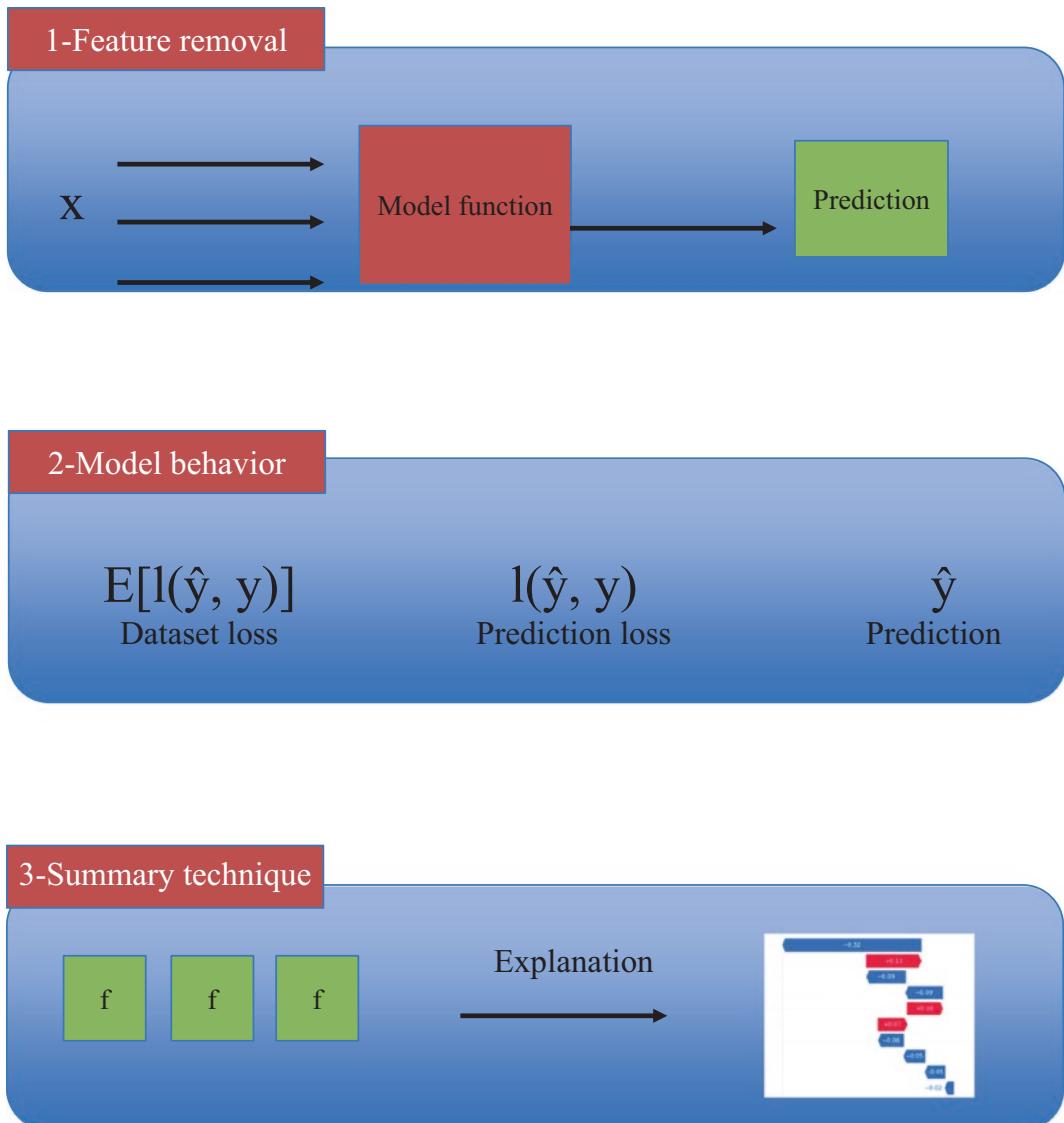


Figure 8-1. A unified framework for removal-based explanations. Each method is determined by three choices: how it removes features, what model behavior it analyzes, and how it summarizes feature influence

Consider an ML model that can predict response variable Y using input X , which can be represented as $X = (X_1, X_2, \dots, X_d)$, where each X_i represents individual features such as age or income. We use uppercase symbols (e.g., X) to denote random variables and lowercase ones (e.g., x) to denote their values. We also use X to denote the domain of the full feature vector X and X_i to denote the domain of each feature X_i .

Finally, $xS \equiv \{x_i : i \in S\}$ denotes a subset of features for $S \subseteq D \equiv \{1, 2, \dots, d\}$, and $S^- \equiv D \setminus S$ represents a set's complement.

Let's leave aside the inherent interpretable models. All other approaches generate explanations by considering the perturbation of the input and using the outcomes to explain each feature's influence on the model. Removal-based explanations are a unified framework developed by connecting methods that define each feature's influence through the impact of removing it from a model.

The following are some of the most common methods that rely on this type of framework or mechanism.

- Breiman, 2001; Ribeiro et al., 2016 – Random forest feature importance and permutation importance
- Fong and Vedaldi, 2017
- Lundberg and Lee, 2017 – SHAP

All these methods work on the principle of feature removal; however, they also have a diverse set of approaches beyond that. For example, LIME fits a linear model to an interpretable representation of the input. The Shapley effect examines how much of the model's variance is explained by each feature. The differences between these two methods are easy to systematize because each method removes different sets of features. This method, however, shows how even these two methods can be explained using only three steps. Removal-based explanations are model explanations that quantify the impact of removing sets of features from the model. Three choices determine these methods.

- **Feature removal:** How the method removes features from the model (e.g., by setting them to default values or by marginalizing over the distribution of values)
- **Model behavior:** What model behavior does the method analyses (e.g., the probability of the true class or the model loss)
- **Summary technique:** How the method summarizes each feature's impact on the model (e.g., by removing a feature individually or by calculating the Shapley values)

These methods help you understand the importance of the internal concept of feature by describing the trade-offs different methods have between them across the three. We read about these differences in upcoming chapters.

Feature Removal

Here, we define the mathematical tools necessary to remove features from ML models and examine how existing explanation methods remove features.

Functions on subsets of features: Most ML models make predictions given a specific set of features $X = (X_1, \dots, X_d)$. Mathematically, these models are functions of form $f : X \rightarrow Y$, and we use F to denote the set of all such possible mappings. The principle behind removal-based explanations is to remove certain features to understand their impact on a model. Since most models require all the features to make predictions, removing a feature is more complicated than simply not giving the model access to it. To remove features from a model or to make predictions given a subset of features, we require a different mathematical object than $f \in F$. Instead of functions with domain X , consider functions with domain $X \times P(D)$, where $P(D)$ denotes the power set of $D \equiv \{1, \dots, d\}$. To ensure invariance to the held out features, these functions must depend only on features specified by the subset $S \in P(D)$, so we formalize subset functions as follows.

A subset function is a mapping of form $F : X \times P(D) \rightarrow Y$ that is invariant to the dimensions that are not in the specified subset. That is, we have $F(x, S) = F(x', S)$ for all (x, x', S) such that $xS = x'S$. We define $F(x_S) \equiv F(x, S)$ for convenience because the held out values x_S are not used by F .

A subset function's invariance property is crucial to ensure that only the specified feature values determine the function's output while guaranteeing that the other feature values do not matter. Another way of viewing subset functions is that they simulate the presence of missing data. While we use F to represent standard prediction functions, we use \mathcal{F} to denote the set of all possible subset functions.

Subset functions are used in this method because they conceptualize how different methods remove features from ML models. Removal-based explanations typically begin with an existing model $f \in F$, and to quantify each feature's influence, they must establish a convention for removing it from the model. A natural approach is to define a subset function $F \in \mathcal{F}$ based on the original model f . To formalize this idea, a model extension is defined as follows. An extension of a model $f \in F$ is a subset function $F \in \mathcal{F}$ that agrees with f in the presence of all features. That is, the model f and its extension F must satisfy $F(x) = f(x) \forall x \in X$.

Explaining Different Model Behaviors

Removal-based explanations demonstrate how a model works, but they can do so by analyzing various model behaviors. We now consider the various choices of target quantities to observe as different features are withheld from the model.

The feature removal principle is flexible enough to explain virtually any function. For example, methods can explain a model's prediction or loss function, a hidden layer in a neural network, or any node in a computation graph. In fact, removal-based explanations need not be restricted to the ML context: any function that accommodates missing inputs can be explained via feature removal by examining either its output or some function of its output as groups of inputs are removed.

Each method's target quantity can be understood as a function of the model output, which is represented by a subset function $F(x_S)$. Many methods explain the model output or a simple function of the output, such as the log-odds ratio. Other methods take a measure of the model's loss for either an individual input or the entire data set. Ultimately, as follows, each method generates explanations based on a set function of the form.

$$u: P(D) \rightarrow R$$

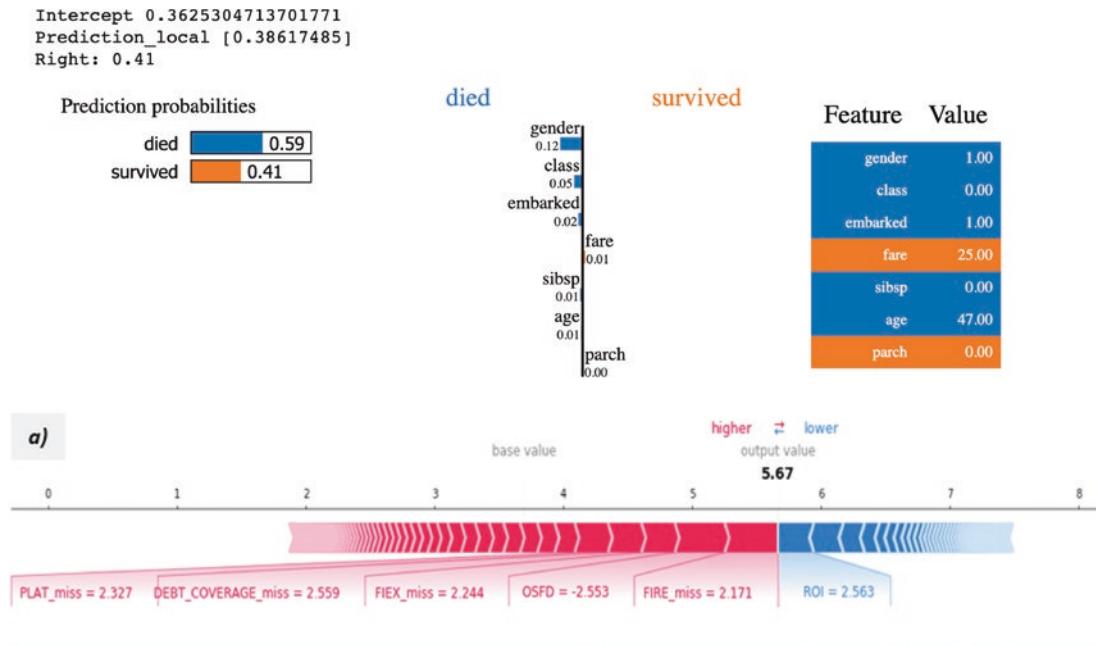
This represents a value associated with each subset of features $S \subseteq D$. This set function represents the model behavior that a method is designed to explain.

Summarizing Feature Influence

The third choice for removal-based explanations is how to summarize each feature's influence on the model. The set functions to represent a model's dependence on different features are complicated mathematical objects that are difficult to communicate fully due to the exponential number of feature subsets and underlying feature interactions. Removal-based explanations confront this challenge by providing users with a concise summary of each feature's influence.

Many methods fall under the feature removal framework and work on the principles of the three points described earlier.

The following are some of the most common methods: IME, QII, SHAP (shown in Figure 8-2), Kernel SHAP, Tree SHAP, Loss SHAP, SAGE, Shapley, Permutation, Conditional, Univariate, L2X, INVASE, LIME (shown in Figure 8-2), PredDiff, CXplain, RISE, and FIDO-CA. Some of these methods are covered in the upcoming chapters.

**Figure 8-2.** LIME and SHAP output sample

Rule-based Explanations

Decision rules are a simpler form of expressing the important features of the model. Apart from giving the important features, decision rule sets also provide the values of these features, which contribute to better prediction. One advantage of decision rules is that they are human-friendly. They explain the model in a very simple textual format that non-technical users and stakeholders easily understand.

A decision rule, r , also called the *logic rule*, has the form $p \rightarrow y$, in which p is a premise, composed of a Boolean condition on feature values, while y is the consequence of the rule. In particular, p is a conjunction of split conditions of the form $x_i \in [v_l(i), v_u(i)]$, where x_i is a feature and $v_l(i)$, $v_u(i)$ are lower and upper bound values in the domain of x_i extended with $\pm\infty$. An instance x satisfies r , or r covers x , if every Boolean condition of p evaluates to true for x . If the instance x to explain satisfies p , the rule $p \rightarrow y$ represents a candidate explanation of the decision $g(x) = y$. Moreover, if the interpretable predictor mimics the behavior of the black box in the neighborhood of x , we further conclude that the rule is a candidate local explanation of $b(x) = g(x) = y$.

In the context of rules, you can also find the counterfactual rules. Counterfactual rules have the same structure of decision rules, with the only difference that the consequence of the rule y is different w.r.t. $b(x) = y$. They are important to explain to the end user what should be changed to obtain a different output.

An example of a rule explanation is $r = \{age < 40, income < 30k, education \leq Bachelor\}, y = deny$. In this case, the record $\{age = 18, income = 15k, education = Highschool\}$ satisfies the rule.

The most common rule-based methods include ANCHOR, LORE, RuleMatrix, TREPAN, SkopeRules, and GLocalX.

$$r = \{Age > 34, HoursPerWeek > 20, Education > 12.5, Occupation \leq 3.5, Relationship \leq 2.5, CapitalGain \leq 20000, CapitalLoss \leq 223\} \rightarrow 1 \quad \{ Prec = 0.79 \%, Rec = 0.15 \%, Cov = 1 \}$$

$$r = \{ Age \leq 19, WorkClass \leq 4.5, HoursPerWeek \leq 30, Education \leq 13.5, Education > 3.5, MaritalStatus > 2.5, Relationship > 2.5, CapitalLoss \leq 1306 \} \rightarrow 0 \quad \{ Prec = 0.99 \%, Rec = 0.19 \%, Cov = 1 \}$$

Figure 8-3. Global explanations of XGB on adult dataframe available online. On the left, a rule for class > 50K, on the right for class < 50K

Prototypes

In a model sense, a prototype is an object that represents a similar set of records.

A prototype can be

- a record from the training data set which is close to the input data x
- a centroid of the cluster to which the input belongs to
- a synthetic record, generating using some ad hoc process

The concept of a prototype is based on the fact that users understand the model's reasoning by looking at records similar to what they have selected.

Prototypes are not covered in detail in the upcoming chapters; hence we cover the basic prototype methods.

MMD-critic is a prototype methodology that works before the model because it analyses the distribution of the data set under the analysis. It works using the *maximum mean discrepancy* (MMD). Prototypes explain the data set's general behavior. It selects the prototypes by measuring the differences between the distribution of instances and instances in the whole data set. The set of instances nearer to the data distribution are called *prototypes*. This method shows only minority data points that differ from the prototype but belong in the same category.

Privacy-preserving explanations are the first method in the domain of explainability, which considers privacy-protected explanations. It is a model agnostic local post hoc method.

The trade-off between privacy and comprehensibility of the explanation is employed by constructing micro aggregation. In this method, a set of clusters are created. Then for each cluster, a shallow decision tree is created to provide an exhaustive explanation while maintaining good readability due to the limited depth of the trees. When a new record x arrives, a representative record and its associated shallow tree are selected.

Prototype selection (PS) is an interpretable model that seeks prototypes that better represent the data under analysis. This method uses a set cover optimization problem with some constraints on the properties of the prototypes. Each record in the input data set is assigned to a representative prototype. After this, the prototypes are employed to learn a nearest neighbor rule classifier.

Counterfactuals

Counterfactuals are a class of model explainability methods that help users generate actionable interpretations of the model. Until now, we have studied different feature importance methods and rule-based methods, which tell at a global or a local level as to which features are important for the model. However, this information does not provide complete clarity and understanding. In real-life scenarios, what catches the users' interests is once they know which variables explain a specific class, then what action can be taken on those variables or features to move the instance from that class to some other class or opposite class in a binary classification problem.

Hence, counterfactuals focus on the differences to obtain the opposite prediction w.r.t. $b(x) = y$.

The general form of a counterfactual explanation should have: $b(x) = y$ was returned because variables of x have values $x_1; x_2; \dots; x_n$. Instead, if x had values $x_1/1; x_1/2; \dots; x_1/n$ and all the other variables have remained constant, $b(x) = -y$ would have been returned, where x is the record x with the suggested changes. A good counterfactual always alters as few variables as possible to find the closest combination of the prediction change.

Counterfactual methods can be divided into the following.

- Exogenous: These methods generate the counterfactuals synthetically.

- Endogenous: In these methods, counterfactuals are drawn from a reference population, resulting in more realistic instances.
- **Instance-based:** These methods use a distance function to detect the decision boundary of the black box. The properties fulfilled by counterfactual explanations are efficiency, robustness, diversity, actionability, and plausibility.

Explanations for Image Data

This section talks about the different frameworks of the explanations that can be utilized for image data. Some of the most common methods are saliency maps (SM), concept attribution (CA), prototypes (PR), and counterfactuals (CF). You learned about counterfactuals and prototypes in the previous sections. The next section focuses on the remaining two methods.

Saliency Maps

An image in which a pixels' brightness represents the saliency of the pixel is called a *saliency map*. SM is the short form for saliency map, which is modeled as a matrix S with dimensions as the size of the image and values as the saliency of that pixel. To visualize a saliency map, divergent color maps are used, ranging from red to blue. A red value means that pixel ij contributes positively to the classification, while a negative means that the same pixel contributes negatively. There are two methods for creating SMs; The first one assigns to every pixel a saliency value. The second one segments the image into different pixel groups and then assigns a saliency value for each group. Figure 8-4 is a visual description of saliency maps.

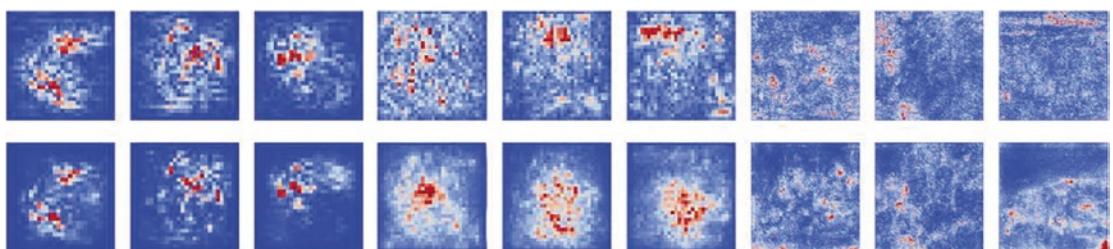


Figure 8-4. Visual Comparison of saliency maps

Concept Attribution

Some models built for image data are designed to work on features like edges and lines in a picture, which are not easily understandable by humans. Feature-based explanations applied to state-of-the-art complex black-box models can yield non-sensible explanations.

Concept-based explainability for images designed the model's explanations based on human-friendly concepts rather than representing the input-based features or model internals.

Concepts like a fifth pixel in the image with a value of 28 do not make sense to humans.

Instead, CA method quantities, for example, how much the stripes concept has contributed to the *zebra* class prediction. Formally, given a set of images belonging to a concept $[x(1); x(2); \dots; x(i)]$ with $x(i) \in C$, CA methods can be thought as a function $f : (b; [x(i)]) \rightarrow e$, which assigns a score e to the concept C based on the predictions and the values of the black-box b on the set $[x(i)]$.

Text Data

Text data does not have a structure. Text data is studied as a different field of machine learning under the name of natural language processing. Examples of usage in text classification are sentiment analysis, topic labeling, and spam detection. Text classification is the process of assigning tags or categories to text according to its content. Using labeled examples as training data, an ML model can learn the different associations between pieces of text and a particular output called *tags*. Tags can be thought of as labels that distinguish different types of text. For sentiment analysis, it is possible to have tags as positive, negative, or neutral. Interpretability techniques are generally applied to understand what words are the most relevant for a specific tag assignment. For text data, sentence highlighting (SH) and attention-based (AB) methods are two widely used methods to understand the workings of the models.

Sentence Highlighting

Like saliency maps with image data, saliency highlighting is saliency maps applied to text and consists of assigning a score to every word based on the importance that that word had in the final prediction. Sentence highlighting is modeled as a vector s , which explains a classification $y = b(x)$ of a black-box b on x . The dimensions of s are the words present in the sentence x , we want to explain, and the s_i value is the saliency value of the word I . The greater the s_i value, the bigger is the importance of that word. A positive value indicates a positive contribution toward y , while a negative one means that the word has contributed negatively. The same is depicted in Figure 8-5 as an example.



Figure 8-5. Example of sentence highlighting

Attention-based Methods

Attention-based sentence highlighting is a mechanism to produce a heatmap explanation similar to the one used for saliency maps. The scores are computed for every word of the sentence by using the attention layer of the black box. The weights α_{ij} of the attention layer are used as a score. The higher the score, the redder highlighting and hence higher is the importance.

An attention matrix looks at the dependencies between words for producing explanations.

Attention matrix relates different positions of a single sequence to compute its internal representation. It is a self-attention method, sometimes called *intra-attention*. The attention of a sentence x composed of N words can be understood as an $N \times N$ matrix, where each row and columns represent a word in the input sentence. The values of the matrix are the attention values of every possible combination of the tokens. This matrix is a representation of values pointing from each word to every other word. The same is depicted in Figure 8-6.

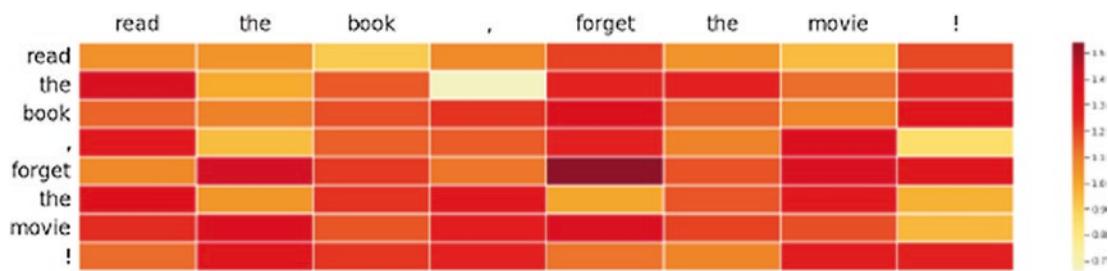


Figure 8-6. Saliency heat-map matrix generated

Here, we come to the end of this chapter. We discussed the different types of frameworks that give birth to various methods across multiple data types. In the next few chapters, we further discuss some methods or frameworks discussed in this chapter across tabular, image, and text data.

Summary

Feature importance is the most common framework for tabular data, particularly for model agnostic black-box explanations. Rule-based explanations are gaining attention since their logic formalization enables a deeper understanding of the AI model's internal decisions, and also they are much easier and simpler to read. Recently, methods explaining counterfactuals are yielding interesting results because they are more actionable for the end user.

The most considerable adopted technique for image data is based on the creation of saliency maps, which translates to the image domain the feature relevance approach for tabular data, highlighting the portions of the relevant images for the AI model outcome. However, other approaches, like concept attribution, prototypes, and counterfactuals, have gained prominence in recent years.

The explanation techniques are still limited in text data, but it is still possible to highlight a few trends. One such method is sentence highlighting. Similar to feature importance for tabular data, it gives weight to the portion of the input that contributed, positively or negatively, to the outcome. Across the different data types, different approaches tend to use similar strategies. This is also evident if we look at the internals of these algorithms.

For example, several methods such as LIME exploit the generation of a synthetic neighborhood around an instance to reconstruct the local distribution of data around the point to investigate. Another frequent strategy is learning a surrogate model from partial training data (sometimes created from the neighborhood generation). This approach tries to bring the benefit of intrinsic methods in the context of black-box explanation.

CHAPTER 9

Feature Importance Methods: Details and Usage Examples

Before diving into various methods and their details, let's look at a sample data set to use across all the code. The next section discusses the details of this data set.

Data Set Name

Online shopper intentions

Data Set Characteristics	Multivariate	Number of Instances:	12330	Area:	Business
Attribute Characteristics	Integer, Real	Number of Attributes:	18	Date Donated	2018-08-31
Associated Tasks	Classification, Clustering	Missing Values?	N/A	Number of Web Hits:	144829

Abstract

The data set consists of 12,330 sessions in which 84.5% (i.e., 10,422) are negative classes. This means these observations/customers did not end with shopping. However, 1,908 were positive classes, which means these observations/customers ended with shopping.

Sources

- C. Okan Sakar Department of Computer Engineering, Faculty of Engineering and Natural Sciences, Bahcesehir University, 34349 Besiktas, Istanbul, Turkey
- Yomi Kastro Inveon Information Technologies Consultancy and Trade, 34335 Istanbul, Turkey

Data Set Information

The data set consists of observations and features belonging to 12,330 sessions. The data set was formed so that each session would belong to a different user after one year to avoid any tendency to a specific campaign, special day, user profile, or period.

Attribute Information

The data set consists of ten numerical and eight categorical attributes.

The response variable is Revenue. The following are the features.

- Administrative
- Administrative Duration
- Informational
- Informational Duration
- Product Related
- Product-Related Duration

These features represent the number of pages visited by the user in that session and the total time spent on each page category. The values of these features are derived from the URL information of the pages visited by the user and updated in real time when a user takes action (e.g., moving from one page to another).

- Bounce Rate
- Exit Rate
- Page Value

These features represent the metrics measured by Google Analytics for each page in the e-commerce site.

The value of the Bounce Rate feature for a web page refers to the percentage of visitors who enter the site from that page and then leave (bounce) without triggering any other requests to the analytics server during that session.

The value of the Exit Rate feature for a specific web page is calculated as for all pageviews to a page, the percentage that was the last in the session.

The Page Value feature represents the average value for a web page that a user visited before completing an e-commerce transaction.

The Special Day feature indicates the closeness of the site visiting time to a specific special day (e.g., Mother's Day, Valentine's Day) in which the sessions are more likely to be finalized with a transaction. The value of this attribute is determined e-commerce dynamics, such as the duration between the order date and delivery date. For example, for Valentine's Day, this value takes a non-zero value between February 2 and February 12. Its value is a zero before and after this date unless it is close to another special day. Its maximum value of 1 is on February 8.

The data set also includes the operating system, browser, region, traffic type, visitor type as returning or new visitor, a Boolean value indicating whether the date of the visit is weekend, and month of the year.

Since you now have insight into the attributes of the data, let's do some EDA on the data to better understand its properties.

Let's first import the required libraries.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_
score, accuracy_score
import seaborn as sns
```

Now let's load the data. We are using a Jupyter notebook to code the EDA and the methods part.

```
df=pd.read_csv(r'~/online_shoppers_intention.csv')
df.head()
```

Next, check for missing values in the data.

```
df.isnull().sum()
```

The following is the output.

Administrative	0
Administrative_Duration	0
Informational	0
Informational_Duration	0
ProductRelated	0
ProductRelated_Duration	0
BounceRates	0
ExitRates	0
PageValues	0
SpecialDay	0
Month	0
OperatingSystems	0
Browser	0
Region	0
TrafficType	0
VisitorType	0
Weekend	0
Revenue	0

It looks like there are no missing values in the data set.

Let's check the distribution of the response variable.

```
df.Revenue.value_counts()
```

The following is the output.

False	10422
True	1908
Name:	Revenue, dtype: int64

The EDA report uses the pandas profiling module. Pandas profiling makes it very easy to get all details consolidated in one place without running multiple lines of code.

Figure 9-1 is an overview of the data from pandas profiling. There are no missing values. Some rows are duplicates in the data. They should be removed.

Number of variables	18	Numeric	14
Number of observations	12330	Categorical	2
Missing cells	0	Boolean	2
Missing cells (%)	0.0%		
Duplicate rows	76		
Duplicate rows (%)	0.6%		
Total size in memory	3.1 MiB		
Average record size in memory	263.2 B		

Figure 9-1. Pandas profiling overview

Figure 9-2 shows that the administrative variable has a mean of 2.3, which means the customers in the data set visited the administrative type of pages on an average two times. (Please refer to the definition of each variable at the beginning of the chapter to better understand the output.)

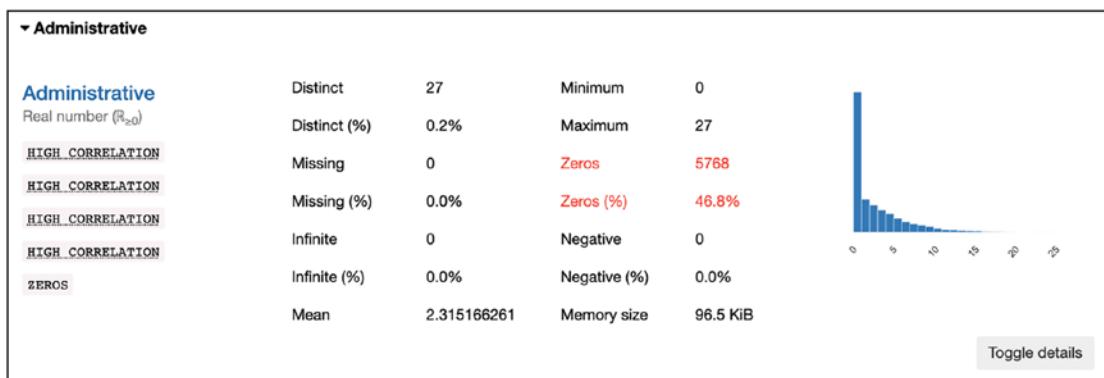


Figure 9-2. Details for Administrative variable

CHAPTER 9 FEATURE IMPORTANCE METHODS: DETAILS AND USAGE EXAMPLES

Figure 9-3 shows that the mean value for the administrative duration for customers is 80 seconds.

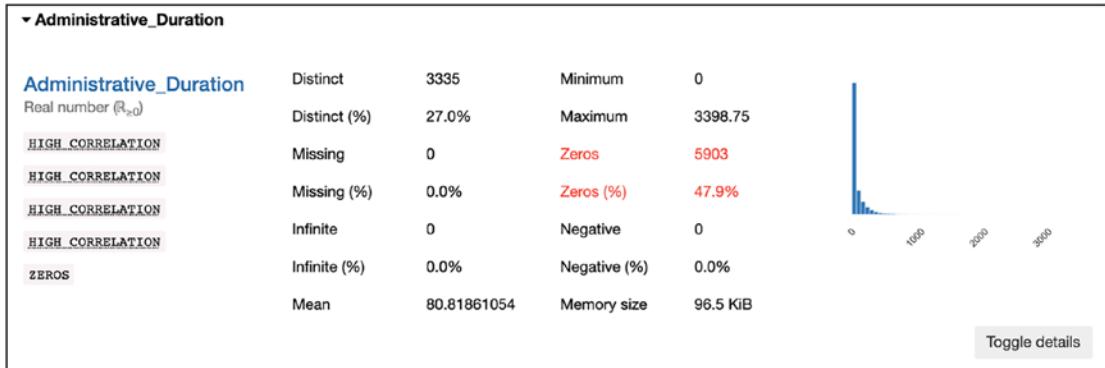


Figure 9-3. Details for Administrative Duration variable

Figure 9-4 shows that this variable has a high number of zeros in the values.

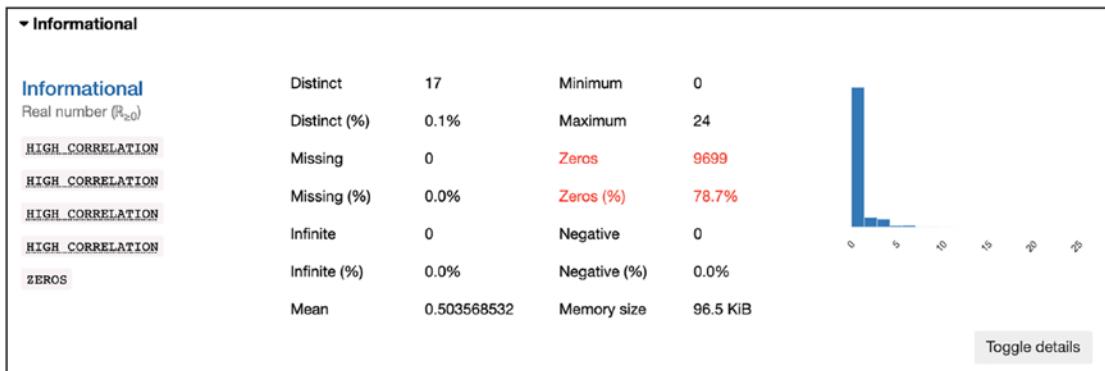


Figure 9-4. Details for Informational variable

Figure 9-5 shows the Information Duration profile. Most of the values are zero.

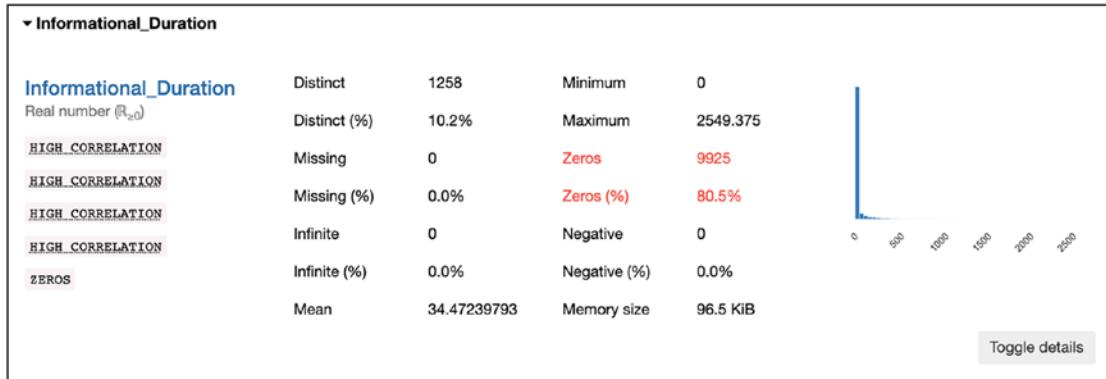


Figure 9-5. Details for Informational_Duration variable

Figure 9-6 shows the overview of the ProductRelated pages visited by the customers given in the data. It shows that, on average, customers visited 38 product-related pages.

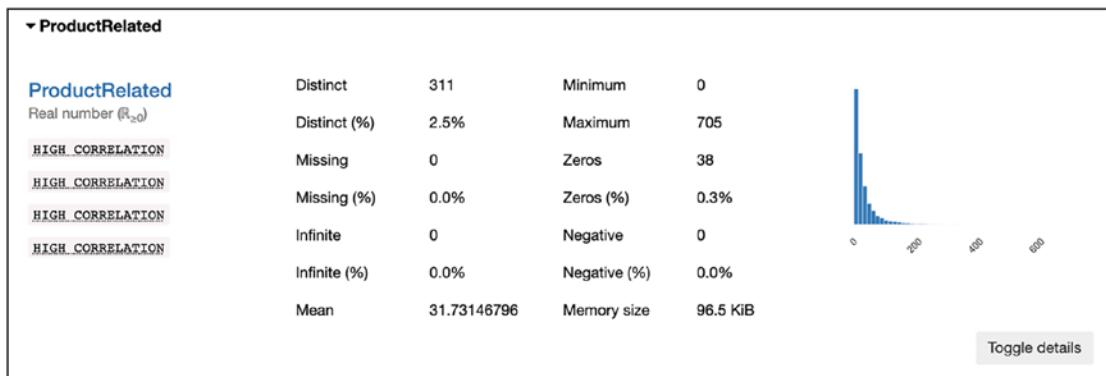


Figure 9-6. Details for ProductRelated variable

CHAPTER 9 FEATURE IMPORTANCE METHODS: DETAILS AND USAGE EXAMPLES

The product-related duration is related to the ProductRelated pages variable. The mean duration is 1,195 seconds, as shown in Figure 9-7.

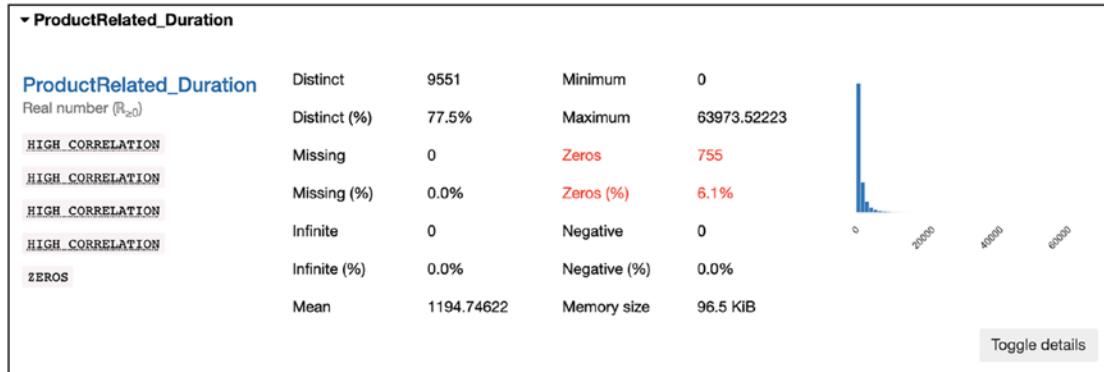


Figure 9-7. Details for ProductRelated_Duration

Figure 9-8 shows the summary of bounce rate variable. A lot of values are zero across all customers given in the data.

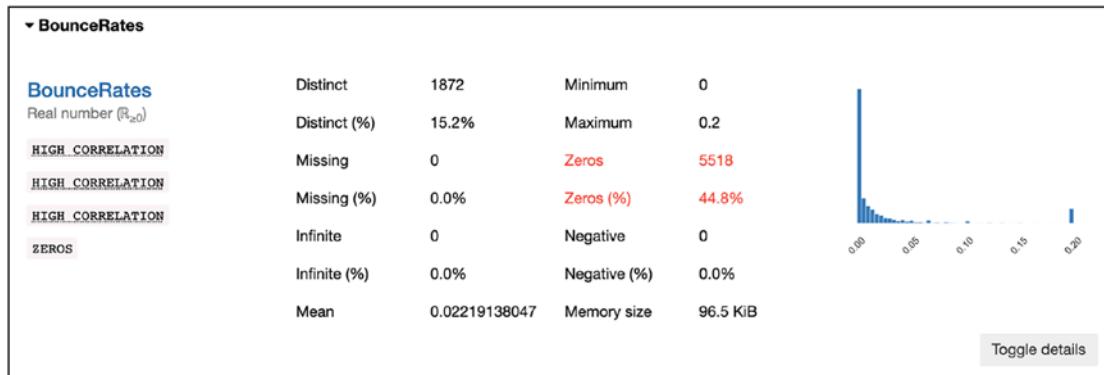


Figure 9-8. Details for BounceRates

Figure 9-9 overviews the ExitRates variable. Unlike other variables, there are not a lot of zeros, and the variable holds a lot of information. This should be a good variable to use in the model.

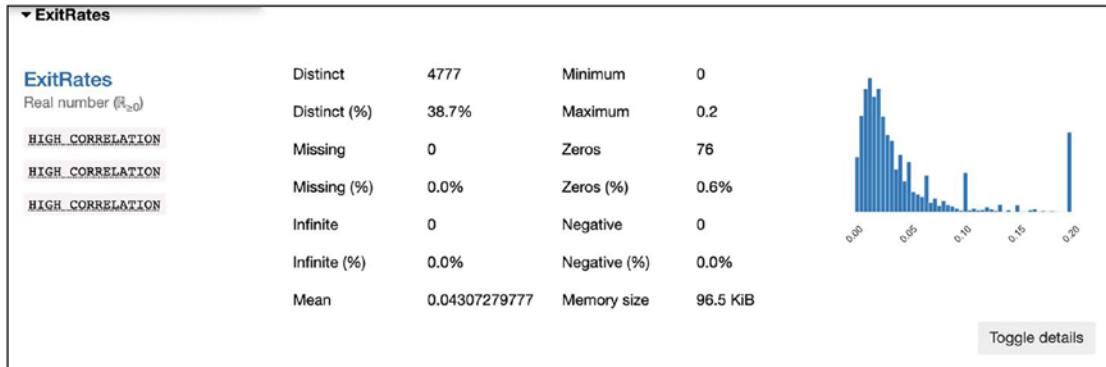


Figure 9-9. Details for ExitRates

Figure 9-10 shows an overview of the PageValues variable. This variable has a mean value of 5, which means, on average, a given customer visits five pages. It has a lot of zero values.

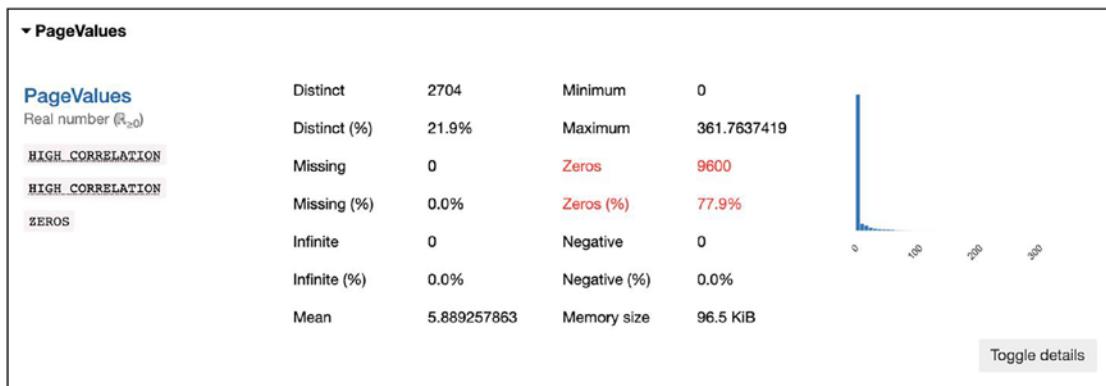


Figure 9-10. Details for PageValues

CHAPTER 9 FEATURE IMPORTANCE METHODS: DETAILS AND USAGE EXAMPLES

Figure 9-11 shows the distribution and overview of SpecialDay variable. Most of the values are zero for this variable too.

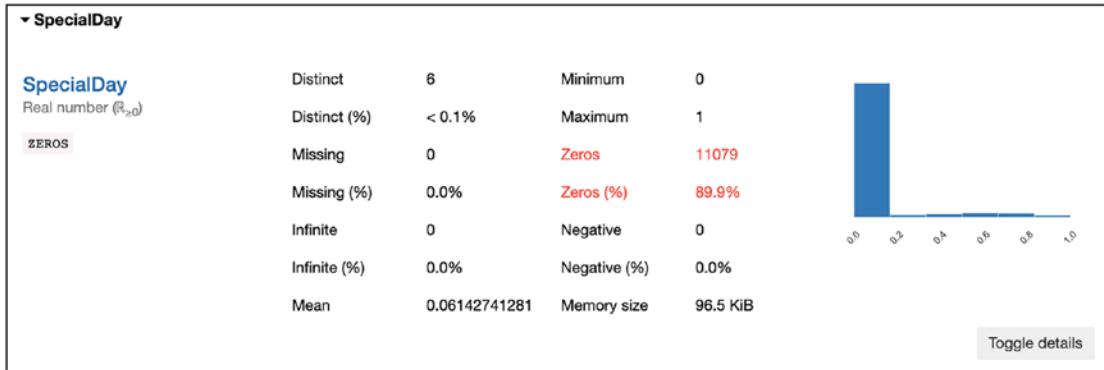


Figure 9-11. Details for SpecialDay

Figure 9-12 shows the distribution and overview of the Month variable. This is a categorical variable and hence the distribution of customers within each category inside this variable.

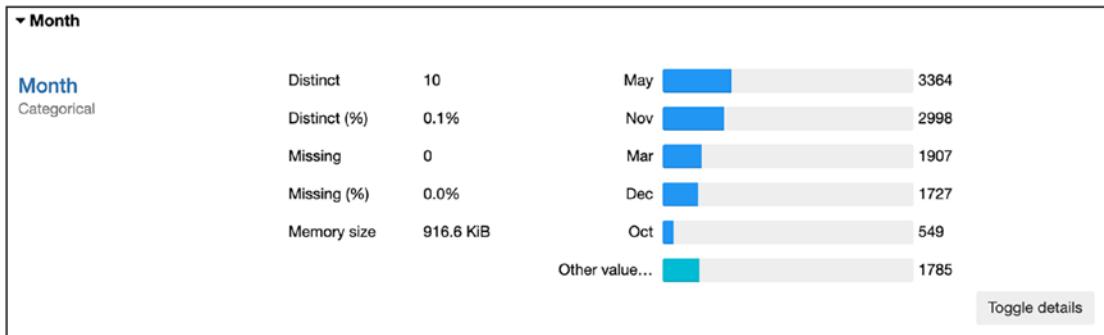


Figure 9-12. Details for Month

Figures 9-13 through 9-18 show the distribution and overview of the remaining independent variables.

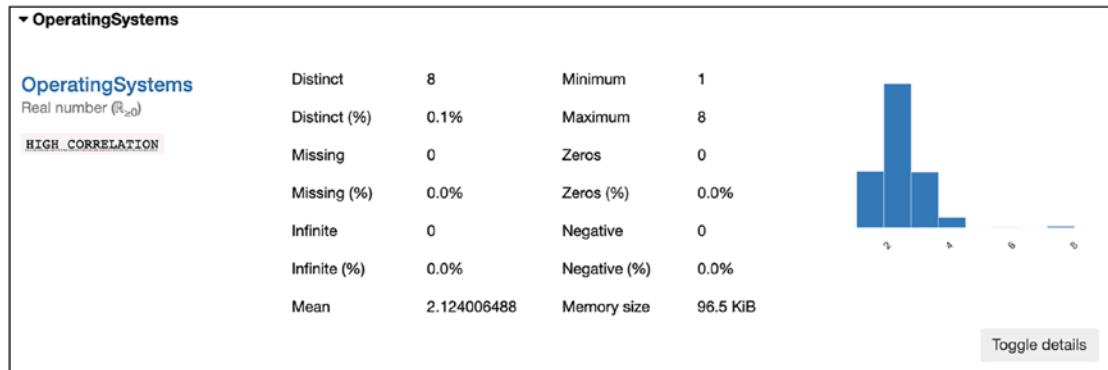


Figure 9-13. Details for OperatingSystems

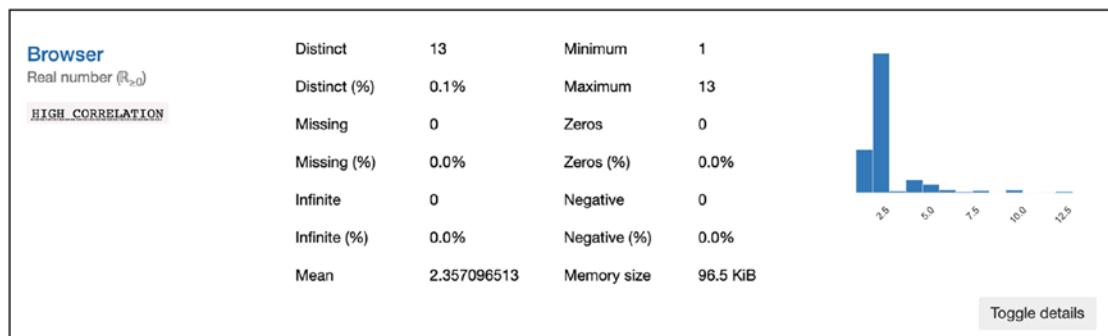


Figure 9-14. Details for Browser

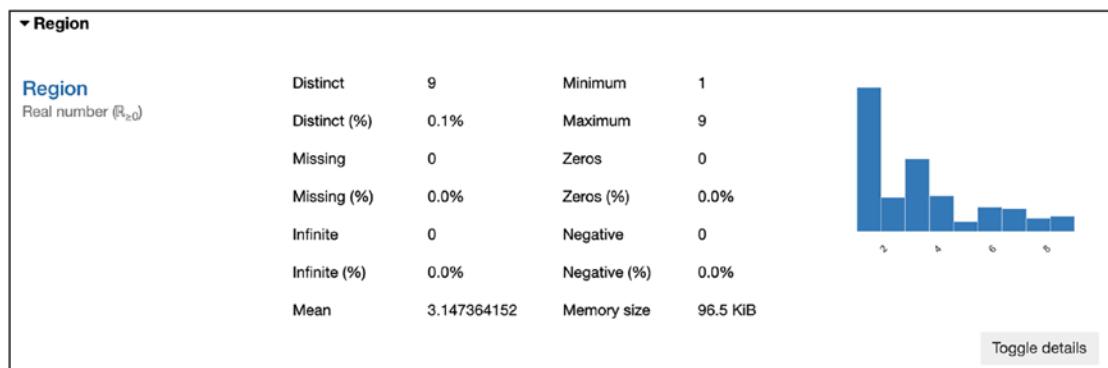


Figure 9-15. Details for Region

CHAPTER 9 FEATURE IMPORTANCE METHODS: DETAILS AND USAGE EXAMPLES

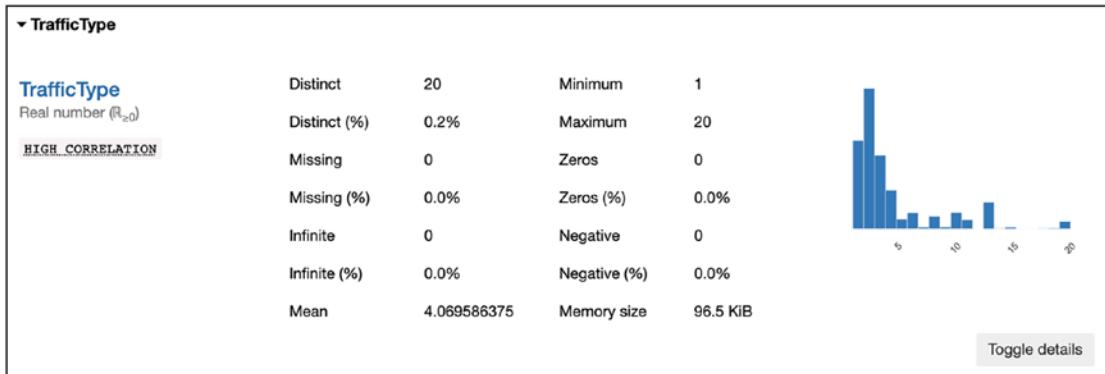


Figure 9-16. Details for TrafficType

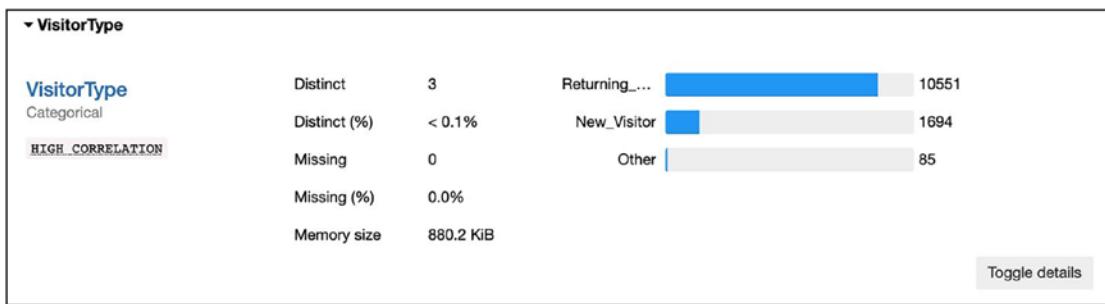


Figure 9-17. Details for VisitorType

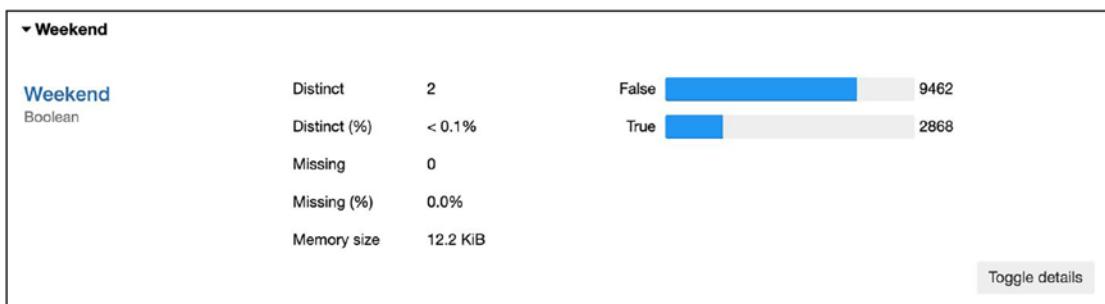


Figure 9-18. Details for Weekend

Figure 9-19 shows the overview and distribution of variable Revenue, which is the target variable.

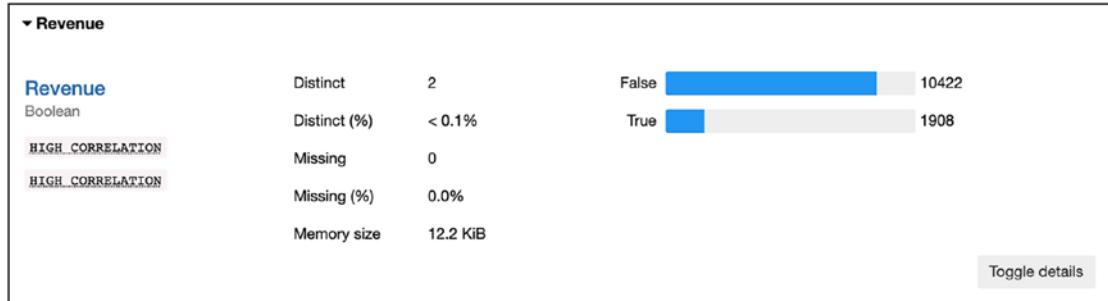


Figure 9-19. Details for Revenue

Figure 9-20 shows the correlation matrix of the input variables.

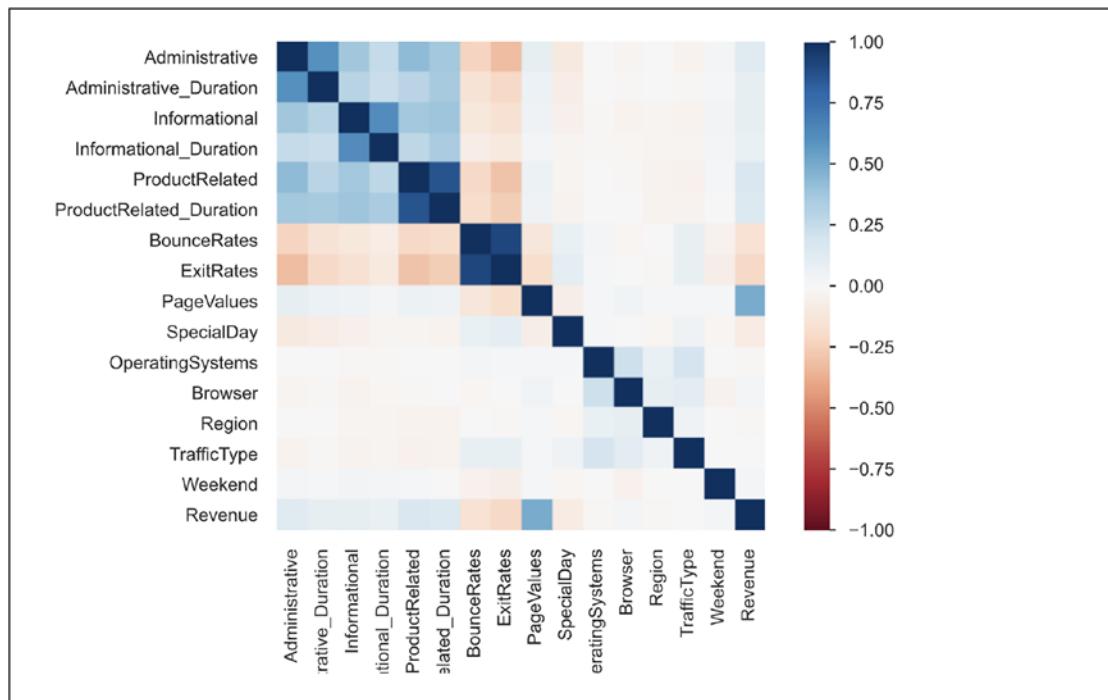


Figure 9-20. Correlation matrix of input variables for the data set

You can see the following in the chart.

- Administrative is highly correlated with Administrative_Duration
- Informational is highly correlated with Informational_Duration
- ProductRelated is highly correlated with ProductRelated_Duration
- BounceRates is highly correlated with ExitRates

To understand how correlated variables affect different explainability methods, these variables are kept in the model.

Now, let's look at the relationship of all features with the response variable.

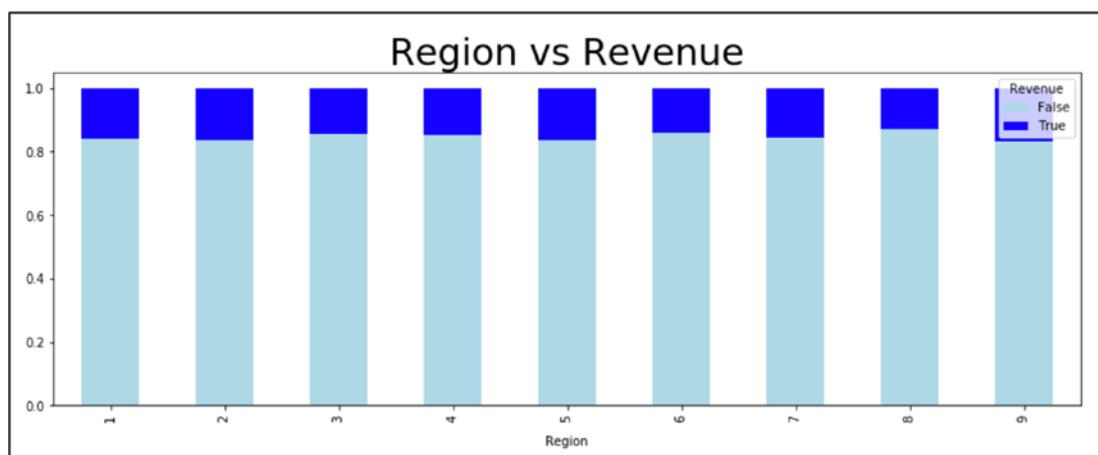


Figure 9-21. Region vs. Revenue distribution

There is not much difference in distribution based on region in the Revenue variable, as shown in Figure 9-22.

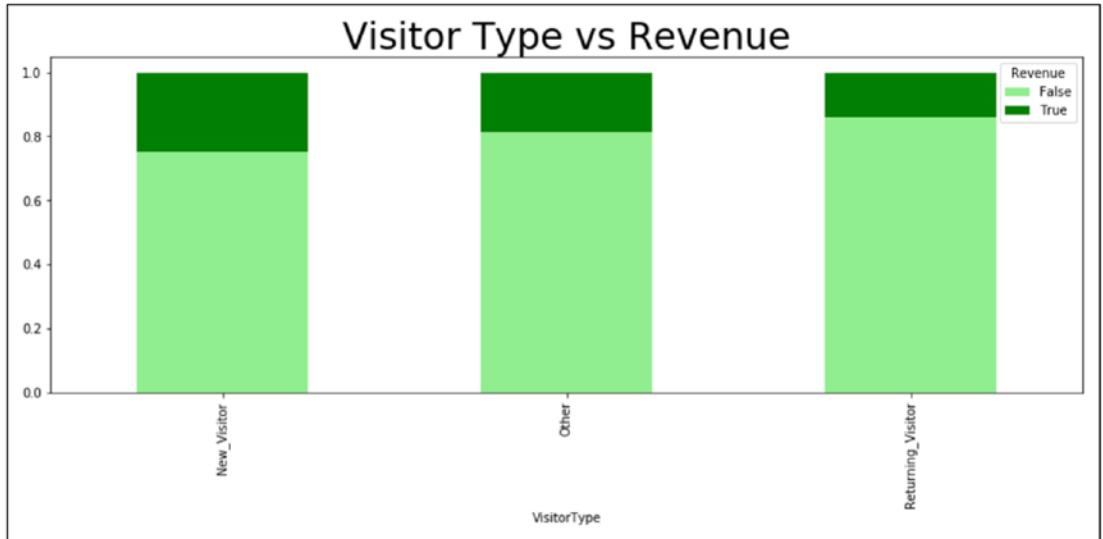


Figure 9-22. Visitor type vs. Revenue distribution

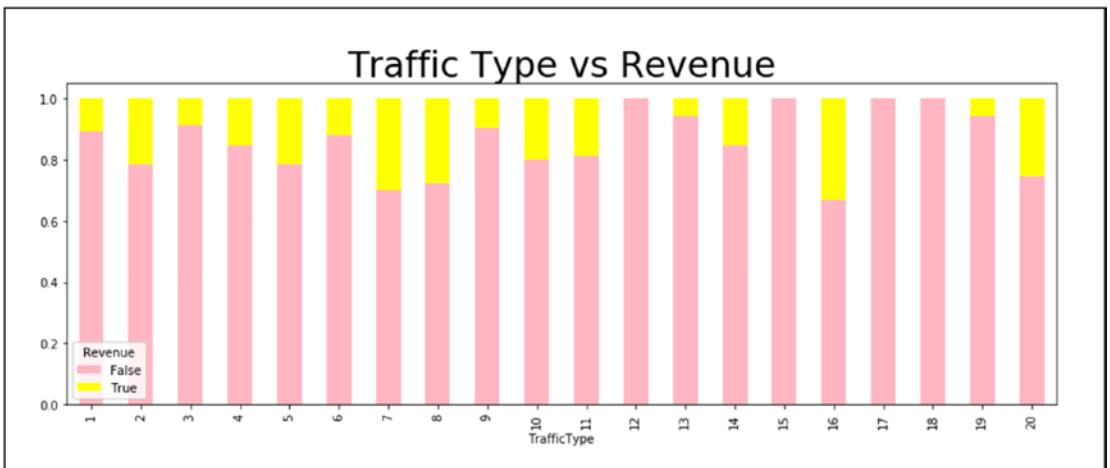


Figure 9-23. Traffic type vs. Revenue

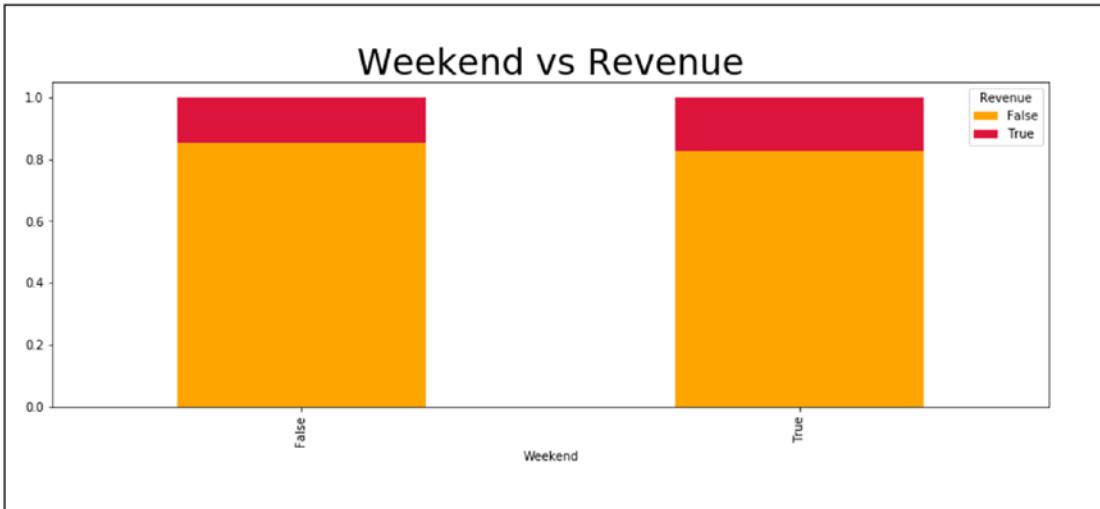


Figure 9-24. Weekend vs. Revenue



Figure 9-25. Page Values and Bounce Rates vs. Revenue

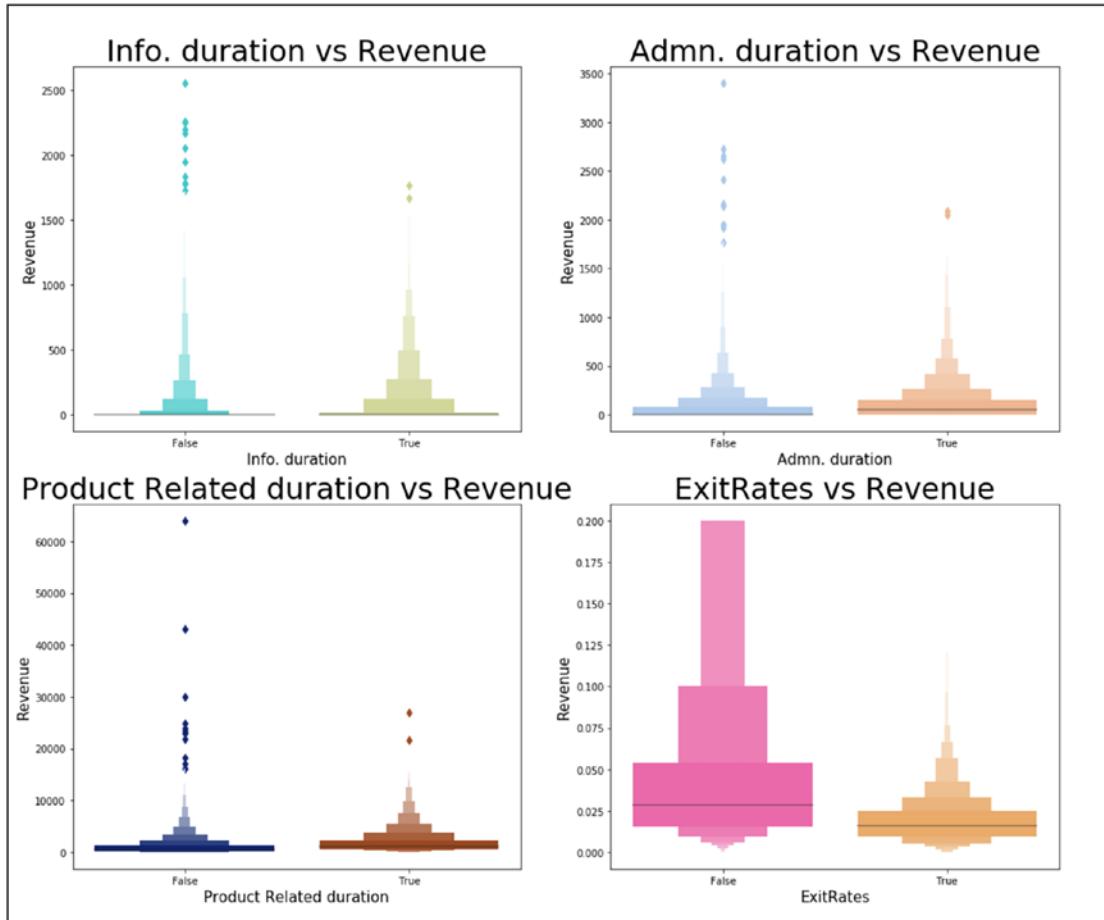


Figure 9-26. Information duration, Administration duration, Product-related duration, and Exit rates vs. Revenue

From the plots in Figure 9-26, you can see that ExitRate might be a good variable to use in the model.

Next, we split the data set into training and test data set.

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.1)
```

Next, build the model.

```
model=RandomForestClassifier(n_estimators=150)
model.fit(x_train,y_train)
```

Do some parameter tuning.

```
from sklearn.model_selection import GridSearchCV
grid_values = {'n_estimators': [200,300], 'min_samples_leaf':[2,3],
'criterion':['gini', 'entropy']}
grid_clf_acc = GridSearchCV(model, param_grid = grid_values,
scoring = 'accuracy')
grid_clf_acc.fit(x_train, y_train)
y_pred_acc = grid_clf_acc.predict(x_test)
```

Let's generate the performance metrics.

```
print(accuracy_score(y_test,y_pred_acc))
print(precision_score(y_test,y_pred_acc))
print(recall_score(y_test,y_pred_acc))
print(f1_score(y_test,y_pred_acc))
```

Accuracy is: 0.8972668288726683

Precision is: 0.7585380116959064

Recall is: 0.5894285714285714

F1 score is: 0.6631012658227848

The data set has a decent performance. Accuracy is a bit higher as this is an imbalanced classification.

Once the model building exercise is complete, we can start building explainability methods on top of this model to generate feature importance scores.

Random Forest Feature Importance

The following are the random forest method's characteristics.

- Post hoc
- Global
- Model specific
- Visualization based

Random forest feature importance can be computed in two ways: one based on how much accuracy decreases when a variable is excluded. The other one is based on a decrease in Gini impurity when a variable splits a node.

Accuracy-based Importance

The accuracy-based importance method uses observations to arrive at importance measures for the variables. Every tree that is built inside the random forest model has its sample of data. The prediction accuracy is measured. The values of the variable in the sample are then randomly shuffled while keeping all other variables the same. The decrease in prediction accuracy of the shuffled data is then measured. This decrease is measured across all trees, and the mean is taken. The importance is broken down into each class of the model. This measuring a variable's importance determines how much accuracy decreases by removing a variable and the opposite. The resultant change in accuracy due to shuffling is less when the variable has little contribution to the predictive power.

Gini-based Importance

The decision by the tree to split a variable at each node is sometimes taken based on Gini impurity. For every variable across every tree in the random forest model, the sum of the Gini decrease is calculated whenever the tree chooses to split a node based on the variable. Then we take an average across all trees. Scale is irrelevant in the variable that comes out; only the order matters. The advantage of this method is that these calculations on Gini impurity are already performed during the model training phase, so minimal computation is required to calculate the relevant scores.

This code follows the model building exercise. We use the same model function.

```
importance = model.feature_importances_
for i,v in enumerate(importance):
    print('Feature: %d, Score: %.5f' % (i,v))
```

```
Feature: 0, Score: 0.04181
Feature: 1, Score: 0.05627
Feature: 2, Score: 0.01603
Feature: 3, Score: 0.02709
```

CHAPTER 9 FEATURE IMPORTANCE METHODS: DETAILS AND USAGE EXAMPLES

```
Feature: 4, Score: 0.07756
Feature: 5, Score: 0.08788
Feature: 6, Score: 0.05604
Feature: 7, Score: 0.09298
Feature: 8, Score: 0.37748
Feature: 9, Score: 0.00359
Feature: 10, Score: 0.04402
Feature: 11, Score: 0.01768
Feature: 12, Score: 0.01873
Feature: 13, Score: 0.02982
Feature: 14, Score: 0.03250
Feature: 15, Score: 0.01127
Feature: 16, Score: 0.00926
```

Let's plot the results as default results does not show feature names.

```
# plot feature importance
plt.figure(figsize=(16,8))
plt.bar(df.iloc[:, :-1].columns, [i*100 for i in importance], color='green')
plt.title('Column wise Feature Importance score in %')
plt.xlabel('Feature Name')
plt.ylabel('Importance score in %')
plt.xticks(rotation=90)
plt.grid()
plt.show()
```

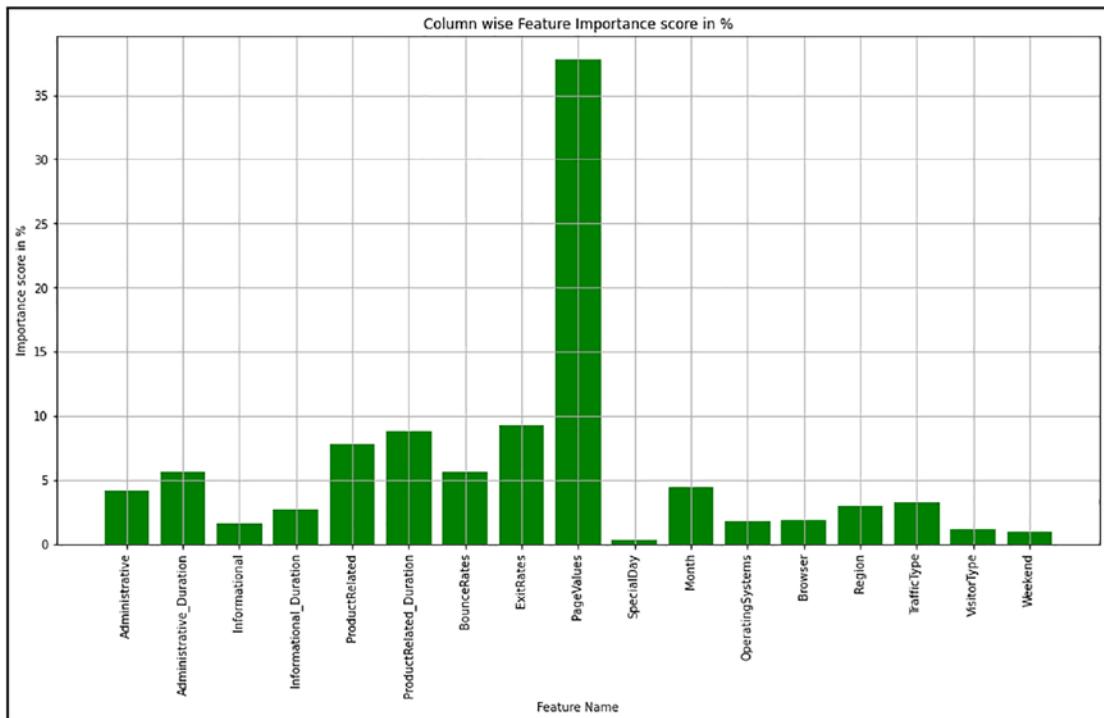


Figure 9-27. The random forest feature importance visualization showcasing the most important features

The built-in basic random forest feature importance method shows that PageValues is the most important variable at a global level for the model. This means that PageValues alone contains the maximum information to describe the decision taken by the model at an overall level.

The job of a technical model builder gets over at this point, and the job of a model reviewer starts. The model reviewer communicates these results to the stakeholders and says that they should probably look at the PageValues variable to understand the difference in the two classes of the model.

Permutation Feature Importance

The following are the permutation feature importance model's characteristics.

- Post hoc
- Global

CHAPTER 9 FEATURE IMPORTANCE METHODS: DETAILS AND USAGE EXAMPLES

- Model agnostic
- Importance based

The permutation feature importance method measures the increase in the model's error when a particular feature's value is permuted hence breaking the relationship between the feature and the response variable.

The concept behind permutation feature importance in simple terms can be described as follows.

Input is important to the model if changing the values of that feature increases the model error. A feature is "unimportant" if shuffling its values leaves the model error unchanged because the model ignored the feature for the prediction.

Breiman introduced the permutation feature importance measurement in 2001 for random forests.

The input for the permutation feature importance is a trained model t , a feature matrix X and target vector y , and an error measure, E .

The original model error can be written as $E(y, t(x))$.

Estimate the original model error = $E(y, t(X))$ (e.g., mean squared error)

Then for each feature $s = 1 \dots p$, we run a loop to do the following.

- Generate feature matrix X^{changed} by permuting feature s in the data X . This kills any relationship between s and outcome y .
- Estimate error on permuted data = $E(Y, t(X^{\text{changed}}))$ based on the predictions of the permuted data.
- Calculate permutation feature importance.

$$FI^s = \text{Error on permuted data} / \text{Error on original data.}$$

Alternatively, the difference can be used: $FI^s = \text{Error on permuted data} - \text{Error on original data}$.

Sort features by descending FI.

We can also split the data in half and swap the values of the features of the two halves. This prevents you from permuting the feature.

If you want a different method than the preceding two, you can estimate the error of permuting feature s by pairing each instance with the value of feature s of each other instance but not with itself. This results in a data set of size $n(n-1)$ to estimate the permutation error. However, this takes a large amount of computation time.

Now the permutation feature importance should be run on test data and not on training data. If we measure the model's error on the same data on which it was trained, then the model seems to work better than it is in reality. And as permutation relies on the model's error, it should always be done on unseen data. The importance that comes out of training data because of overfitting gives misleading feature importance.

Advantages

- Easy-to-understand interpretation: the result of permutation importance is easy to understand and present
- Feature importance measurements are comparable across different problems
- Takes into account all interactions with other features
- Permutation feature importance does not require retraining the model

Disadvantages

- Need access to the true outcome. If someone only provides you with the model and unlabeled data—but not the true outcome, you cannot compute the permutation feature importance.
- When the permutation is repeated, the results might vary greatly.
- If features are correlated, the permutation feature importance can be biased by unrealistic data instances.
- Adding a correlated feature can decrease the importance of the associated feature by splitting the importance between both features.

Code

You read the data in the previous section. The same data set is used to develop the code of this method.

CHAPTER 9 FEATURE IMPORTANCE METHODS: DETAILS AND USAGE EXAMPLES

Let's first convert the categorical variables into dummy values.

```
data1 = pd.get_dummies(df, drop_first=True)
data1.columns
```

Now we change the string response variable to numerical.

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['Revenue'] = le.fit_transform(df['Revenue'])
df['Revenue'].value_counts()
```

Remove the target revenue from x.

```
x = data1
x = x.drop(['Revenue'], axis = 1)
y = df['Revenue']
```

Now let's divide data into train and test and build a model.

```
train_X, val_X, train_y, val_y = train_test_split(x, y, random_state=1)
my_model = RandomForestClassifier(n_estimators=100,
                                  random_state=0).fit(train_X, train_y)
```

It's time to compute the permutation feature importance.

```
import eli5
from eli5.sklearn import PermutationImportance

perm = PermutationImportance(my_model, random_state=1).fit(val_X, val_y)
eli5.show_weights(perm, feature_names = val_X.columns.tolist())
```

The output is displayed in Figure 9-28. This table gives the weight associated to each feature (same as logistic regression gives out of box). The value tells how much of an impact a feature has on the predictions on average, the sign states in which direction.

Weight	Feature
0.1235 ± 0.0088	PageValues
0.0091 ± 0.0037	Month_Nov
0.0076 ± 0.0045	ExitRates
0.0062 ± 0.0034	ProductRelated
0.0046 ± 0.0025	Administrative
0.0042 ± 0.0046	BounceRates
0.0040 ± 0.0030	ProductRelated_Duration
0.0014 ± 0.0014	Month_Dec
0.0012 ± 0.0018	TrafficType
0.0010 ± 0.0024	Administrative_Duration
0.0010 ± 0.0014	Month_May
0.0004 ± 0.0013	VisitorType_New_Visitor
0.0004 ± 0.0006	Month_Sep
0.0004 ± 0.0021	Browser
0.0002 ± 0.0003	Month_June
0.0001 ± 0.0008	Month_Aug
0.0001 ± 0.0005	Month_Jul
0.0001 ± 0.0003	Month_Oct
0 ± 0.0000	Month_Feb
0 ± 0.0000	VisitorType_Other
... 8 more ...	

Figure 9-28. Permutation feature importance output for most important variables

As in the previous example, PageValues is indeed the most important variable for this model.

Let's dive into a few more techniques and look at those results.

SHAP

The following are SHAP's characteristics.

- Post hoc
- Global and local
- Model agnostic
- Visualization based

SHAP is based on Shapely values, which are computed in the game theory. But game theory is directly not related to the machine learning domain. The components of game theory can be related to the model components as follows.

- The “game” reproduces the outcome of the model.
- The “players” are the features included in the model.

SHAP quantifies the importance of each feature to the model or the prediction in the same way as Shapely quantifies each player’s contribution to the game. A modeling scenario game can be a single observation of the model. Hence SHAP is by default a local interpretation method, but it can also be used for global interpretability.

Let’s take the example of a machine learning model, which is a linear regression model. It predicts the income of a person using features such as age, gender, and job.

In Shapely values, each possible outcome of the features should be considered to determine the importance of a single feature. Hence it means each possible combination of f features (f going from 0 to F , F -number of all features available).

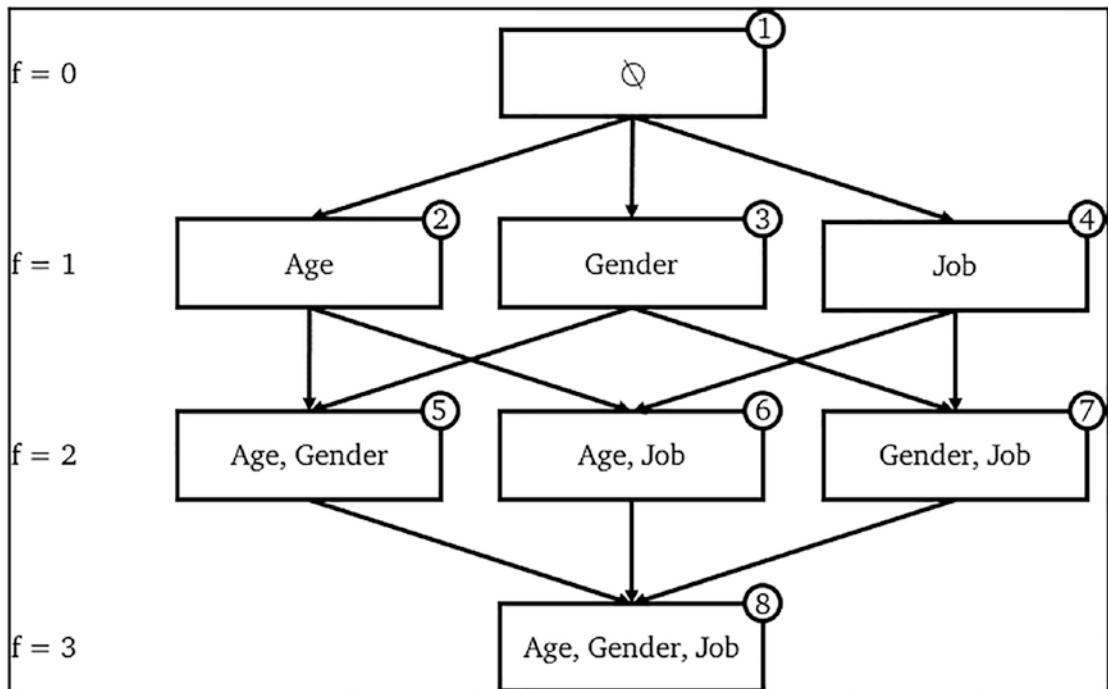


Figure 9-29. Each node represents a coalition of features. Each edge represents the inclusion of a feature not present in the previous coalition

In this case, the total number of possible outcomes hence be $2^F = 2^3 = 8$ because there are three features.

SHAP now trains the model on all these combinations. The models are similar. The only difference is in the feature set of the models.

Now let's take a new observation x_0 and see what the eight different models predict for the same observation x_0 .

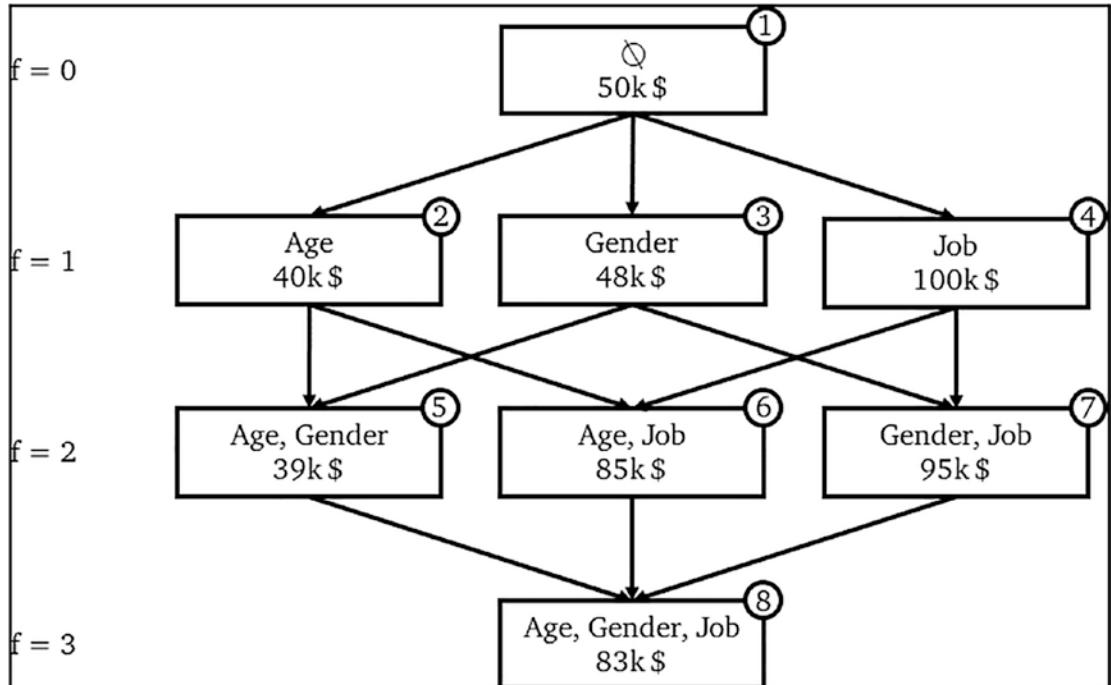


Figure 9-30. Predictions made by different models for x_0 . In each node, the first row reports the coalition of features included in the model, the second row reports the income predicted for x_0 by that model

Figure 9-30 shows that there is one more feature at each level than the previous level. The difference between the feature can be attributed to the marginal contribution of the additional feature. In node 1, which is a model with no features, the model simply predicts the average income of all training observations (\$50K). In node 2, which is a model with one feature (age), the prediction for x_0 is \$40K. This can be interpreted that when you know the age, it lowers the prediction by \$10K.

Thus, the marginal contribution of age to the model containing only age as a feature is \$10K.

To compute the overall effect of age on the final model, we need to consider the marginal contribution of age across all models where age is present. Figure 9-31 highlights those edges in red.

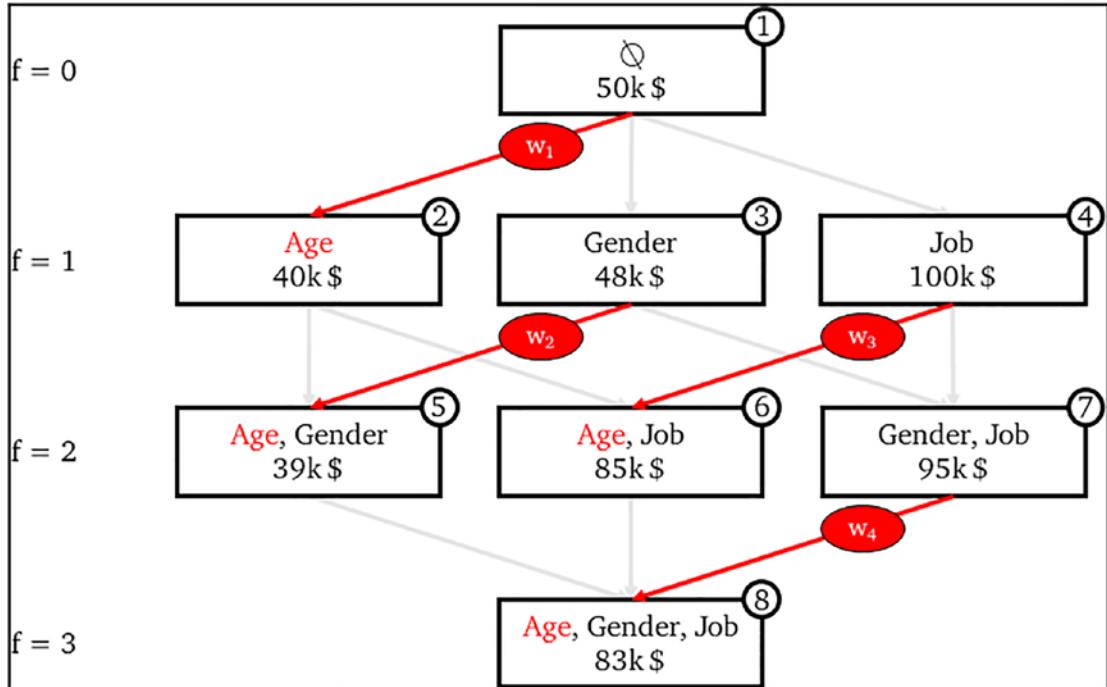


Figure 9-31. Red highlighted edges of the model to be used in age marginal contribution computation

The following is the formula for computing the weighted marginal contribution.

$$\begin{aligned} SHAP_{Age}(x_0) = & w_1 \times MC_{Age, \{Age\}}(x_0) + \\ & w_2 \times MC_{Age, \{Age, Gender\}}(x_0) + \\ & w_3 \times MC_{Age, \{Age, Job\}}(x_0) + \\ & w_4 \times MC_{Age, \{Age, Gender, Job\}}(x_0) \end{aligned}$$

The next question from this formula determines the weights.

The idea is that the sum of the weights of all the marginal contributions to 1-feature-models should equal the sum of all the marginal contributions to 2-feature-models and so on.

In other words, the sum of all the weights on the same row should equal the sum of all the weights on any other row.

In the example, this means $w_1 = w_2 + w_3 = w_4$.

All the weights of marginal contributions to f-feature-models should equal each other for each f.

In other words, all the edges on the same “row” should equal each other. In the example, this means $w_2 = w_3$.

Therefore, if they all sum to 1, the values of the weights should be as follows.

$$\begin{aligned}w_1 &= 1/3 \\w_2 &= 1/6 \\w_3 &= 1/6 \\w_4 &= 1/3\end{aligned}$$

The formula to calculate these values is explained as follows. Each f -feature-model has f marginal contributions (one per feature), so it is enough to count the number of possible f -feature-models and to multiply it by f . Thus, the problem can be simplified by counting the number of possible f -feature-models, given f and knowing that the total number of features is F . This is the definition of binomial coefficient.

You can say that the number of edges in a row can be computed as follows.

$$f \times \binom{F}{f}$$

It is enough to take the reciprocal of this and weight a marginal contribution to an f -feature model.

Figure 9-32 illustrates this.

	N. of Nodes $\binom{F}{f}$	N. of Edges $f \times \binom{F}{f}$
$f = 0$	1	
$f = 1$	3	3
$f = 2$	3	6
$f = 3$	1	3
Sum	$2^F = 8$	$F \times 2^{F-1} = 12$

Figure 9-32. Computation of weights required for computing SHAP marginal value

Now substitute the values of the weights in the formula.

$$\begin{aligned}
 SHAP_{Age}(x_0) &= [(1 \times \binom{3}{1})^{-1} \times MC_{Age, \{Age\}}(x_0) + \\
 &\quad [(2 \times \binom{3}{2})^{-1} \times MC_{Age, \{Age, Gender\}}(x_0) + \\
 &\quad [(2 \times \binom{3}{2})^{-1} \times MC_{Age, \{Age, Job\}}(x_0) + \\
 &\quad [(3 \times \binom{3}{3})^{-1} \times MC_{Age, \{Age, Gender, Job\}}(x_0) + \\
 &= \frac{1}{3} \times (-10k\$) + \frac{1}{6} \times (-9k\$) + \frac{1}{6} \times (-15k\$) + \frac{1}{3} \times (-12k\$) \\
 &= -11.33k\$
 \end{aligned}$$

When generalized to accommodate any model and n number of features, the same formula is as follows.

$$SHAP_{feature}(x) = \sum_{set: feature \in set} [|set| \times \binom{F}{|set|}]^{-1} [Predict_{set}(x) - Predict_{set \setminus feature}(x)]$$

Applied to the example, the formula yields the following.

$$SHAP_Age(x_0) = -11.33k \$$$

$$SHAP_Gender(x_0) = -2.33k \$$$

$$SHAP_Job(x_0) = +46.66k \$$$

Summing them up gives +\$33K, which is exactly the difference between the output of the full model (\$83K) and the output of the dummy model with no features (\$50K).

This is a fundamental characteristic of SHAP values: **summing the SHAP values of each feature of a given observation yields the difference between the prediction of the model and the null model**. Hence, they have the name Shapely Additive explanations.

Let's talk about the whole process of SHAP computation in mathematical terms as described by the authors of SHAP, Lundberg and Lee, in a paper titled “A Unified Approach to Interpreting Model Predictions.”

In the paper, the authors talk about the additive feature attribution methods.

Additive feature attribution methods have an explanation model that is a linear function of binary variables.

$$g(z') = \phi_0 + \sum_{i=1}^M \phi_i z'_i,$$

$z' \in \{0, 1\}^M$, M is the number of simplified input features and $\phi_i \in R$.

Next, let's discuss the properties of additive explanations, which the authors mention as a part of the paper.

Property 1 (Local Accuracy)

$$f(x) = g(x') = \phi_0 + \sum_{i=1}^M \phi_i x'_i$$

The explanation model $g(x')$ matches the original model $f(x)$ when $x = hx(x')$.

Property 2 (Missingness)

$$x'_i = 0 \implies \phi_i = 0$$

Missingness constrains features where $x'_i = 0$ to have no attributed impact. If the simplified inputs represent feature presence, then missingness requires features missing in the original input to have no impact.

Property 3 (Consistency)

Let $fx(z') = f(hx(z'))$ and $z' \setminus i$ denote setting $z'_i = 0$. For any two models f and f' , if

$$f'_x(z') - f'_x(z' \setminus i) \geq f_x(z') - f_x(z' \setminus i)$$

for all inputs $z' \in \{0, 1\}^M$, then $\phi_i(f', x) \geq \phi_i(f, x)$.

Next, the authors mention a mode that follows the definition of additive explanations and satisfies all properties.

$$\phi_i(f, x) = \sum_{z' \subseteq x'} \frac{|z'|!(M - |z'| - 1)!}{M!} [f_x(z') - f_x(z' \setminus i)]$$

$|z'|$ is the number of non-zero entries in z' , and $z' \subseteq x'$ represents all z' vectors where the non-zero entries are a subset of the non-zero entries in x' .

The SHAP or Shapely Additive explanation values are the solutions to the preceding equation. SHAP values provide the unique additive feature importance measure that adheres to all three properties and uses conditional expectations to define simplified inputs.

Kernel SHAP in a five-step process estimates for an instance, the contributions each feature value makes to the prediction.

1. Define the sample coalitions as $z'k \in \{0,1\}^M$, $k \in \{1, \dots, K\}$ (1 = feature present in coalition, 0 = feature absent).
2. Get a prediction for each $z'k$ by first converting $z'k$ to the original feature space and then applying model f : $f(h(x(z'k)))$.
3. The SHAP kernel computes the weight for each $z'k$.
4. Fit the weighted linear model.
5. Return the Shapley values, ϕ_k , which are the coefficients from the linear model.

When we do sampling from the marginal distribution in the SHAP kernel, we ignore the dependence structure between present and absent features.

The KernelSHAP method suffers from the same problem as all permutation-based interpretation methods, which is that the estimation puts too much weight on unlikely instances. This can result in unreliable results. But it is necessary to sample from the marginal distribution. The solution would be to sample from the conditional distribution, which changes the value function, and therefore the game to which Shapley values are the solution. As a result, the Shapley values have a different interpretation; for example, a feature that the model might not have used at all can have a non-zero Shapley value when the conditional sampling is used. For the marginal game, this feature value would always get a Shapley value of 0 because otherwise, it would violate the concept.

The big difference to LIME is the weighting of the instances in the regression model. LIME weights the instances according to how close they are to the original instance. The more 0's in the coalition vector, the smaller the weight in LIME. SHAP weights the sampled instances according to the weight the coalition would get in the Shapley value estimation. Small coalitions (few 1s) and large coalitions (i.e., many 1's) get the largest weights. The intuition behind it is that you learn most about individual features if you study their effects in isolation. If a coalition consists of a single feature, you can learn about the features' isolated main effect on the prediction. If a coalition consists of all but one feature, you can learn about this features' total effect (main effect plus feature interactions). If a coalition consists of half the features, you learn little about an

individual feature's contribution, as there are many possible coalitions with half of the features. To achieve Shapley compliant weighting, Lundberg et. al propose the SHAP kernel, as follows.

$$\pi_x(z') = \frac{(M - 1)}{\binom{M}{|z'|} |z'| (M - |z'|)}$$

Here, M is the maximum coalition size and $|z'|$, the number of present features in instance z' . Lundberg and Lee show that linear regression with this kernel weight yields Shapley values. If you would use the SHAP kernel with LIME on the coalition data, LIME would also estimate Shapley values!

TreeSHAP, a variant of SHAP for tree-based machine learning models such as decision trees, random forests, and gradient boosted trees. TreeSHAP was introduced as a fast, model-specific alternative to KernelSHAP, but it turned out that it can produce unintuitive feature attributions.

TreeSHAP defines the value function using the conditional expectation $\text{EXS}|\text{XC}(f(x)|xS)$ instead of the marginal expectation. The problem with the conditional expectation is that features that do not influence the prediction function f can get a TreeSHAP estimate different from zero. The non-zero estimate can happen when the feature is correlated with another feature that influences the prediction.

TreeSHAP uses the conditional expectation $\text{EXS}|\text{XC}(f(x)|xS)$ to estimate effects. It gives intuition on how to compute the expected prediction for a single tree, an instance x , and feature subset S . If we conditioned all features, and S is the set of all features, the prediction from the node in which the instance x falls would be the expected prediction. If we did no condition on any feature, and S is empty, we would use the weighted average of predictions of all terminal nodes. If S contains some, but not all, features, we ignore predictions of unreachable nodes. Unreachable means that the decision path that leads to this node contradicts values in xS . From the remaining terminal nodes, we average the predictions weighted by node sizes (i.e., number of training samples in that node). The mean of the remaining terminal nodes, weighted by the number of instances per node, is the expected prediction for x given S . The problem is that we must apply this procedure for each possible subset S of the feature values. TreeSHAP computes in polynomial time instead of exponential. The basic idea is to push all possible subsets S down the tree at the same time. For each decision node we must keep track of the number of subsets. This depends on the subsets in the parent node and the split feature.

Let's use the same df we read from the CSV in the previous example.

Convert categorical variables into dummies.

```
data1 = pd.get_dummies(data, drop_first=True)
```

```
data1.columns
```

```
Index(['Administrative', 'Administrative_Duration', 'Informational',
       'Informational_Duration', 'ProductRelated', 'ProductRelated_Duration',
       'BounceRates', 'ExitRates', 'PageValues', 'SpecialDay',
       'OperatingSystems', 'Browser', 'Region', 'TrafficType', 'Weekend',
       'Revenue', 'Month_Aug', 'Month_Dec', 'Month_Feb', 'Month_Jul',
       'Month_June', 'Month_Mar', 'Month_May', 'Month_Nov', 'Month_Oct',
       'Month_Sep', 'VisitorType_New_Visitor', 'VisitorType_Other',
       'VisitorType_Returning_Visitor'],
      dtype='object')
```

Figure 9-33.

Let's label the Revenue column.

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data['Revenue'] = le.fit_transform(data['Revenue'])
data['Revenue'].value_counts()
```

0	10422
1	1908
Name: Revenue, dtype: int64	

Figure 9-34. Class labels output

Let's separate dependent and independent variables.

```
x = data1
# removing the target column revenue from x
```

```
x = x.drop(['Revenue'], axis = 1)  
y = data['Revenue']  
# checking the shapes  
print("Shape of x:", x.shape)  
print("Shape of y:", y.shape)
```

Next, we split the data.

```
# splitting the data  
  
from sklearn.model_selection import train_test_split  
  
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3,  
random_state = 0)  
  
# checking the shapes  
  
print("Shape of x_train :", x_train.shape)  
print("Shape of y_train :", y_train.shape)  
print("Shape of x_test :", x_test.shape)  
print("Shape of y_test :", y_test.shape)
```

```
Shape of x_train : (8631, 28)  
Shape of y_train : (8631, )  
Shape of x_test : (3699, 28)  
Shape of y_test : (3699, )
```

Figure 9-35. Code output of data frame shape

We now build the model.

```
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import confusion_matrix  
from sklearn.metrics import classification_report
```

```
model = RandomForestClassifier()
model.fit(x_train, y_train)

y_pred = model.predict(x_test)

# evaluating the model
print("Training Accuracy :", model.score(x_train, y_train))
print("Testing Accuracy :", model.score(x_test, y_test))

# confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.rcParams['figure.figsize'] = (6, 6)
sns.heatmap(cm ,annot = True)

# classification report
cr = classification_report(y_test, y_pred)
print(cr)
```

```
Training Accuracy : 0.9901517784729463
Testing Accuracy : 0.8902406055690727
      precision    recall  f1-score   support
          0       0.91      0.97      0.94     3077
          1       0.75      0.52      0.61      622

      accuracy                           0.89     3699
   macro avg       0.83      0.74      0.78     3699
weighted avg       0.88      0.89      0.88     3699
```

Figure 9-36. Code output of accuracy

Now let's look at the SHAP values for this data set.

```
#importing shap
import shap

explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(x_test)

shap.summary_plot(shap_values[1], x_test, plot_type = 'bar')
```

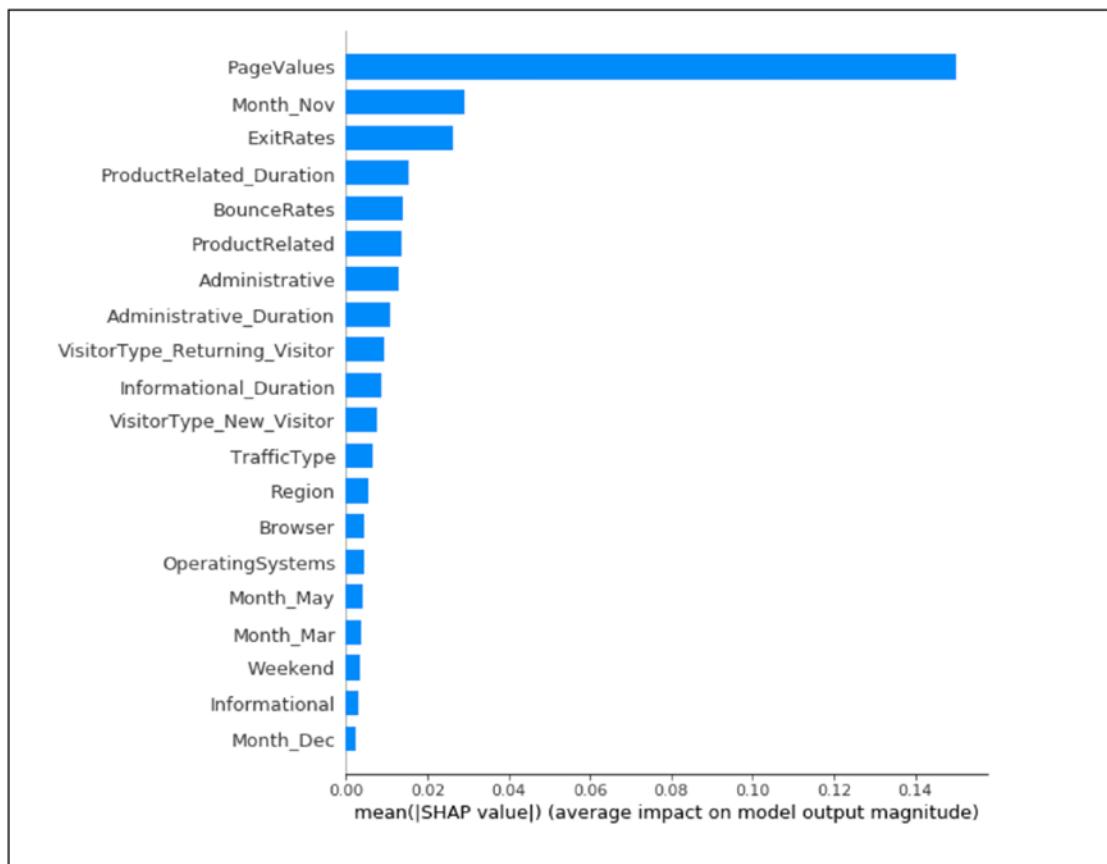


Figure 9-37. Idea behind the SHAP feature importance plot. Features with large absolute Shapley values are important

As in the previous example, the PageValues variable is the most important. Let's now look at the butterfly plot to understand the behavior a bit more deeply.

```
shap.summary_plot(shap_values[1], x_test)
```

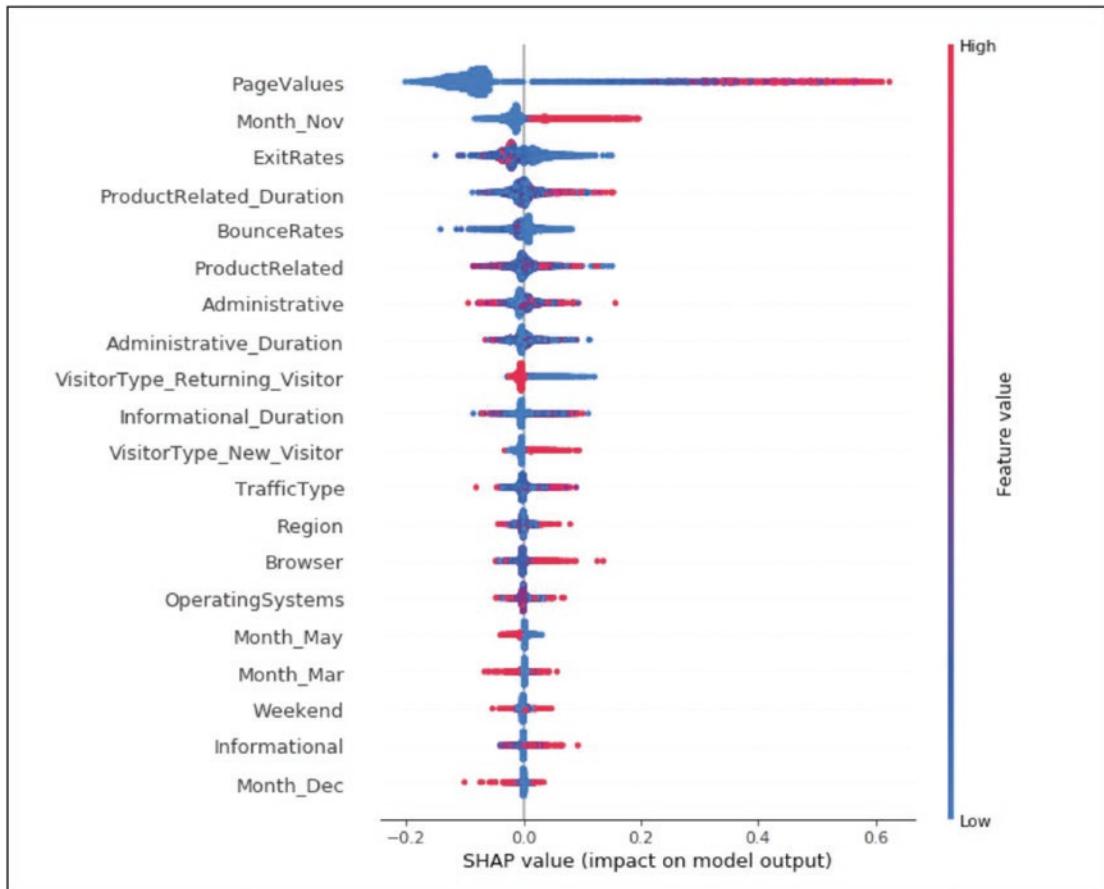


Figure 9-38. SHAP summary plot

The summary plot combines feature importance with feature effects. Each point on the summary plot is a Shapley value for a feature and an instance. The position on the y axis is determined by the feature and on the x axis by the Shapley value. The color represents the value of the feature from low to high. Overlapping points are jittered in the y-axis direction, so you get a sense of the distribution of the Shapley values per feature.

The features are ordered according to their importance. Figure 9-38 shows that higher values of PageValues have higher SHAP values, which means higher values of PageValues correspond to the prediction of positive class in the model.

CHAPTER 9 FEATURE IMPORTANCE METHODS: DETAILS AND USAGE EXAMPLES

Now let's look at the different force plots. Force plots allow you to see how features contributed to the model's prediction for a specific observation. This is perfect for explaining to someone exactly how your model arrived at the prediction it did for a specific observation.

```
customers = x_test.iloc[1,:].astype(float)
customer_analysis(model, customers)
```



Figure 9-39. SHAP force plot

The average prediction for this observation is 0.17. The prediction is lowered by the contribution of PageValue = 0.

```
customers = x_test.iloc[15,:].astype(float)
customer_analysis(model, customers)
```



Figure 9-40. SHAP force plot

The average prediction here is 0.07, which is again largely driven by the effects of PageValue = 0.

```
shap_values = explainer.shap_values(x_train.iloc[:50])
shap.initjs()
shap.force_plot(explainer.expected_value[1], shap_values[1], x_test.
iloc[:50])
```

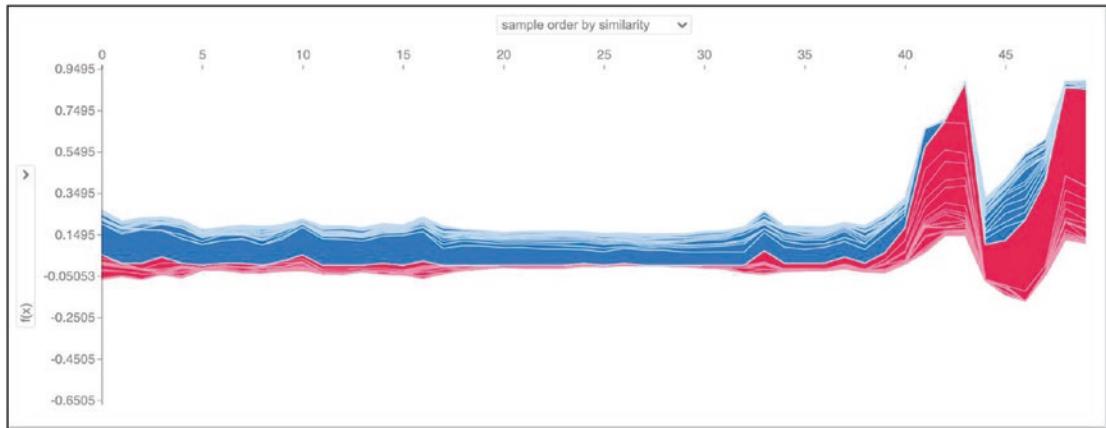


Figure 9-41. SHAP force plot

The plot shows stacked SHAP explanations clustered by explanation similarity. Each position on the x axis is an instance of the data. Red SHAP values increase the prediction. The blue values decrease it. A cluster stands out: On the right is a group with a high predicted probability of online shoppers.

SAGE

The following are the SAGE model's characteristics.

- Post hoc
- Global
- Model agnostic
- Importance based

SAGE (Shapley Additive Global ImportancE) is a global method that is very useful to understanding which features are important for the model performance and behavior. SAGE is similar to the global feature importance method we studied previously in this chapter; however, it is statistically more stable as it uses the Shapely value. You have seen the importance of Shapely while studying SHAP.

This section shows how SAGE applies Shapley values to provide a different kind of model. You learn how much the model function, f , relies on each feature X_1, X_2, \dots, X_d overall.

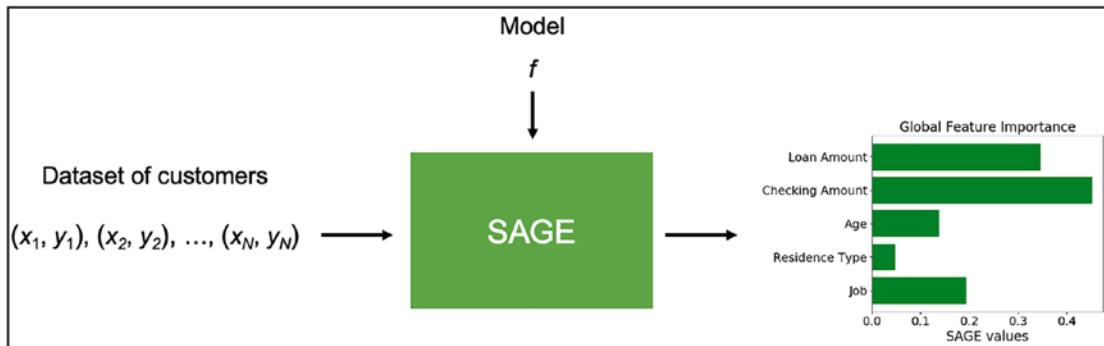


Figure 9-42. SAGE flowchart

To understand which features the technical expert derives the most information from, you can run an experiment where we deprive them of information and see how much their prediction accuracy suffers. For example, you can take a large sample of customers and ask the technical expert to predict their probability of repayment given everything but the customer's age. Their accuracy probably drops by a little bit. Or, you can ask the expert to predict the probability of repayment given everything except age, checking account balance, and job. Now their accuracy should drop by a lot because they're being deprived of critical information.

To apply the same logic to a ML model, f , we must again confront the problem that f requires a fixed set of features. We can use the same trick and deal with missing features using their conditional distribution $S^-|XS=xS$. We can now define a cooperative game that represents the model's performance given subsets of features. Given a loss function ℓ (e.g., MSE or cross-entropy loss), the game v_f is defined as follows.

$$v_f(S) = -E[\ell(E[f(X)| XS], Y)]$$

For any subset $S \subseteq D$, the quantity $v_f(S)$ represents f 's performance given the features XS . A minus sign is in front of the loss so that lower loss (improved accuracy) increases the value $v_f(S)$.

Now, we can use the Shapley values $\phi_i(v_f)$ to quantify each feature's contribution to the model's performance. The features that are most critical for the model to make good predictions have large values $\phi_i(v_f) > 0$, while unimportant features have small values $\phi_i(v_f) \approx 0$, and only features that make the model's performance worse have negative values $\phi_i(v_f) < 0$. These are SAGE values, and that's SAGE in a nutshell.

SAGE provides insight into the intrinsic properties of the data distribution (which might be called explaining the data, rather than explaining the model). SAGE unifies several existing feature importance methods.

SAGE is primarily a tool for model interpretation, but it can also provide insight into intrinsic relationships in the data. For example, when SAGE is applied with the Bayes classifier $f(x)=p(y|x)$ and cross-entropy loss, SAGE values are equal to the Shapley values of the mutual information function $v_f(S)=I(Y;XS)$. This is good news for users who use ML to learn about the world (e.g., which genes are associated with breast cancer) and who aren't as interested in models for their own sake. The framework of *additive importance measures* connects many methods that were previously viewed as unrelated. For example, SAGE differs from permutation tests, one of the methods in the framework. Permutation tests, proposed by Breiman, calculate how much the model performance drops when each column of the data set is permuted. SAGE can be viewed as a modified version of a permutation test.

Instead of holding out one feature at a time, SAGE holds out larger subsets of features. (By only removing individual features, permutation tests may erroneously assign low importance to features with good proxies.)

SAGE draws held out features from their conditional distribution $p(XXXS^-|XS^-)$ rather than their marginal distribution $p(XS^-)$. (Using the conditional distribution simulates a feature's absence, whereas using the marginal breaks feature dependencies and produces unlikely feature combinations.). This is SAGE in a nutshell, and hopefully, this helps you understand its foundation in game theory and its importance for understanding model behavior.

The code implementation of SAGE is complicated. Let's look at it step by step.

Define the list of features for the model.

```
# Feature names and categorical columns (for CatBoost model)
feature_names = df.columns.tolist()[:-1]
categorical_columns = [
    'Month', 'OperatingSystems', 'Browser', 'Region', 'TrafficType',
    'VisitorType', 'Weekend'
]
categorical_inds = [feature_names.index(col) for col in categorical_
columns]
```

Split data into train and test.

```
train, test = train_test_split(
    df.values, test_size=int(0.1 * len(df.values)), random_state=0)
train, val = train_test_split(
    train, test_size=int(0.1 * len(df.values)), random_state=0)
Y_train = train[:, -1].copy().astype(int)
Y_val = val[:, -1].copy().astype(int)
Y_test = test[:, -1].copy().astype(int)
train = train[:, :-1].copy()
val = val[:, :-1].copy()
test = test[:, :-1].copy()
```

Build a model function and train the model on the input data.

```
import numpy as np
from sklearn.metrics import log_loss
from catboost import CatBoostClassifier

model = CatBoostClassifier(iterations=50,
                           learning_rate=0.3,
                           depth=3)

model = model.fit(train, Y_train, categorical_inds, eval_set=(val, Y_val),
                   verbose=False)
```

Calculate performance of the model.

```
p = np.array([np.sum(Y_train == i) for i in np.unique(Y_train)]) /
len(Y_train)
base_ce = log_loss(Y_test.astype(int), p[np.newaxis].repeat(len(test), 0))
ce = log_loss(Y_test.astype(int), model.predict_proba(test))

print('Base rate cross entropy = {:.3f}'.format(base_ce))
print('Model cross entropy = {:.3f}'.format(ce))

import sage
```

Set up and calculate the SAGE values.

```
imputer = sage.MarginalImputer(model, train[:512])
estimator = sage.PermutationEstimator(imputer, 'cross entropy')
sage_values = estimator(test, Y_test)
```

Print the SAGE plot.

```
sage_values.plot(feature_names, title='Feature Importance (Marginal Sampling)')
```

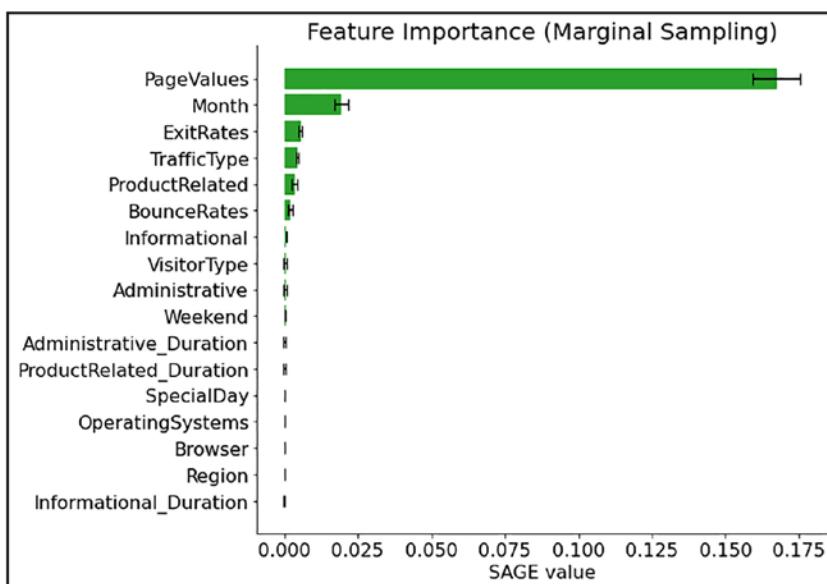


Figure 9-43. SHAP plot output

In this data set, PageValues is the most important feature. These results are in line with the output of the earlier methods. However, SAGE values are much more reliable and can be trusted more than normal random forest feature importance.

How SHAP and SAGE Are Related

SHAP and SAGE use Shapley values, but since they answer fundamentally different questions about a model (local vs. global interpretability), it isn't immediately clear whether their connection goes deeper.

It turns out that there is no simple relationship between SAGE values $\phi_i(vf)$ and SHAP values $\phi_i(vf, x)$. However, SAGE values are related to a SHAP variant known as LossSHAP. Instead of explaining an individual prediction using the cooperative game v_f, x , we can use a game v_f, x, y that represents the loss for an individual input-output pair (x, y) given a subset of features x_S .

$$v_f, x, y(S) = -\ell(E[f(X)|X_S=x_S], y)$$

The Shapley values of this game $\phi_i(vf, x, y)$ are called *LossSHAP values*, and they represent how much each feature contributes to the prediction's accuracy. Features that make the prediction more accurate have large values $\phi_i(vf, x, y) > 0$, and features that make the prediction less accurate have negative values $\phi_i(vf, x, y) < 0$.

SAGE values are equal to the expectation of the LossSHAP values. Mathematically, it is as follows.

$$\phi_i(vf) = E_{Y|X}[phi_i(vf, X, Y)]$$

The connection is important because it shows that a global explanation can be obtained via many local explanations: SAGE values can be calculated by calculating LossSHAP values for the whole data set and then taking the mean.

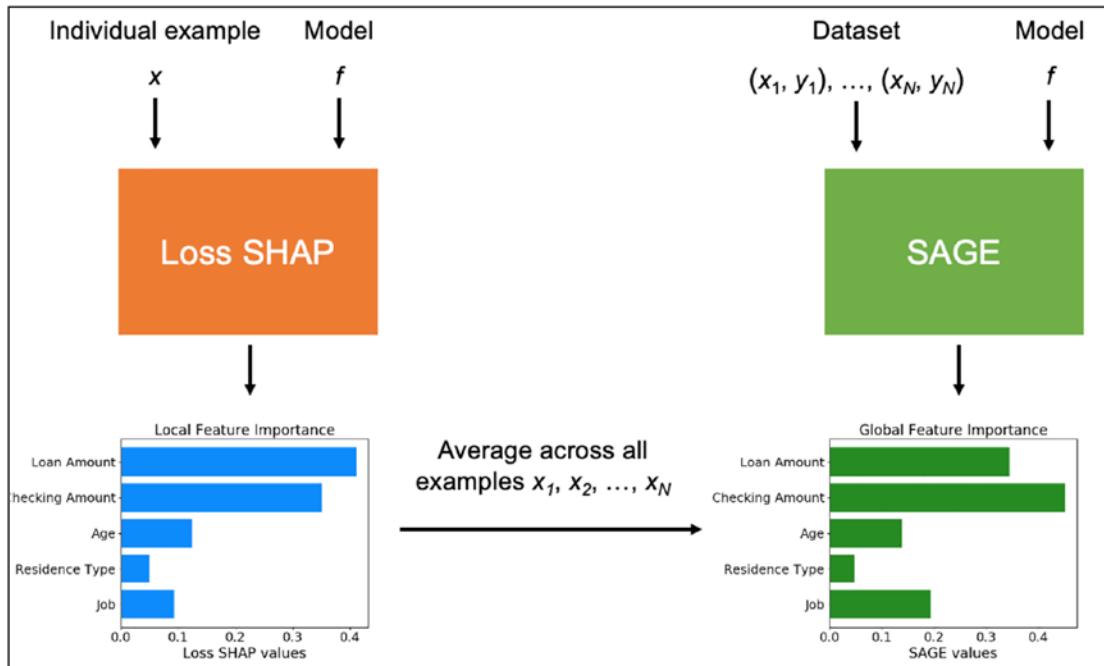


Figure 9-44. SAGE values are equal to the mean LossSHAP value

However, waiting for LossSHAP values to converge for hundreds or thousands of examples in the data set takes a long time. The SAGE method proposes a faster approximation algorithm that aims directly at a global explanation—an approach referred to as *SAGE sampling*. With SAGE sampling, SAGE values can be calculated in the same amount of time as a handful of individual LossSHAP values.

LIME

LIME is model agnostic, meaning that it can be applied to any machine learning model. The technique attempts to understand the model by perturbing the input of data samples and understanding how the predictions change.

Model-specific approaches aim to understand the black-box machine learning model by analyzing the internal components and interaction. In deep learning models, it is possible to investigate activation units and link internal activations back to the input. This requires a thorough understanding of the network and doesn't scale to other models.

LIME provides local model interpretability. LIME modifies a single data sample by tweaking the feature values and observing the output's resulting impact. Often, this is also related to what humans are interested in when observing the output of a model. The most common question is probably: why was this prediction made or which variables caused the prediction?

Other model interpretability techniques only answer this question from the perspective of the entire data set. Feature importance's explain on a data set level which features are important. It allows you to verify hypotheses and whether the model is overfitting to noise, but it is hard to diagnose specific model predictions.

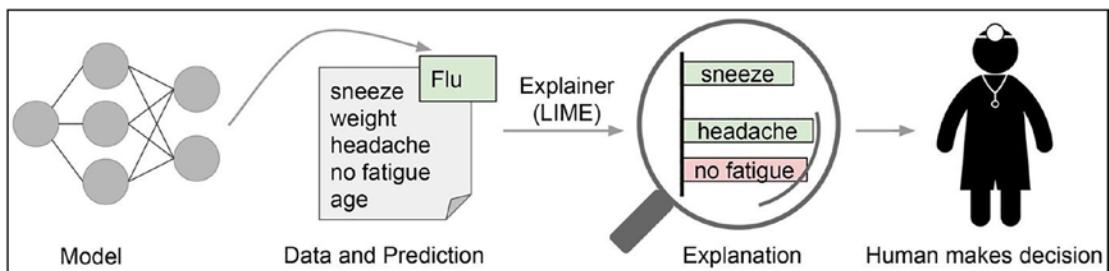


Figure 9-45. Sample model interpretation

A key requirement for LIME is to work with an interpretable representation of input that is understandable to humans. Examples of interpretable representations are a BoW vector for NLP or an image for computer vision. On the other hand, dense embeddings are not interpretable, and applying LIME probably won't improve interpretability.

The output of LIME in Figure 9-46 is a list of explanations, reflecting the contribution of each feature to the prediction of a data sample. This provides local interpretability, and it also allows to determine which feature changes have the most impact on the prediction.

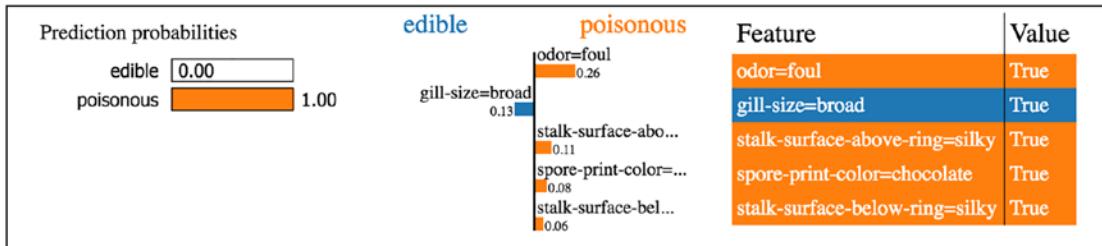


Figure 9-46. Sample lime output

An explanation is created by approximating the underlying model locally by an interpretable one. Interpretable models are linear models with strong regularization, decision trees, and so forth. The interpretable models are trained on small perturbations of the original instance and should only provide a good local approximation. The data set is created by adding noise to continuous features, removing words, or hiding parts of the image. By only approximating the black box *locally* (in the neighborhood of the data sample), the task is significantly simplified.

The overall goal of LIME is to identify an interpretable model over the interpretable representation that is locally faithful to the classifier.

Before we present the explanation system, it is important to distinguish between features and interpretable data representations. As mentioned, interpretable explanations need to use a representation that is understandable to humans, regardless of the actual features used by the model. For example, a possible interpretable representation for text classification is a binary vector indicating the presence or absence of a word, even though the classifier may use more complex (and incomprehensible) features such as word embeddings. Likewise, for image classification, an interpretable representation may be a binary vector indicating the “presence” or “absence” of a contiguous patch of similar pixels (a superpixel). At the same time, the classifier may represent the image as a tensor with three color channels per pixel. $x \in \mathbb{R}^d$ is denoted

the original representation of an instance being explained, and $x_0 \in \{0, 1\}^d$ denotes a binary vector for its interpretable representation. Figure 9-47 is an illustration of this process.

The original model's decision function is represented by the blue/pink background and is nonlinear. The bright red cross is the instance being explained (let's call it X). We sample perturbed instances around X and weigh them according to their proximity to X (weight here is represented by size). We get the original model's prediction on these perturbed instances. We then learn a linear model (dashed line) that approximates the model well in the vicinity of X. Note that the explanation is not faithful globally, but it is faithful locally around X.

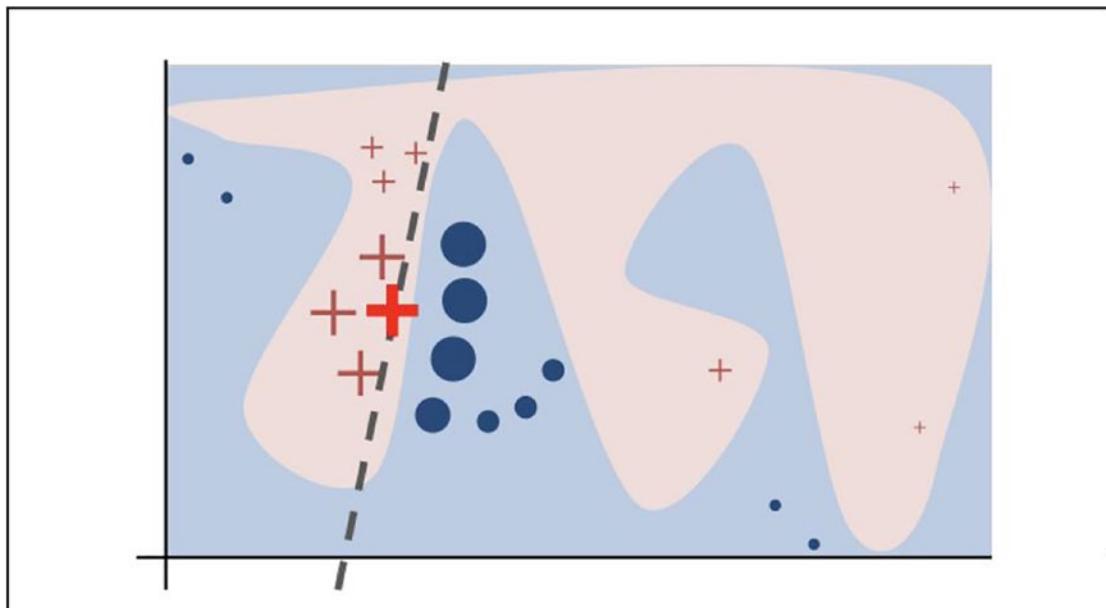


Figure 9-47. The black-box model's complex decision function f (unknown to LIME) is represented by the blue/pink background, which cannot be approximated well by a linear model. The bold red cross is the instance being explained. LIME sample instances get predictions using f , and weigh them by the proximity to the instance being explained (represented here by size). The dashed line is the learned explanation that is locally (but not globally) faithful

LIME for text differs from LIME for tabular data. Variations of the data are generated differently: Starting from the original text, new texts are created by randomly removing words from the original text. The data set is represented with binary features for each word. A feature is 1 if the corresponding word is included and 0 if it has been removed.

LIME for images works differently than LIME for tabular data and text. Intuitively, it would not make much sense to perturb individual pixels since more than one pixel contributes to one class. Randomly changing individual pixels would probably not change the predictions by much. Therefore, variations of the images are created by segmenting the image into superpixels that can be turned off or on. Superpixels are interconnected pixels with similar colors and can be turned off by replacing each pixel with a user-defined color such as gray. The user can also specify a probability for turning off a superpixel in each permutation.

Even if you replace the underlying machine learning model, you can still use the same local, interpretable model for explanation. Suppose the people looking at the explanations understand decision trees best. Because you use local surrogate models, you use decision trees as explanations without using a decision tree to make the predictions. For example, you can use an SVM. And if it turns out that an XGBoost model works better, you can replace the SVM and still use a decision tree to explain the predictions.

Local surrogate models benefit from the literature and experience of training and interpreting interpretable models. When using Lasso or short trees, the resulting explanations are short (= selective) and possibly contrastive. Therefore, they make human-friendly explanations. Therefore, LIME appears more in applications where the recipient of the explanation is a lay person or someone with very little time. It is not sufficient for complete attributions, so we do not see LIME in compliance scenarios where you might be legally required to fully explain a prediction. Also, it is useful to have all the reasons for debugging machine learning models instead of a few.

LIME is one of the few methods that work for tabular data, text, and images.

The fidelity measure gives a good idea of how reliable the interpretable model is in explaining the black-box predictions about the data instance of interest.

The explanations created with local surrogate models can use other (interpretable) features than the original model was trained on. Of course, these interpretable features must be derived from the data instances. A text classifier can rely on abstract word embeddings as features, but the explanation can be based on the presence or absence of words in a sentence. A regression model can rely on a non-interpretable transformation of some attributes, but the explanations can be created with the original attributes. For example, the regression model could be trained on components of a principal component analysis (PCA) of answers to a survey. LIME might be trained on the original survey questions. Using interpretable features for LIME can be a big advantage over other methods, especially when the model was trained with non-interpretable features.

The correct definition of the neighborhood is a very big, unsolved problem when using LIME with tabular data. In my opinion, it is the biggest problem with LIME and the reason why I would recommend using LIME only with great care. For each application, you have to try different kernel settings and see for yourself if the explanations make sense. Unfortunately, this is the best advice I can give to find good kernel widths.

Sampling could be improved in the current implementation of LIME. Data points are sampled from a Gaussian distribution, ignoring the correlation between features. This can lead to unlikely data points, which can then be used to learn local explanation models.

The complexity of the explanation model has to be defined in advance. This is a small complaint because, in the end, the user always must define the compromise between fidelity and sparsity.

Another big problem is the instability of the explanations. In an article, the authors showed that the explanations of two very close points varied greatly in a simulated setting. Also, in my experience, if you repeat the sampling process, then the explanations that come out can be different. Instability means that it is difficult to trust the explanations, and you should be very critical. Data scientists can manipulate LIME explanations to hide biases. The possibility of manipulation makes it more difficult to trust explanations generated with LIME.

Let's start by importing the required libraries.

```
import pandas as pd
import numpy as np
import pandas as pd
import os
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
```

Read the data and do preprocessing for categorical variables.

```
data = pd.read_csv('online_shoppers_intention.csv')
train,test=train_test_split(data,test_size=0.3,random_state=0,stratify=data
['Revenue'])
# Convert categorical variables into dummy/indicator variables
train_processed = pd.get_dummies(train)
test_processed = pd.get_dummies(test)
```

CHAPTER 9 FEATURE IMPORTANCE METHODS: DETAILS AND USAGE EXAMPLES

Fill in any null values.

```
train_processed = train_processed.fillna(train_processed.mean())
test_processed = test_processed.fillna(test_processed.mean())
```

Create train and test data.

```
X_train = train_processed.drop(['Revenue'], axis=1)
Y_train = train_processed['Revenue']

X_test = test_processed.drop(['Revenue'], axis=1)
Y_test = test_processed['Revenue']

print("Processed DataFrame for Training : Survived is the Target, other
columns are features.")
display(train_processed.head())
X_test = X_test.reset_index(drop=True)
X_test.head(5)
```

Build a simple random forest model in the data processed.

```
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, Y_train)
random_forest_preds = random_forest.predict(X_test)
print('The accuracy of the Random Forests model is :\t',metrics.accuracy_
score(random_forest_preds,Y_test))
```

Time to evaluate the model using LIME.

```
import lime
import lime.lime_tabular
predict_fn_rf = lambda x: random_forest.predict_proba(x).astype(float)
X = X_train.values
explainer = lime.lime_tabular.LimeTabularExplainer(X,feature_names = X_
train.columns,class_names=['FALSE','TRUE'],kernel_width=5)
```

Now let's choose a single instance and check the explanations for the prediction made for that instance.

```
choosen_instance = X_test.loc[[21]].values[0]
exp = explainer.explain_instance(choosen_instance, predict_fn_rf,num_
features=10)
exp.show_in_notebook(show_all=False)
```

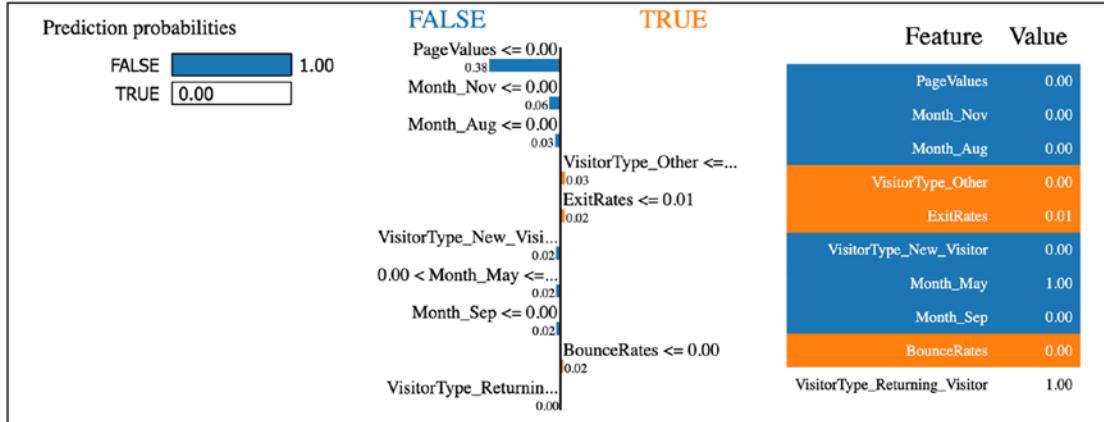


Figure 9-48. LIME output of a single instance

```
choosen_instance = X_test.loc[[31]].values[0]
exp = explainer.explain_instance(choosen_instance, predict_fn_rf,num_
features=10)
exp.show_in_notebook(show_all=False)
```

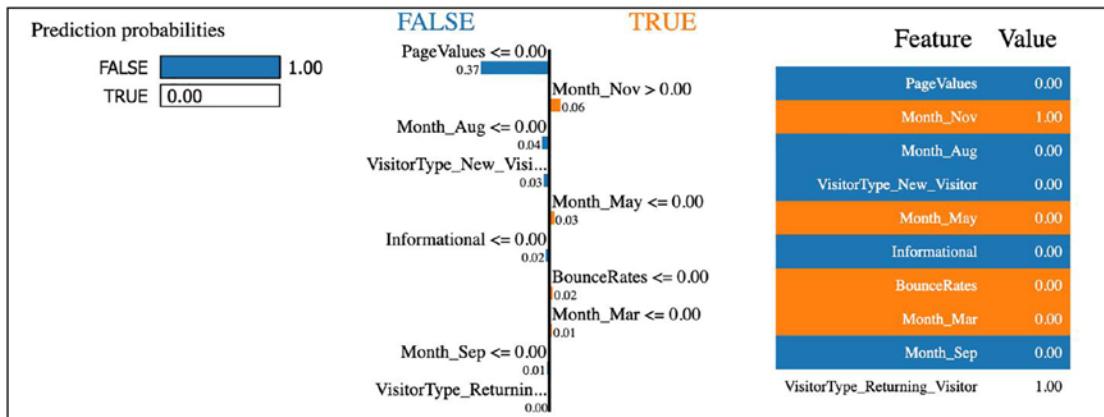


Figure 9-49. LIME output of a single instance

```
choosen_instance = X_test.loc[[1]].values[0]
exp = explainer.explain_instance(choosen_instance, predict_fn_rf,num_
features=10)
exp.show_in_notebook(show_all=False)
```

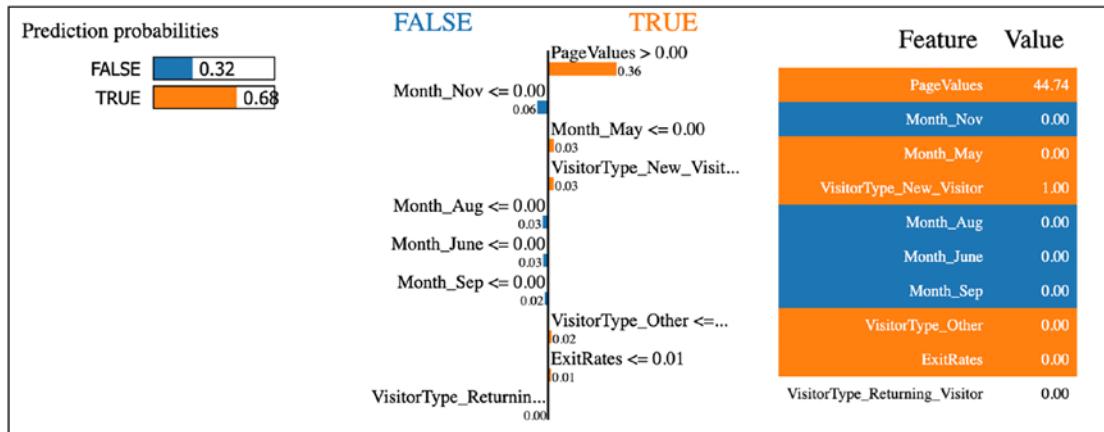


Figure 9-50. LIME output of a single instance

Figure 9-50 shows that PageValues was the most important variable for almost all the instances picked up for analysis. PageValues >0 points toward the fact that prediction is positive. For example, customers with high PageValues value are more likely to be online shoppers, which is understandable from general heuristics. For instances where the predicted value is False, PageValues are zero.

FACET

FACET is an open source library for human-explainable AI. It combines sophisticated model inspection and model-based simulation to enable better explanations of your supervised machine learning models.

FACET is composed of the following key components.

Model Inspection

FACET introduces a new algorithm to quantify dependencies and interactions between features in ML models. This new tool for human-explainable AI adds a new, global perspective to the observation-level explanations provided by the popular SHAP approach.

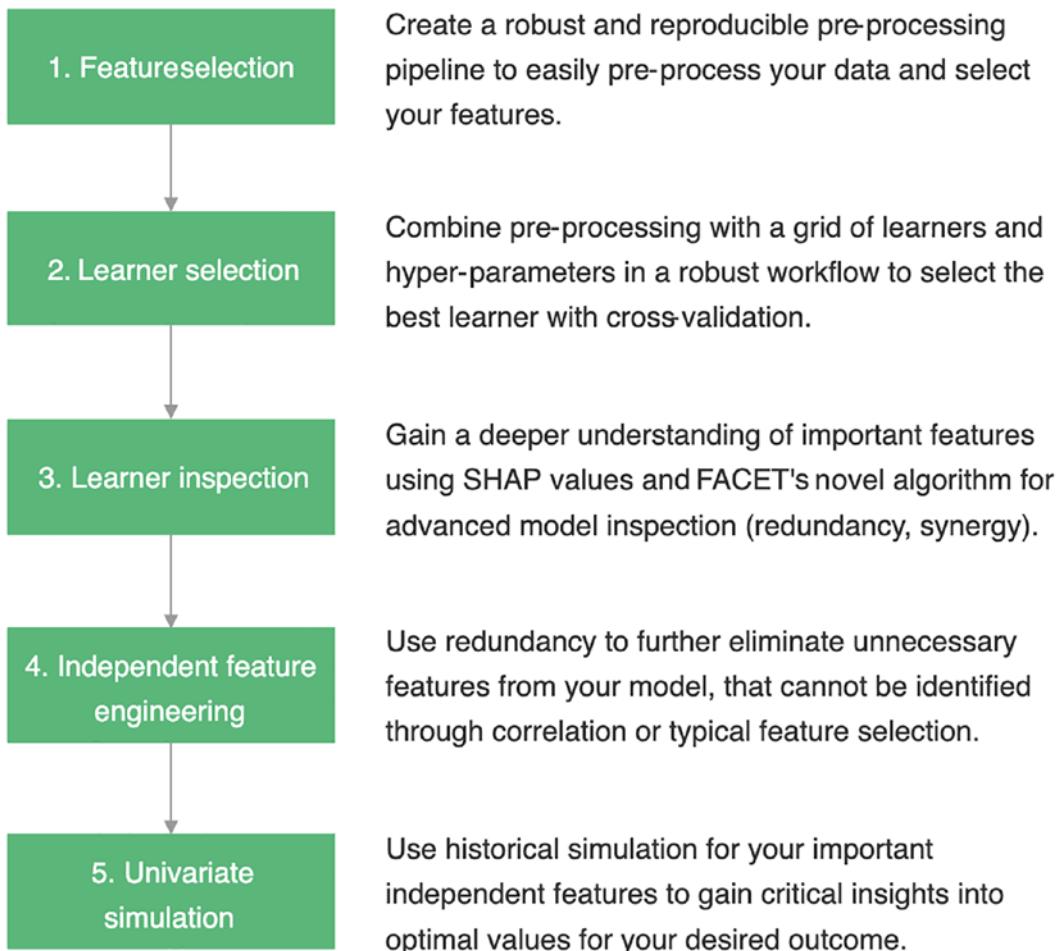
Model Simulation

FACET's model simulation algorithms use ML models for *virtual experiments* to identify scenarios that optimize predicted outcomes. FACET utilizes a range of bootstrapping algorithms to quantify the uncertainty in simulations, including stationary and stratified bootstraps.

Enhanced Machine Learning Workflow

FACET offers an efficient and transparent machine learning workflow, enhancing scikit-learn's tried and tested pipelining paradigm with new model selection, inspection, and simulation capabilities. FACET also introduces sklearndf an augmented version of scikit-learn with enhanced support for pandas data frames that ensures end-to-end traceability of features.

The following diagram is a high-level overview of the FACET workflow. Each step in the workflow includes a brief description.



Code

```
from sklearn.model_selection import RepeatedKFold

# some helpful imports from sklearndf
from sklearndf.pipeline import ClassifierPipelineDF
from sklearndf.classification import RandomForestClassifierDF
```

Let's start with relevant FACET imports.

```
from facet.data import Sample
from facet.selection import LearnerRanker, LearnerGrid

df=pd.read_csv('online_shoppers_intention.csv')

df1=df.copy()
df1=df1.values
df1=df1[:512,:]
df1=oe.fit_transform(df1)
df1=pd.DataFrame(df1)
df1.columns=df.columns
revenue_sample = Sample(observations=df1, target_name="Revenue")

# create a (trivial) pipeline for a random forest classifier
rnd_forest_clf = ClassifierPipelineDF(
    classifier=RandomForestClassifierDF(n_estimators=200, random_state=42)
)

# define grid of models which are "competing" against each other
rnd_forest_grid = [
    LearnerGrid(
        pipeline=rnd_forest_clf,
        learner_parameters={
            "min_samples_leaf": [8, 11, 15],
            "max_depth": [4, 5, 6],
        }
    ),
]

# create repeated k-fold CV iterator
rkf_cv = RepeatedKFold(n_splits=5, n_repeats=10, random_state=42)

# rank your candidate models by performance (default is mean CV
# score - 2*SD)
ranker = LearnerRanker(
    grids=rnd_forest_grid, cv=rkf_cv, n_jobs=-3).fit(sample=revenue_sample)
```

CHAPTER 9 FEATURE IMPORTANCE METHODS: DETAILS AND USAGE EXAMPLES

```
# Fit the model inspector
from facet.inspection import LearnerInspector

inspector = LearnerInspector(n_jobs=-3, verbose=2)
inspector.fit(crossfit=ranker.best_model_crossfit_)

f_importance = inspector.feature_importance()
plt.subplot(1, 2, 1)
f_importance.sort_values().plot.barh()

# get some info for standard SHAP model inspection
shap_data = inspector.shap_plot_data()

# standard SHAP summary plot using the shap package
plt.subplot(1, 2, 2)
shap.summary_plot(shap_values=shap_data.shap_values, features=shap_data.
features, show=False, plot_size=(16.0, 8.0))
plt.tight_layout()
```

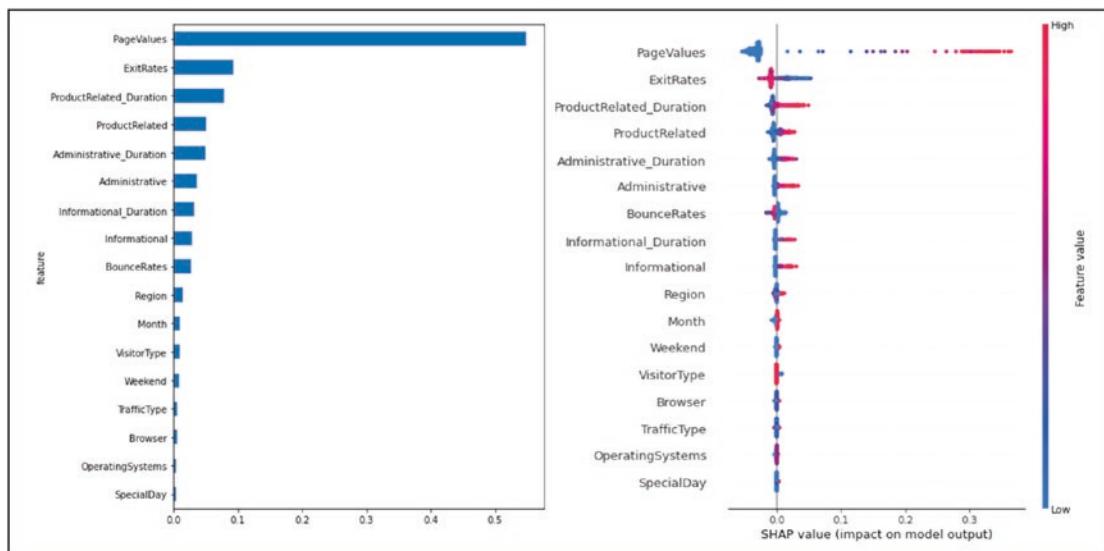


Figure 9-51. SHAP summary output using FACET

```
from pytools.viz.matrix import MatrixDrawer  
  
plt.figure(figsize=(10,8))  
synergy_matrix = inspector.feature_synergy_matrix()  
MatrixDrawer(style="matplotlib").draw(synergy_matrix, title="Synergy Matrix")
```

Synergy

The degree to which the model combines information from one feature with another to predict the target. For example, let's assume we are predicting cardiovascular health using age and gender and the fitted model includes a complex interaction between them. This means these two features are synergistic for predicting cardiovascular health. Further, both features are important to the model, and removing either one would significantly impact performance. Let's assume age brings more information to the joint contribution than gender. This asymmetric contribution means the synergy for (age, gender) is less than the synergy for (gender, age). To think about it another way, imagine the prediction is a coordinate you are trying to reach. From your starting point, age gets you much closer to this point than gender; however, you need both to get there. Synergy reflects that gender gets more help from age (higher synergy from the perspective of gender) than age does from gender (lower synergy from the perspective of age) to reach the prediction. *This leads to an important point: synergy is a naturally asymmetric property of the global information two interacting features contribute to the model predictions.* Synergy is expressed as a percentage ranging from 0% (full autonomy) to 100% (full synergy).

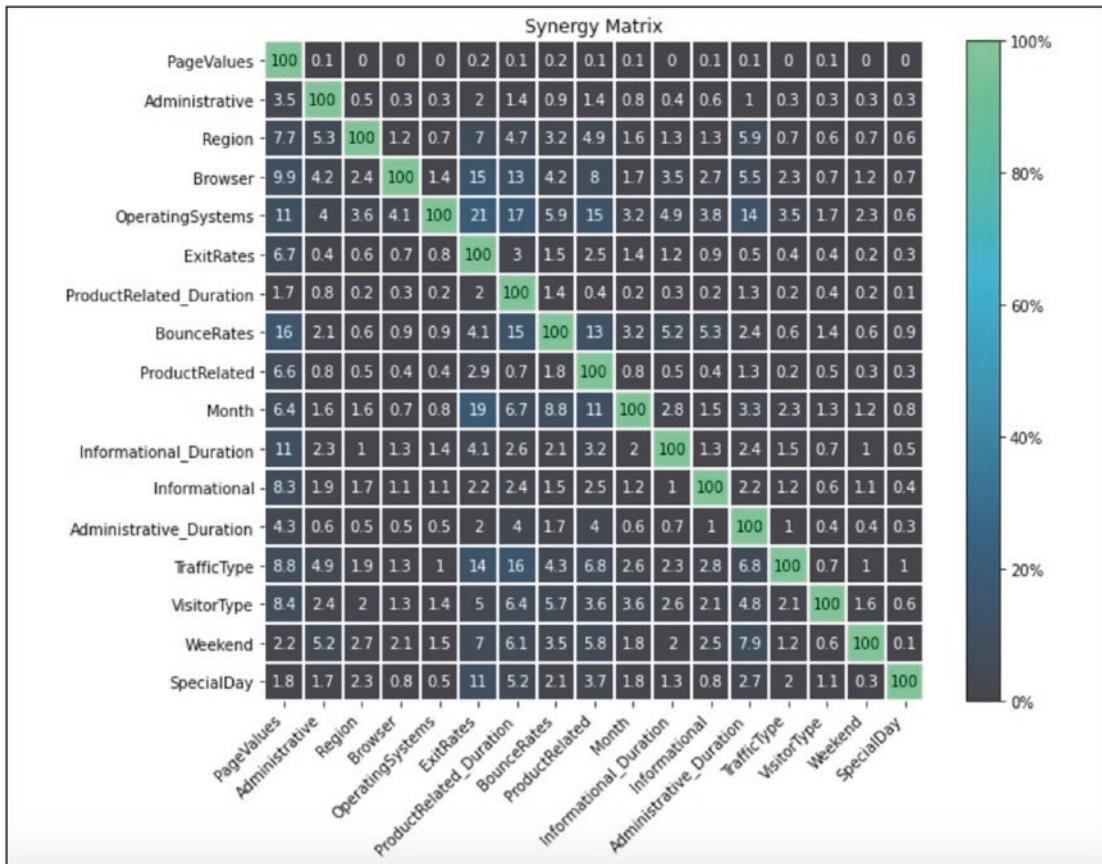


Figure 9-52. Synergy output of FACET

To interpret the synergy matrix, the first feature in a pair is the row (perspective from), and the second feature is the column. For example, for (BounceRates_to_PageValues), we find that 16% of the information is combined to predict prediabetes risk. This represents an example of little synergy between the feature pair from the perspective of BounceRates_to_PageValues.

Looking at the ExitRates we see from the perspective of other features, such as Month and Traffic type, shows values ranging up to 15%, suggesting a small amount of synergy. This is consistent with the idea that ExitRate is a strong independent feature (high feature importance) and that ExitRate partly enables traffic type and month in predicting shoppers.

```
plt.figure(figsize=(10,8))
redundancy_matrix = inspector.feature_redundancy_matrix()
MatrixDrawer(style="matplotlib").draw(redundancy_matrix, title="Redundancy Matrix")
```

Redundancy

Redundancy is the degree to which a model feature duplicates a second feature's information to predict the target. For example, let's assume we had house size and number of bedrooms for predicting house price. These features capture similar information as the more bedrooms, the larger the house, and likely a higher price on average. The redundancy for (number of bedrooms, house size) is greater than the redundancy for (house size, number of bedrooms). This is because house size "knows" more of what the number of bedrooms does for predicting house price than the reverse. Hence, there is greater redundancy from the perspective of the number of bedrooms. Another way to think about it is removing house size is more detrimental to model performance than removing the number of bedrooms, as house size can better compensate for the absence of the number of bedrooms. This also implies that house size would be a more important feature than the number of bedrooms in the model.

The important point here is that like synergy, redundancy is a naturally asymmetric property of the global information feature pairs have for predicting an outcome.

Redundancy is expressed as a percentage ranging from 0% (full uniqueness) to 100% (full redundancy).

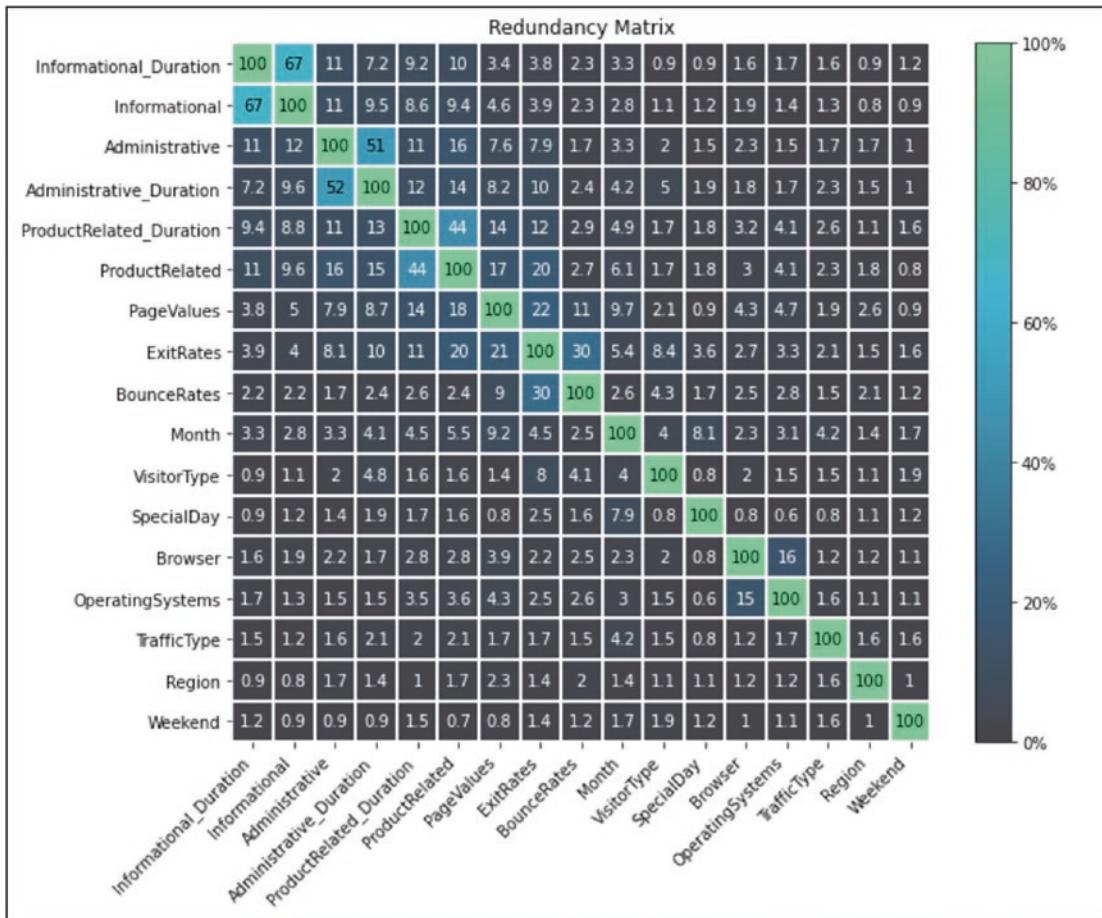
**Figure 9-53.** Redundancy output of FACET

Figure 9-53 shows that ExitRates and PageValues have a fair bit of redundancy. This means while taking action, we can invest our efforts in only one of these variables. Since every action we take after a model-building exercise comes with a cost associated with it. For example, if a model suggests that coupon price is an imp variable, the associated action can increase the value, which reduces cost. Hence redundancy narrows down the features that capture all the model information and reduce the number of actions we need to take based on model results.

```
from pytools.viz.dendrogram import DendrogramDrawer
plt.figure(figsize=(10,8))
redundancy = inspector.feature_redundancy_linkage()
DendrogramDrawer().draw(data=redundancy, title="Redundancy Dendrogram")
```

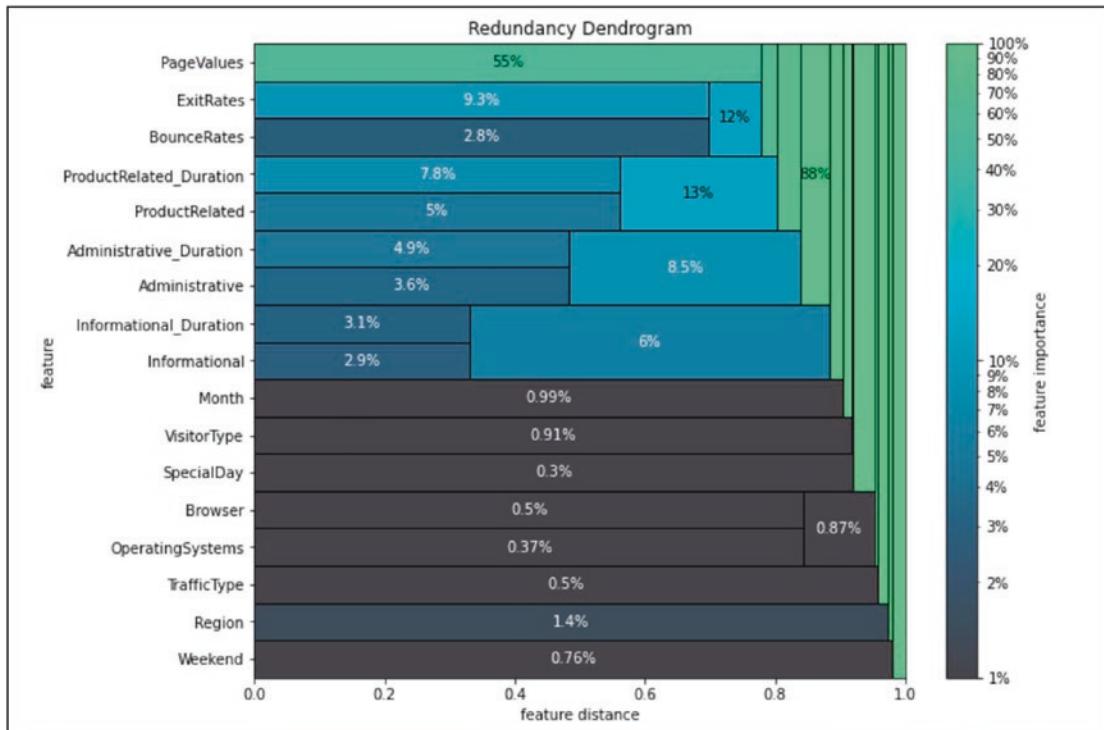


Figure 9-54. Feature clustering to group similar importance features together

Figure 9-54 shows that redundancy and synergy for a feature pair are from the perspective of one of the features in the pair, and so yields two distinct values. However, a symmetric version can also be computed that provides a simplified perspective and allows the use of (1 metric) as a feature distance. With this distance hierarchical, single linkage clustering is applied to create a dendrogram visualization. This identifies groups of low-distance features that activate in tandem to predict the outcome. Such information can then be used to reduce clusters of highly redundant features to a subset or highlight clusters of highly synergistic features that should always be considered together.

For this example, let's apply clustering to redundancy to see how the apparent grouping observed in the heatmap appears in the dendrogram. Ideally, we want to see features only start to cluster as close to the right-hand side of the dendrogram as possible. This implies all features in the model are contributing uniquely to our predictions.

Partial Dependence Plots (PDP)

The partial dependence plot (short PDP or PD plot) shows the marginal effect one or two features have on the predicted outcome of a machine learning model (J. H. Friedman 2001). A partial dependence plot can show whether the relationship between the target and a feature is linear, monotonic, or more complex. For example, partial dependence plots always show a linear relationship when applied to a linear regression model.

The partial dependence function for regression is defined as follows.

$$\hat{f}_{x_S}(x_S) = E_{x_C} \left[\hat{f}(x_S, x_C) \right] = \int \hat{f}(x_S, x_C) d\mathbb{P}(x_C)$$

The x_S are the features for which the partial dependence function should be plotted, and x_C are the other features used in the machine learning model f . Usually, there are only one or two features in the set S . The feature(s) in S are those we want to know the effect on the prediction. The feature vectors x_S and x_C combined make up the total feature space x . Partial dependence works by marginalizing the machine learning model output over the distribution of the features in set C . The function shows the relationship between the features in set S we are interested in and the predicted outcome.

Marginalizing over the other features gets a function that depends only on features in S , interactions with other features included.

The partial function \hat{f}_{x_S} is estimated by calculating averages in the training data, also known as the *Monte Carlo method*.

$$\hat{f}_{x_S}(x_S) = \frac{1}{n} \sum_{i=1}^n \hat{f}(x_S, x_C^{(i)})$$

The partial function states for a given value(s) of features S the average marginal effect on the prediction. In this formula, $x_{C(i)}$ are actual feature values from the data set for the features we are not interested in, and n is the number of instances in the data set. The PDP assumes that the features in C are not correlated with the features in S. If this assumption is violated, the averages calculated for the partial dependence plot include data points that are very unlikely or even impossible.

For classification where the machine learning model outputs probabilities, the partial dependence plot displays the probability for a certain class given different values for feature(s) in S. An easy way to deal with multiple classes is to draw one line or plot per class.

The partial dependence plot is a global method: The method considers all instances and gives a statement about the global relationship of a feature with the predicted outcome.

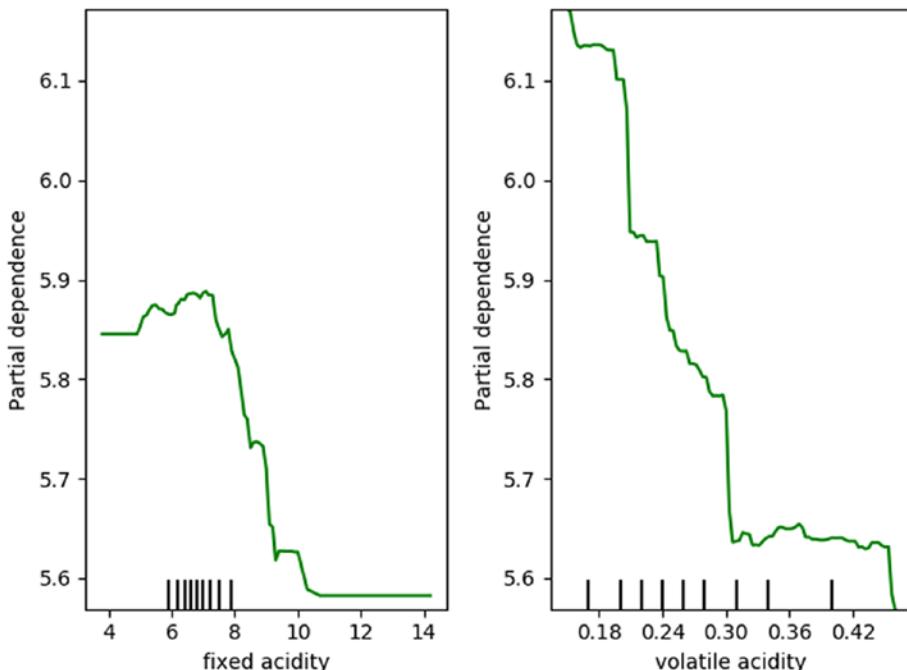


Figure 9-55. Example plots of feature dependence

The left plot in Figure 9-55 shows the partial dependence between the target—the sales price—and the distance variable. Distance in this data set measures the distance to Melbourne's central business district.

The partial dependence plot is calculated only after the model has been fit. The model is fit on real data. In that real data, houses in different parts of town may differ in myriad ways (different ages, sizes, etc.) But after the model is fit, we could start by taking all the characteristics of a single house—let's say it's a ten-year-old house with two bedrooms, two bathrooms, on a large lot.

We then use the model to predict that house's price, but we change the distance variable before making a prediction. We first predict the price for that house when sitting distance to 4. We then predict its price-setting distance to 5. Then predict again for 6. And so on. We trace out how predicted price changes (on the vertical axis) as we move from small values of distance to large values (on the horizontal axis).

This description used only a single house. But because of interactions, the partial dependence plot for a single house may be atypical. Instead, we repeat that mental experiment with multiple houses and plot the average predicted price on the vertical axis. You see some negative numbers. That doesn't mean the price would sell for a negative price. Instead, it means the prices would have been less than the actual average price for that distance.

These plots are useful to extract insights and check that your model is learning something you think is sensible.

So far, we have only considered numerical features. For categorical features, the partial dependence is very easy to calculate. Each of the categories gets a PDP estimate by forcing all data instances to have the same category. For example, if you look at the bike rental data set and are interested in the partial dependence plot for the season, there are four numbers, one for each season. To compute the value for summer, we replace the season of all data instances with *summer* and average the predictions.

The computation of partial dependence plots is *intuitive*. If we force all data points to assume that feature value, the partial dependence function at a particular feature value represents the average prediction. In my experience, lay people usually understand the idea of PDPs quickly.

If the feature for which you computed the PDP is not correlated with the other features, then the PDPs perfectly represent how the feature influences the prediction on average. In the uncorrelated case, the *interpretation is clear*. The partial dependence plot shows how the average prediction in your data set changes when the j-th feature is changed. It is more complicated when features are correlated.

Partial dependence plots are *easy to implement*.

The calculation for the partial dependence plots has a *causal interpretation*. We intervene on a feature and measure the changes in the predictions. In doing so, we analyze the causal relationship between the feature and the prediction.²⁸ The relationship is causal for the model because we explicitly model the outcome as a function of the features—but not necessarily in the real world!

The realistic *maximum number of features* in a partial dependence function is two. This is not the fault of PDPs but the 2-dimensional representation (paper or screen) and our inability to imagine more than three dimensions.

Some PD plots do not show the *feature distribution*. Omitting the distribution can be misleading because you might overinterpret regions with almost no data. This problem is easily solved by showing a rug (indicators for data points on the x axis) or a histogram.

The *assumption of independence* is the biggest issue with PD plots. It is assumed that the feature(s) for which the partial dependence is computed is not correlated with other features. For example, suppose you want to predict how fast a person walks, given the person's weight and height. For the partial dependence of one of the features (e.g., height), we assume that the other features (weight) are not correlated with height, which is false. For the computation of the PDP at a certain height (e.g., 200 cm), we average over the marginal distribution of weight, which might include a weight below 50 kg, which is unrealistic for a person two meters tall. In other words: When the features are correlated, we create new data points in areas of the feature distribution where the actual probability is very low (for example, it is unlikely that someone is two meters tall but weighs less than 50 kg). One solution is *accumulated local effect* (ALE) plots that work with the conditional instead of the marginal distribution.

Heterogeneous effects might be hidden because PD plots only show the average marginal effects. Suppose that for a feature. Half your data points positively affect the prediction: the larger the feature value, the larger the prediction. The other half has a negative association: the smaller the feature value, the larger the prediction. The PD curve could be a horizontal line since the effects of both halves of the data set could cancel each other out. You then conclude that the feature does not affect the prediction. By plotting the individual conditional expectation curves instead of the aggregated line, you can uncover heterogeneous effects.

Code

```
import pandas as pd
from sklearn.ensemble import GradientBoostingRegressor,
GradientBoostingClassifier
from sklearn.ensemble.partial_dependence import partial_dependence, plot_
partial_dependence
from sklearn.impute import SimpleImputer

df = pd.read_csv('online_shoppers_intention.csv')

data1 = pd.get_dummies(df)
data1.columns
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
df['Revenue'] = le.fit_transform(df['Revenue'])
df['Revenue'].value_counts()
x = data1
# removing the target column revenue from x
x = x.drop(['Revenue'], axis = 1)

y = df['Revenue']

# checking the shapes
print("Shape of x:", x.shape)
print("Shape of y:", y.shape)

clf = GradientBoostingClassifier()
my_imputer = SimpleImputer()
imputed_x = my_imputer.fit_transform(x)

clf.fit(imputed_x,y)
plots = plot_partial_dependence(clf, features=['ExitRates', 'PageValues'],
X=imputed_x, feature_names=['ExitRates', 'PageValues'], grid_resolution=4)
```

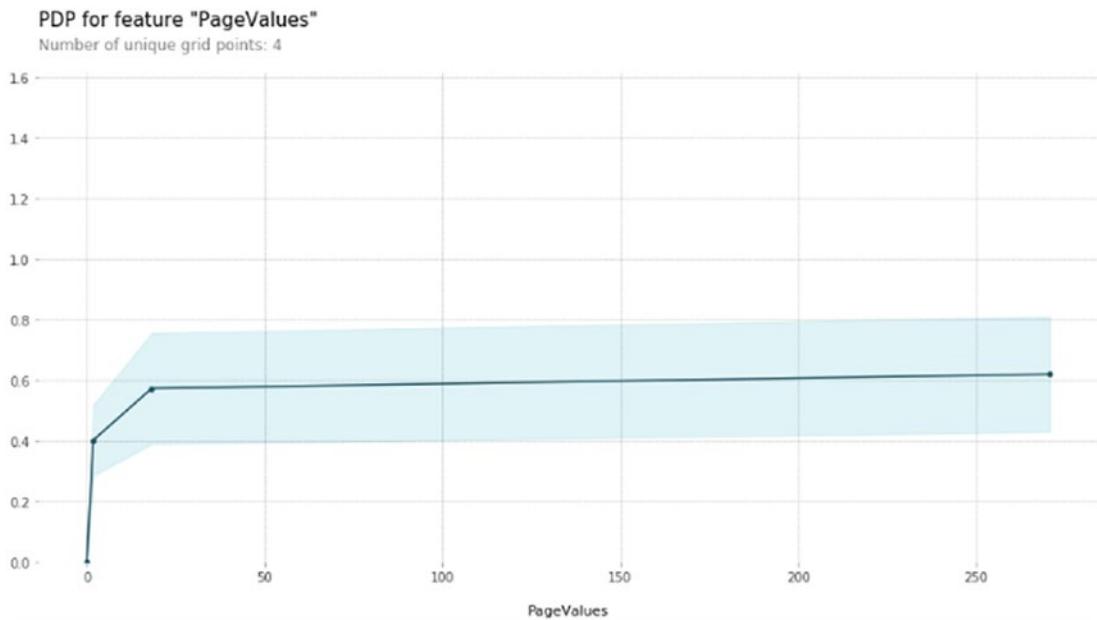


Figure 9-56. PDP plots for top variables

Individual Conditional Expectation

Individual conditional expectation (ICE) plots display one line per instance that shows how the instance's prediction changes when a feature changes.

The partial dependence plot for the average effect of a feature is a global method because it does not focus on specific instances but an overall average. The equivalent to a PDP for individual data instances is an ICE plot. An ICE plot visualizes the dependence of the prediction on a feature for *each* instance separately, resulting in one line per instance, compared to one line overall in partial dependence plots. A PDP is the average of the lines of an ICE plot.

The values for a line (and one instance) can be computed by keeping all other features the same, creating variants of this instance by replacing the feature's value with values from a grid, and making predictions with the black-box model for these newly created instances.

The result is a set of points with the feature value from the grid and the respective predictions. What is the point of looking at individual expectations instead of partial dependencies? Partial dependence plots can obscure a heterogeneous relationship created by interactions. PDPs can show you what the average relationship between a feature and the prediction looks like. This only works well if the interactions between the features for which the PDP is calculated and the other features are weak. In interactions, the ICE plot provides much more insight.

DALEX

The explainability of predictive models is often linked with methods such as LIME, SHAP, or PDP, which help you better understand local or global model behavior. These techniques respond to specific needs for understanding models as a whole or understanding predictions for specific instances. To avoid getting lost in the variety of different explainability techniques, several approaches to cataloging are proposed (local/global, model agnostic/specific, case-based/rule-based/profile-based, and so on). We tried a few in the last two years. DALEX puts together the most popular techniques into the XAI pyramid presented in Figure 9-57 (a triangular version of the periodic table).

The model exploration for an individual instance starts with a single number — a prediction. This is the top level of the pyramid. We want to assign particular variables to this prediction to understand which are important and how strongly they influence this particular prediction. One can use methods as SHAP, LIME, Break Down, Break Down with interactions. This is the second from the top level of the pyramid. Moving down, the next level is related to the model's sensitivity to change one or more variables' values. *Ceteris paribus* (CP) profiles allow the exploration of the conditional behavior of the model.

Going further, you can investigate how good is the local fit of the model. The model may be very good on average, but for the selected observation, the local fit is very low, errors/residuals are larger than average. The pyramid can be further extended by adding interactions of variable pairs.

The exploration for the whole model starts with an assessment of the quality of the model, either with F1, MSE, AUC, or LIFT/ROC curves. Such information tells how good the model is in general. The next level helps you understand which variables are important and which ones make the model work or not. A common technique is the

permutation importance of variables. Moving down, methods on the next level help you understand the model's response profile as a function of certain variables. Here you can use such techniques as partial dependence profiles or accumulated local dependence.

Going further, there are more detailed analyses related to the diagnosis of the errors/residuals.

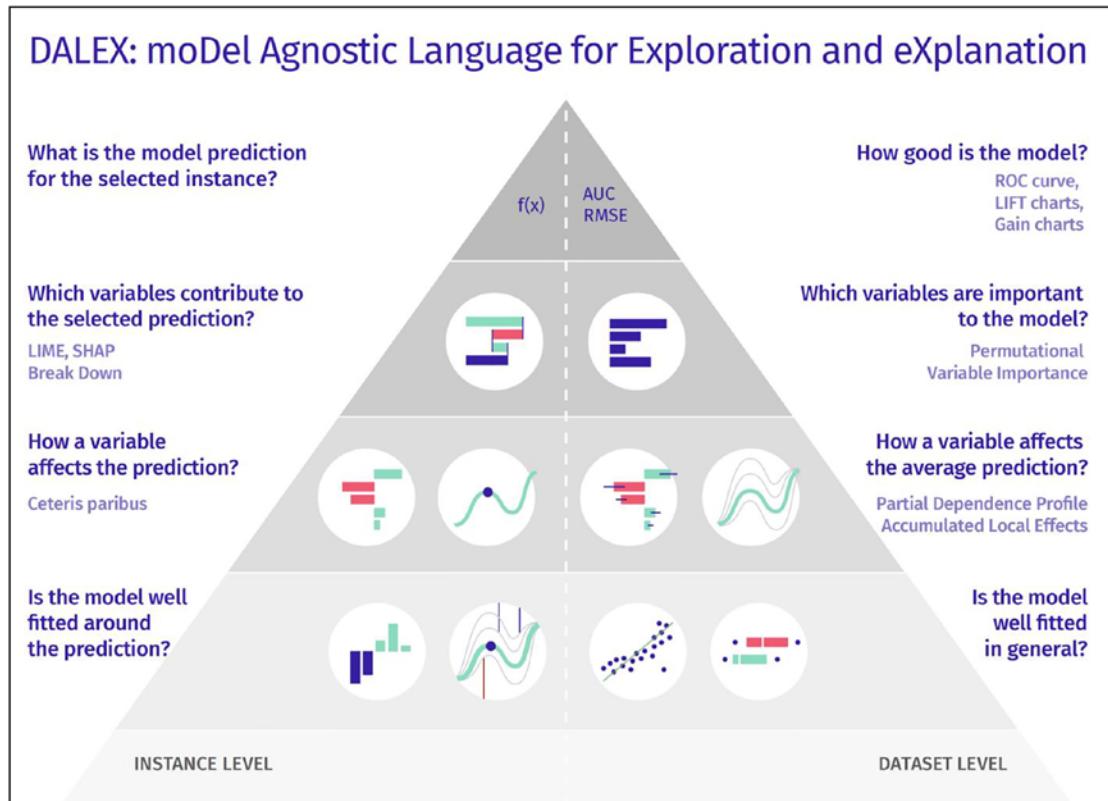


Figure 9-57. In the pyramid, needs related to exploring predictive models are gathered into an extensible drill-down map. The left side is about needs related to a single instance, right side to a model as a whole. Consecutive layers dig into more and more detailed questions about the model behavior (local or global)

The first version of the DALEX package was released in 2018. During this time, the architecture has been continuously improved. The final approach is implemented in version 1.0. Each function is responsible for a single part of the pyramid. If a given part can be calculated in several ways, the argument type can specify the desired method.

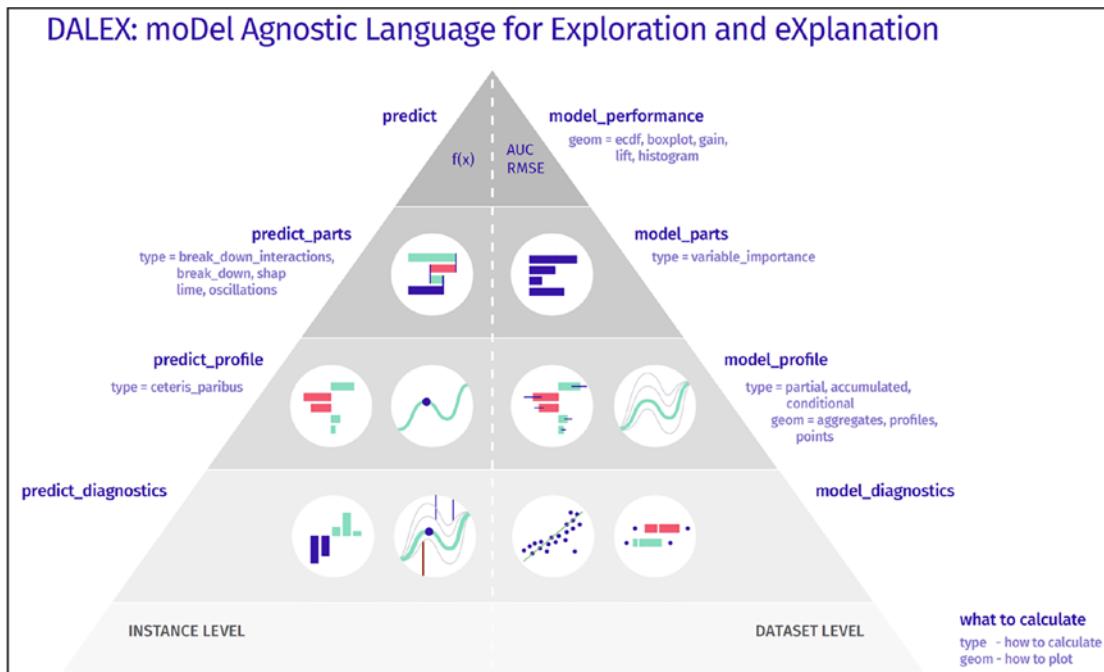


Figure 9-58. Implementation of the explainability pyramid in the DALEX package. Dark violet names are names of functions that implement methods for particular needs. Light violet names stand for implemented methods that address selected needs

The functions available in the DALEX package are the backbone on which many tools for exploring machine learning models are built. The following are among the latest.

- DALEXtra with connectors to popular ML frameworks like scikit-learn, H2O, MLR, Caret, TensorFlow, and Keras
- modelDown builds a static HTML website with a summary of data set level model exploration
- modelStudio builds an interactive serverless D3js based website with a dashboard of complementary model views
- ingredients, iBreakDown, an auditor with specific techniques for model explorations.

Introduction to Instance-level Exploration

Instance-level exploration methods help you understand how a model yields a prediction for a particular single observation. Consider the following situations as examples.

- We may want to evaluate the effects of explanatory variables on the model's predictions. For instance, we may be interested in predicting the risk of heart attack based on a person's age, sex, and smoking habits. A model may construct a score (for instance, a linear combination of the explanatory variables representing age, sex, and smoking habits) that could be used for prediction. For a particular patient, we may want to learn how much the different variables contribute to the score?
- We may want to understand how the model's predictions would change if some of the explanatory variables changed? For instance, what would be the predicted risk of heart attack if the patient cut the number of cigarettes smoked per day by half?
- We may discover that the model is providing incorrect predictions, and we may want to find the reason. For instance, a patient with a very low-risk score experienced a heart attack. What has driven the wrong prediction?

Breakdown Plots for Additive Attributions

Breakdown plots show how the contributions attributed to individual explanatory variables change the mean model's prediction to yield the actual prediction for a particular single instance (observation).

- Panel A: The first row shows the distribution and the mean value (red dot) of the model's predictions for all data. The next rows show the distribution and the mean value of the predictions when fixing values of subsequent explanatory variables. The last row shows the prediction for the instance of interest.
- Panel B: Red dots indicate the mean predictions from panel A.

- Panel C: The green and red bars indicate, respectively, positive and negative changes in the mean predictions (contributions attributed to explanatory variables), as shown in Figure 9-59.

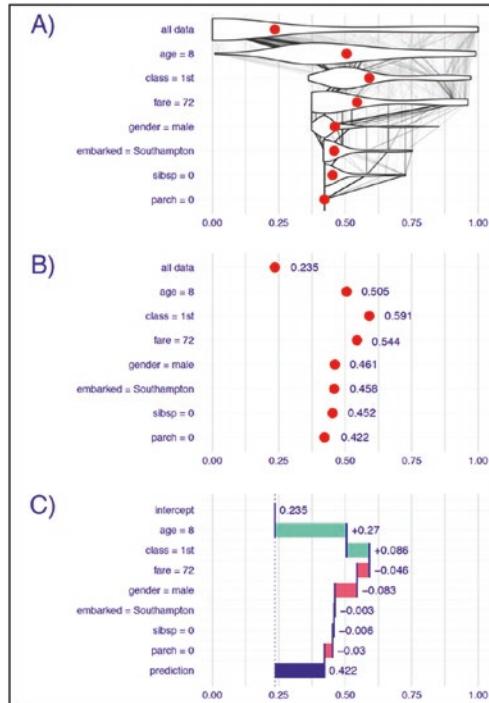


Figure 9-59. DALEX breakdown plots

To evaluate the contribution of individual explanatory variables to this single-instance prediction, we investigate the changes in the model's predictions when fixing the values of consecutive variables. For instance, the violin plot in the row marked “age=8” in panel A of Figure 9-59 summarizes the distribution of the predictions obtained when the *age* variable takes the value “8 years”, as for Johnny D. Again, the red dot indicates the mean of the predictions. It can be interpreted as estimating the expected value of the predictions over the distribution of all explanatory variables other than *age*.

The violin plot in the “class=1st” row describes the distribution and the mean value of predictions with the values of variables *age* and *class* set to “8 years” and “1st class”, respectively. Subsequent rows contain similar information for other explanatory variables included in the random forest model. In the last row, all explanatory variables

are fixed at the values describing Johnny D. Hence, the last row contains only one point, the red dot, which corresponds to the model's prediction, such as survival probability, for Johnny D.

The thin gray lines in panel A of Figure 9-59 show the change of predictions for different individuals when the value of a particular explanatory variable is replaced by the value indicated in the row's name. For instance, the lines between the first and the second row indicate that fixing the value of the age variable to "8 years" has a different effect for different individuals. For some individuals (most likely, passengers that are eight years old), the model's prediction does not change at all. For others, the predicted value increases (probably for the passengers older than eight years) or decreases (most likely for the passengers younger than eight years).

Breakdown Plots for Interactions

Interaction (deviation from additivity) means that the effect of an explanatory variable depends on the value(s) of other variables (see Figure 9-60).

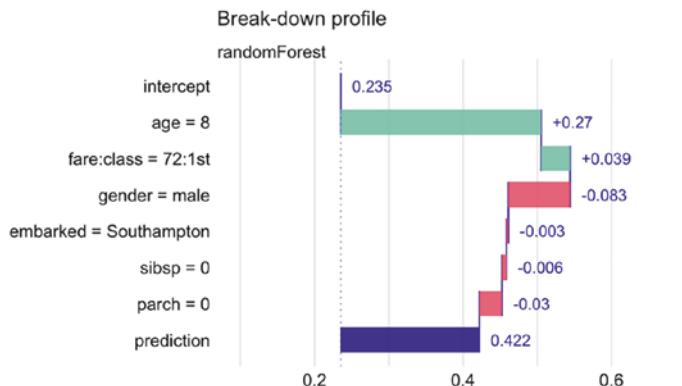


Figure 9-60. Breakdown plot with interaction

The interaction between *fare* and *class* variables is included in the plot as a single bar. As the effects of these two variables cannot be disentangled, the plot uses that single bar to represent the contribution of both variables.

Interaction breakdown plots (iBD) share many advantages and disadvantages of BD plots for models without interactions. However, in models with interactions, iBD plots provide more correct explanations. Though the numerical complexity of the iBD procedure is quadratic, it may be time-consuming in models with a large number of explanatory variables. For data sets with a small number of observations, the calculations of the net contributions are subject to a larger variability and, therefore, larger randomness in the ranking of the contributions. It is also worth noting that the presented procedure of identification of interactions is not based on any formal statistical-significance test. Thus, the procedure may lead to false-positive findings and, especially for small sample sizes, false-negative errors.

Ceteris Paribus Profiles

This feature of DALEX evaluates the effect of a selected explanatory variable in changes of a model's prediction induced by changes in the variable's values. The method is based on the *ceteris paribus* principle. *Ceteris paribus* is a Latin phrase meaning "other things held constant" or "all else unchanged." The method examines the influence of an explanatory variable by assuming that the values of all other variables do not change. The main goal is to understand how changes in the values of the variable affect the model's predictions.

CP profiles show how a model's prediction would change if the value of a single exploratory variable changed. In essence, a CP profile shows the dependence of the conditional expectation of the dependent variable (response) on the values of the particular explanatory variable.

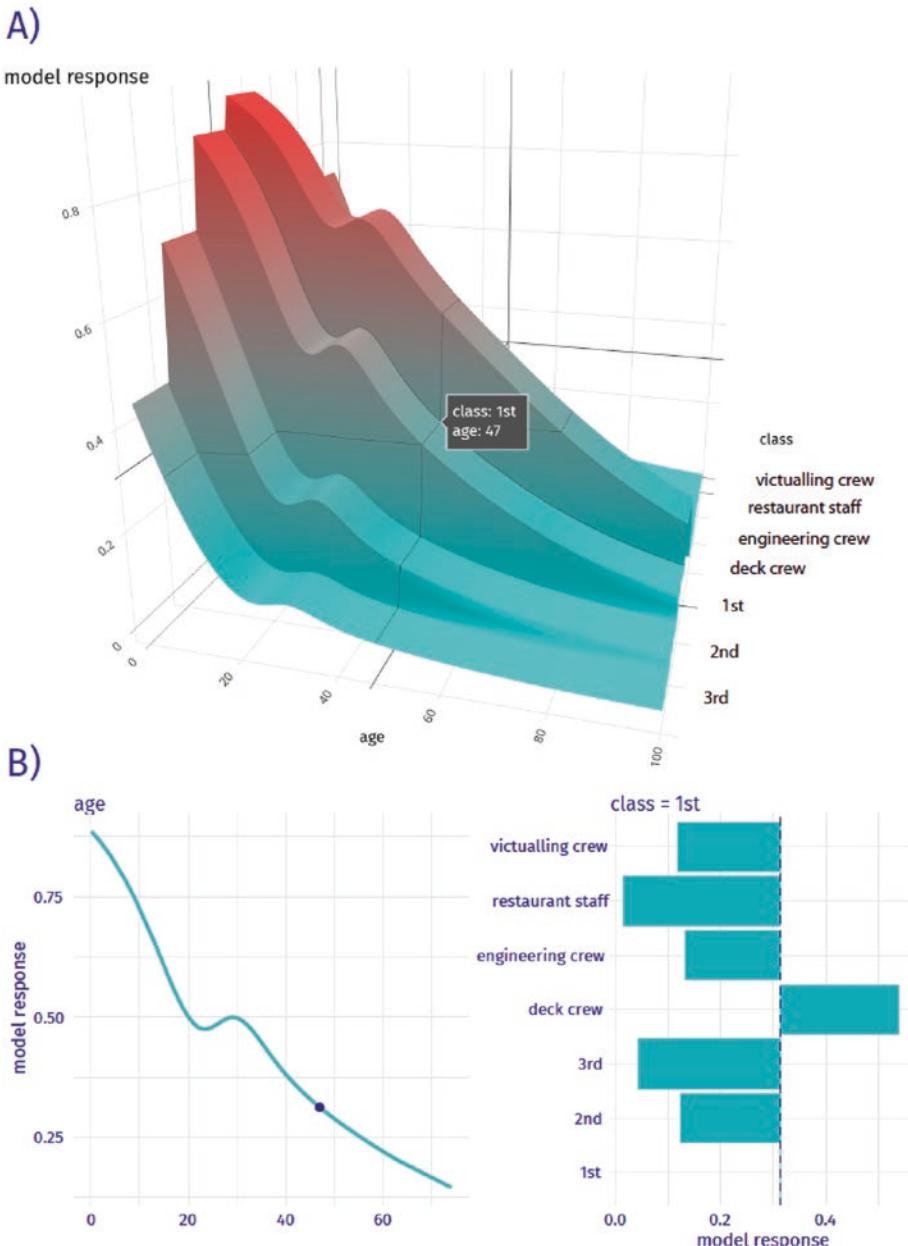


Figure 9-61. Panel A shows the model response (prediction) surface for variables age and class. CP profiles are conditional, one-dimensional plots that are marked with black curves. They help you understand the changes in the curvature of the surface induced by changes in only a single explanatory variable. Panel B shows CP profiles for individual variables, age (continuous), and class (categorical)

Local Diagnostics Plots

The basic idea behind local-fidelity plots is to compare the distribution of residuals (i.e., differences between the observed and predicted value of the dependent variable) for the neighbors with the distribution of residuals for the entire training data set. The idea behind local-stability plots is to check whether small changes in the explanatory variables, as represented by the changes within the set of neighbors, have much influence on the predictions.

Apart from this, DALEX implements SHAP and LIME when it comes to generating instance-level explanations.

Implementation Example of DALEX on the *Titanic* Data Set

The following is a summary of the *Titanic* data set.

- pclass: Passenger class (1 = 1st; 2 = 2nd; 3 = 3rd)
- survival: A Boolean indicating whether the passenger survived or not (0 = No; 1 = Yes); this is the target
- name: A field rich in information as it contains title and family names
- sex: Male/Female
- age: Age; a significant portion of values are missing
- sibsp: The number of siblings/spouses aboard
- parch: The number of parents/children aboard
- ticket: Ticket number
- fare: Passenger fare (British pound)
- cabin: Does the location of the cabin influence chances of survival?
- embarked: Port of embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)
- boat: Lifeboat, many missing values
- body: Body identification number
- home.dest: Home/destination

There are 1,309 records and 14 attributes.

```
import dalex as dx

import pandas as pd
import numpy as np

from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

import warnings
warnings.filterwarnings('ignore')

data = dx.datasets.load_titanic()

X = data.drop(columns='survived')
y = data.survived

data.head(10)
```

	gender	age	class	embarked	fare	sibsp	parch	survived
0	male	42.0	3rd	Southampton	7.1100	0	0	0
1	male	13.0	3rd	Southampton	20.0500	0	2	0
2	male	16.0	3rd	Southampton	20.0500	1	1	0
3	female	39.0	3rd	Southampton	20.0500	1	1	1
4	female	16.0	3rd	Southampton	7.1300	0	0	1
5	male	25.0	3rd	Southampton	7.1300	0	0	1
6	male	30.0	2nd	Cherbourg	24.0000	1	0	0
7	female	28.0	2nd	Cherbourg	24.0000	1	0	1
8	male	27.0	3rd	Cherbourg	18.1509	0	0	1
9	male	20.0	3rd	Southampton	7.1806	0	0	1

Figure 9-62. sample data set taken for this exercise of DALEX

Create a Pipeline Model

1. Start with the numerical_transformer pipeline.
The following instructs numerical_features.
 - a. Choose numerical features to transform.
 - b. Impute missing data with median strategy.
 - c. Scale numerical features with standard scaler.
2. Next, the categorical_transformer pipeline. The following instructs categorical_features.
 - d. Choose categorical features to transform.
 - e. Impute missing data with 'missing' string.
 - f. Encode categorical features with one-hot.
3. Aggregate these two pipelines into a preprocessor using ColumnTransformer.
4. Make a basic classifier model using MLPClassifier. It has three hidden layers with sizes 150, 100, and 50 respectively.
5. Construct a CLF pipeline model, which combines the preprocessor with the basic classifier model.

```
numerical_features = ['age', 'fare', 'sibsp', 'parch']
numerical_transformer = Pipeline(
    steps=[
        ('imputer', SimpleImputer(strategy='median')),
        ('scaler', StandardScaler())
    ]
)
```

```

categorical_features = ['gender', 'class', 'embarked']
categorical_transformer = Pipeline(
    steps=[
        ('imputer', SimpleImputer(strategy='constant', fill_
            value='missing')),
        ('onehot', OneHotEncoder(handle_unknown='ignore'))
    ]
)
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ]
)
classifier = MLPClassifier(hidden_layer_sizes=(150,100,50), max_iter=500,
random_state=0)

clf = Pipeline(steps=[('preprocessor', preprocessor),
                     ('classifier', classifier)])

clf.fit(X, y)

exp = dx.Explainer(clf, X, y)

Preparation of a new explainer is initiated

-> data           : 2207 rows 7 cols
-> target variable : Parameter 'y' was a pandas.Series. Converted to a numpy.ndarray.
-> target variable : 2207 values
-> model_class    : sklearn.neural_network._multilayer_perceptron.MLPClassifier (default)
-> label          : Not specified, model's class short name will be used. (default)
-> predict function: <function yhat_proba_default at 0x000002AEFC996700> will be used (default)
-> predict function: Accepts only pandas.DataFrame, numpy.ndarray causes problems.
-> predicted values: min = 2.72e-06, mean = 0.337, max = 1.0
-> model type     : classification will be used (default)
-> residual function: difference between y and yhat (default)
-> residuals      : min = -0.921, mean = -0.0146, max = 0.975
-> model_info      : package sklearn

A new explainer has been created!

```

Figure 9-63. Sample output of DALEX function

The following functionalities are available with DALEX.

- Explainer—a uniform abstraction over predictive models
- Consistent grammar for model analysis
- Predict-level explanations
- Model-level explanations
- Fairness checks
- Arena: Interactive comparative model analysis

These functionalities are accessible from the Explainer object through its methods.

Model-level and predict-level methods return a new unique object that contains the result attribute (pandas.DataFrame) and the plot method.

Predict-level Explanations

predict

This function is a normal model prediction; however, it uses the Explainer interface.

Let's create two example persons for this tutorial.

```
john = pd.DataFrame({'gender': ['male'],
                     'age': [25],
                     'class': ['1st'],
                     'embarked': ['southampton'],
                     'fare': [72],
                     'sibsp': [0],
                     'parch': 0},
                     index = ['John'])

mary = pd.DataFrame({'gender': ['female'],
                     'age': [35],
                     'class': ['3rd'],
                     'embarked': ['Cherbourg'],
                     'fare': [25],
                     'sibsp': [0],
                     'parch': [0]},
                     index = ['Mary'])
```

You can predict many samples at the same time.

```
exp.predict(X)[0:10]
```

```
array([0.07907226, 0.20628711, 0.13463174, 0.60372994, 0.76485216,
       0.16150944, 0.03705073, 0.99324938, 0.19563509, 0.12184964])
```

As well as on only one instance. However, the only accepted format is pandas. DataFrame.

The following is the survival prediction for John.

```
exp.predict(john)
```

The following is the survival prediction for Mary.

```
exp.predict(mary)
```

predict_parts

- 'break_down'
- 'break_down_interactions'
- 'shap'

This function calculates variable attributions as breakdown, breakdown interactions, or Shapley values explanations.

Model prediction is decomposed into parts that are attributed to particular variables.

Breakdown plots show how the contributions attributed to individual explanatory variables change the mean model's prediction to yield the actual prediction for a particular instance (observation).

- Panel A: The first row shows the distribution and the mean value (red dot) of the model's predictions for all data. The next rows show the distribution and the mean value of the predictions when fixing values of subsequent explanatory variables. The last row shows the prediction for the particular instance of interest.
- Panel B: Red dots indicate the mean predictions from panel A.

CHAPTER 9 FEATURE IMPORTANCE METHODS: DETAILS AND USAGE EXAMPLES

- Panel C: The green and red bars indicate, respectively, positive, and negative changes in the mean predictions (contributions attributed to explanatory variables). Figure 9-63 can be explained using the theory.

```
bd_john = exp.predict_parts(john, type='break_down', label=john.index[0])
bd_interactions_john = exp.predict_parts(john, type='break_down_
interactions', label="John+")

sh_mary = exp.predict_parts(mary, type='shap', B = 10, label=mary.index[0])
bd_john.result
```

Figure 9-64 shows the model summary. It includes the intercept of the model and the contribution or coefficient of each feature.

	variable_name	variable_value	variable	cumulative	contribution	sign	position	label
0	intercept	1	intercept	0.336735	0.336735	1.0	8	John
1	class	1st	class = 1st	0.583093	0.246358	1.0	7	John
2	age	25.0	age = 25.0	0.595401	0.012308	1.0	6	John
3	sibsp	0.0	sibsp = 0.0	0.585751	-0.009650	-1.0	5	John
4	fare	72.0	fare = 72.0	0.319029	-0.266722	-1.0	4	John
5	parch	0.0	parch = 0.0	0.300772	-0.018257	-1.0	3	John
6	embarked	Southampton	embarked = Southampton	0.284191	-0.016580	-1.0	2	John
7	gender	male	gender = male	0.081277	-0.202914	-1.0	1	John
8			prediction	0.081277	0.081277	1.0	0	John

Figure 9-64. result for the model

```
bd_john.plot(bd_interactions_john)
```

Figure 9-65 shows the contribution of different variables for an instance of passenger John. It looks as if class is the most important variable for this passenger. The chances for the survival of this passenger are low.

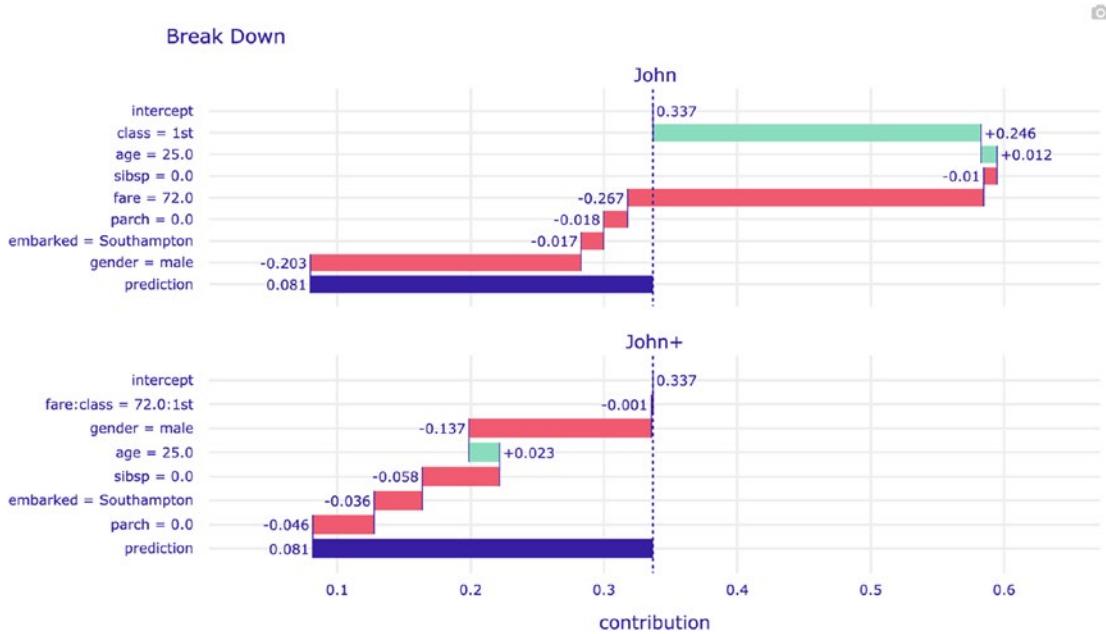


Figure 9-65. prediction breakdown for John

Figure 9-66 shows the contribution for each variable for passenger Mary. The gender= Female dummy variable has the highest contribution.

```
sh_mary.result.loc[sh_mary.result.B == 0, ]
```

	variable	contribution	variable_name	variable_value	sign	label	B
0	gender = female	0.480023	gender	female	1.0	Mary	0
1	embarked = Cherbourg	0.159716	embarked	Cherbourg	1.0	Mary	0
2	class = 3rd	-0.103756	class	3rd	-1.0	Mary	0
3	age = 35.0	0.014379	age	35	1.0	Mary	0
4	fare = 25.0	-0.014103	fare	25	-1.0	Mary	0
5	parch = 0.0	0.012243	parch	0	1.0	Mary	0
6	sibsp = 0.0	0.007677	sibsp	0	1.0	Mary	0

Figure 9-66. Results for Mary passenger

```
sh_mary.plot(bar_width = 16)
```

Figure 9-67 shows the SHAP contribution for the passenger named Mary. This plot is a representation of the table in Figure 9-68.

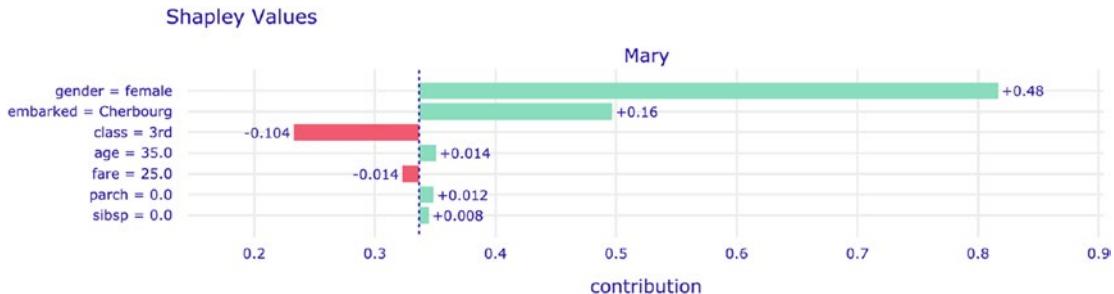


Figure 9-67. Prediction breakdown for Mary using SHAP

Figure 9-68 shows the same type of plot for the passenger named John.

```
exp.predict_parts(john, type='shap', B = 10, label=john.index[0]).  
plot(max_vars=5)
```



Figure 9-68. prediction breakdown for John using SHAP

predict_profile

- ‘ceteris_paribus’

This function computes individual profiles a.k.a. CP profiles (see <https://ema.drwhy.ai/ceterisParibus.html>). CP profiles show how a model’s prediction would change if the value of a single exploratory variable changed. Figure 9-69 shows the same for passenger Mary. Changing the variable gender to male reduces it from 0.89 to 0.12.

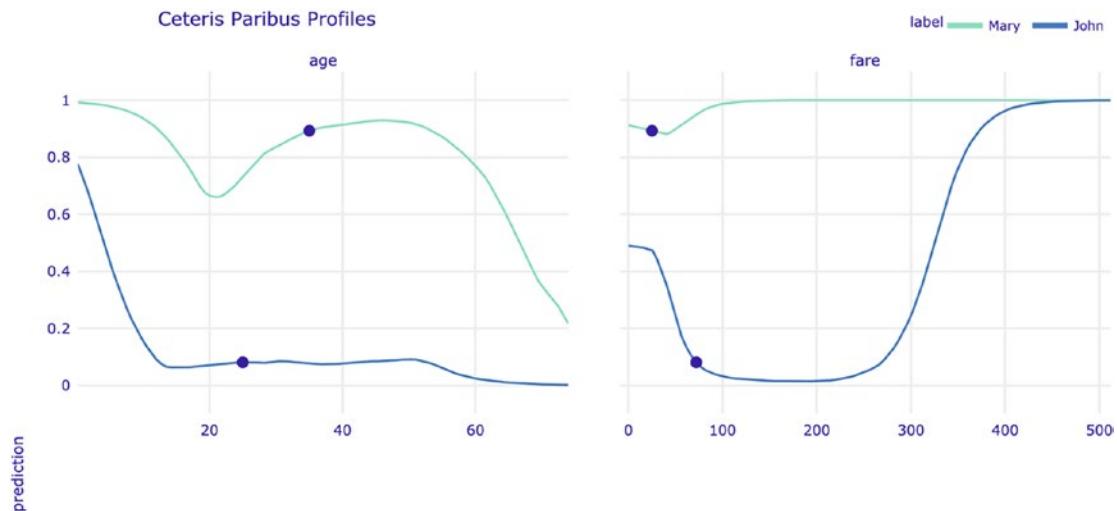
```
cp_mary = exp.predict_profile(mary, label=mary.index[0])  
cp_john = exp.predict_profile(john, label=john.index[0])  
  
cp_mary.result.head()
```

	gender	age	class	embarked	fare	sibsp	parch	_original_	_yhat_	_vname_	_ids_	_label_
Mary	male	35.000000	3rd	Cherbourg	25.0	0.0	0.0	female	0.129222	gender	Mary	Mary
Mary	female	35.000000	3rd	Cherbourg	25.0	0.0	0.0	female	0.892914	gender	Mary	Mary
Mary	female	0.166667	3rd	Cherbourg	25.0	0.0	0.0	35	0.992733	age	Mary	Mary
Mary	female	0.905000	3rd	Cherbourg	25.0	0.0	0.0	35	0.991484	age	Mary	Mary
Mary	female	1.643333	3rd	Cherbourg	25.0	0.0	0.0	35	0.990023	age	Mary	Mary

Figure 9-69.

```
cp_mary.plot(cp_john)
```

Figure 9-70 shows the CP profiles of different variables. The prediction value falls significantly for age near 20. These plots can be a very handy tool to understand how output changes which feature values. A very similar phenomena to counterfactual explanations are discussed in upcoming chapters.

**Figure 9-70.** CP profiles for variable age and fare

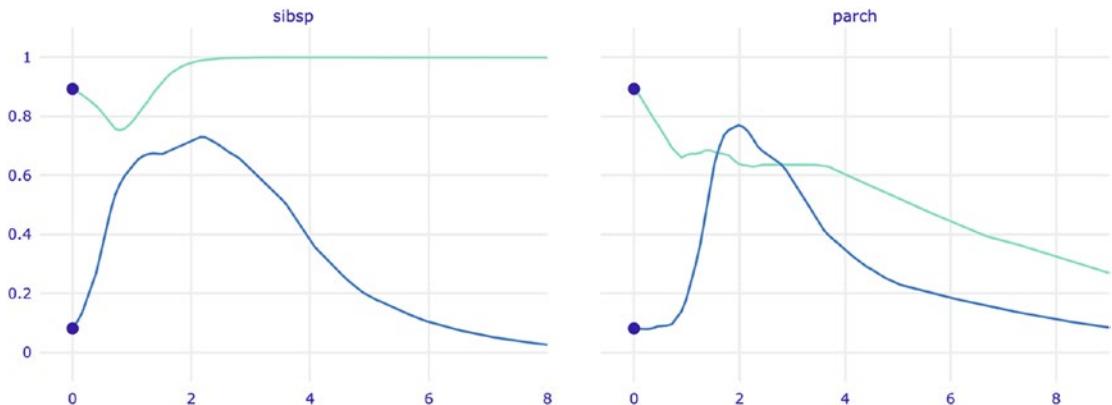


Figure 9-71. CP profiles for variable sidsp and parch

Model-level Explanations

model_performance

- ‘classification’
- ‘regression’

This function calculates various model performance measures.

- classification: F1, accuracy, recall, precision, and AUC
- regression: mean squared error, R squared, median absolute deviation

Figure 9-72 shows the sample model output with recall, precision, f1, accuracy, and AUC metrics.

```
mp = exp.model_performance(model_type = 'classification')
mp.result
```

	recall	precision	f1	accuracy	auc
MLPClassifier	0.651195	0.813708	0.723437	0.839601	0.877305

Figure 9-72. Model accuracy output

```
mp.result.auc[0]
```

```
0.877305256586716
```

```
mp.plot(geom="roc")
```

We can also generate ROC curves, as shown in Figure 9-73.

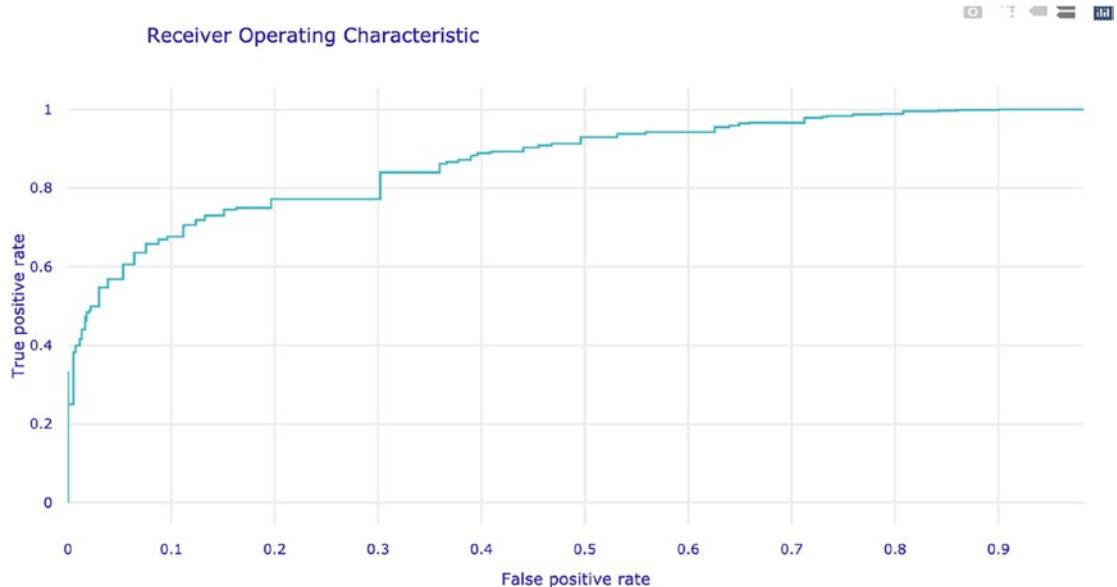


Figure 9-73. ROC curve for the model

model_parts

- ‘variable_importance’
- ‘ratio’
- ‘difference’

This function calculates Variable Importance.

```
vi = exp.model_parts()  
vi.result
```

Figure 9-74 shows the variable importance based on the model loss. Figure 9-75 is a pictorial description of variable importance.

	variable	dropout_loss	label
0	_full_model_	0.120623	MLPClassifier
1	parch	0.148471	MLPClassifier
2	fare	0.149296	MLPClassifier
3	sibsp	0.158364	MLPClassifier
4	embarked	0.166941	MLPClassifier
5	age	0.202489	MLPClassifier
6	class	0.255905	MLPClassifier
7	gender	0.368589	MLPClassifier
8	_baseline_	0.486014	MLPClassifier

Figure 9-74. Variable importance measures for the model

```
vi.plot(max_vars=5)
```



Figure 9-75. variable importance plots for model

There is also a possibility of calculating variable importance of group of variables as shown in Figure 9-76.

```
vi_grouped = exp.model_parts(variable_groups={'personal': ['gender', 'age',
'sibsp', 'parch'], 'wealth': ['class', 'fare']})
vi_grouped.result
```

	variable	dropout_loss	label
0	_full_model_	0.120793	MLPClassifier
1	wealth	0.267377	MLPClassifier
2	personal	0.465402	MLPClassifier
3	_baseline_	0.497155	MLPClassifier

Figure 9-76. Variable importance for a group of variables

model_profile

- ‘partial’
- ‘accumulated’

This function calculates explanations that explore model response as a function of selected variables.

The explanations can be calculated as a *partial dependence profile* or an *accumulated local dependence profile* (see Figure 9-77).

```
pdp_num = exp.model_profile(type = 'partial', label="pdp")
ale_num = exp.model_profile(type = 'accumulated', label="ale")
pdp_num.plot(ale_num)
```

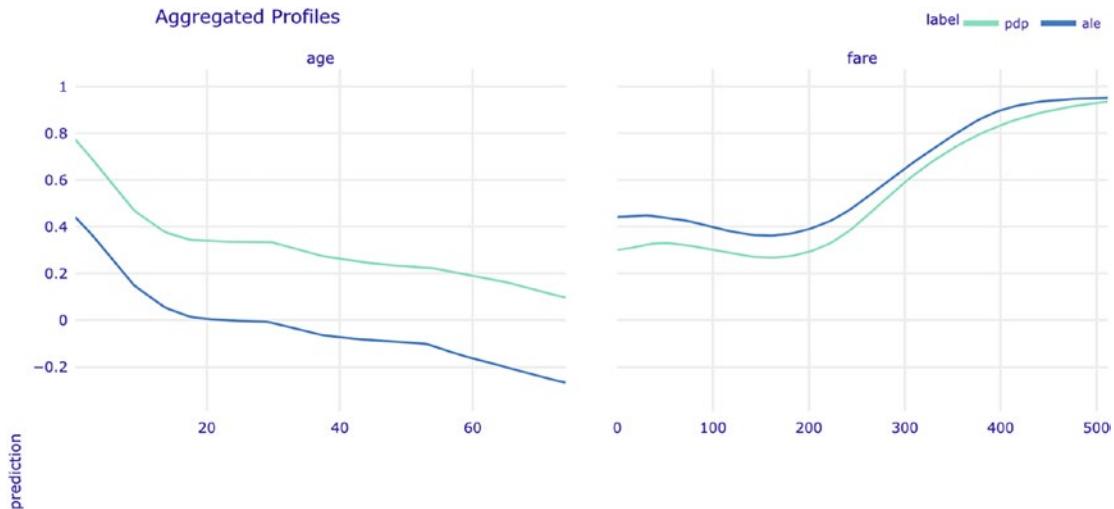


Figure 9-77. partial dependence plots for variables age and fare

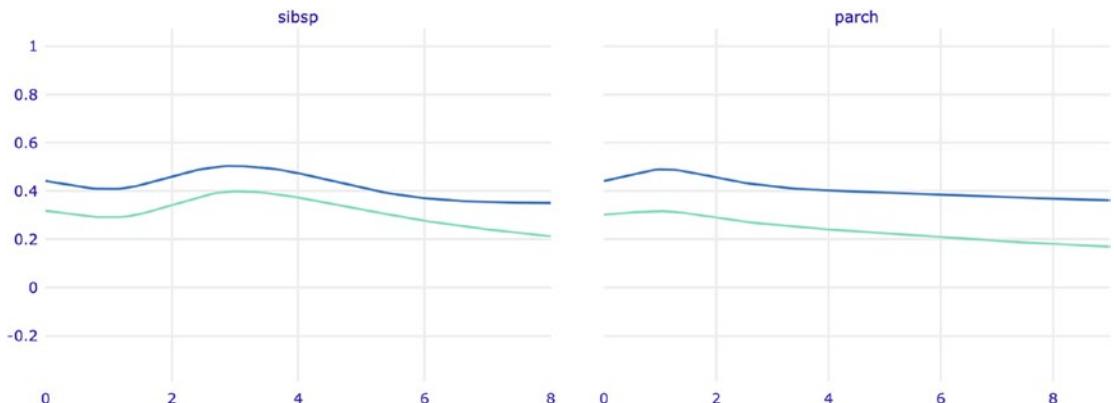


Figure 9-78. Partial dependence plots for variables sibsp and parch

```
pdp_cat = exp.model_profile(type = 'partial', variable_type='categorical',
                             variables = ["gender","class"], label="pdp")

ale_cat = exp.model_profile(type = 'accumulated', variable_
                             type='categorical', variables = ["gender","class"], label="ale")

ale_cat.plot(pdp_cat)
```

Summary

This chapter discussed quite a few methods. It incorporated the code wherever possible. The feature importance methods are the most famous domain of machine learning interpretability because they help you understand the role of features in the model. A lot of research has been done in this field, and stakeholders across various firms are also comfortable with feature importance methods. Methods such as SHAP and LIME are now widely used, and more and more people understand how these algorithms explain the model and their decision-making process. The next chapter talks about the rule-based methods, which are easier to comprehend than the feature importance method and are human-friendly.

CHAPTER 10

Detailing Rule-Based Methods

The previous chapter covered different feature importance methods. Let's now shift the focus toward different rule-based methods. One advantage of rule-based methods is that apart from giving the important features, these methods also provide the values of those features for easy understanding of why the model made a particular decision. The rule-based methods are easy to understand by both technical practitioners and stakeholders. Let's investigate some of the rule-based methods.

We have used partial dependence plots, feature importance methods, and intrinsic models to understand model behavior for the last few years. SHAP has recently gained a lot of attention for its post hoc model interpretability.

These techniques or methods learn monotonic response functions. When we use complex black-box models such as neural networks or boosting methods, then mostly the response functions are non-monotonic. In non-monotonic functions, changes in the input variables in the same direction can lead to response variables changing in different directions. Let's look at the example in the model for the data from previous chapters. In the online shoppers intention model, the age variable increases from 20 to 25, which may decrease the intention for shopping. However, those ages 40 to 45 might produce the opposite effect and increase the intention of shopping. Many models exhibit such behavior, and interpreting such models at a global level is not an easy to crack problem. You can use a surrogate model in certain methods where a decision tree is extracted to represent the model behavior. In such cases, each path to the end node goes through the first variables on which the tree is split. Hence all the rules include this variable. Also, each tree can be multiple levels deep which might make it more complex to derive rules.

Another approach can be to use the Apriori algorithm; however, it generates conditions with a high frequency in the data set and may not necessarily distinguish between classes.

Let's now look at some approaches that do not have the disadvantages mentioned earlier.

MAGIE (Model-Agnostic Global Interpretable Explanations)

The MAGIE algorithm is a global model-agnostic method for model explanations. It uses the LIME algorithm to learn important conditions that explain the classification of certain instances. Rules are learned by a *genetic algorithm* that tries different combinations of conditions to optimize a fitness function. Hence, this algorithm can be thought of as an extension to LIME that explains model behavior globally in independent if-then rules.

Next, let's look at how the algorithm is structured. Let's start with the mathematical terms used in the algorithm.

Let's assume that the MAGIE algorithm takes as input a data set, D, and a model, M, trained using that data set. The data set $D = \{(x_i, y_i)\} N i=1$, where each row consists of an instance $x_i = \{x_1 i, x_2 i, \dots, x_p i\}$ having fixed p columns and a class label $y_i \in Y = \{Y_1, \dots, Y_k\}$, the set of k classes. The classification model M has a $\text{predict-proba}(x_i)$ function that assigns a $1 \times k$ -vector of probabilities to each $x_i \in D$. Each element in the vector represents the probability that the instance belongs to the corresponding class. The algorithm outputs the set of rules for each class Y_j . Each rule is a conjunction of conditions, and a condition is a set of constraints on values of one feature column.

Table 10-1 highlights the metrics in this algorithm.

Table 10-1. Symbols and Definitions of Metrics and Terms Used in the Algorithm

Symbol	Definition
X	The training data, consisting of instances.
Y	A set of predicted classes.
X _i	An instance in the training set.
Y _i	The predicted class.
C _i	A condition, which is a combination of an attribute and a range of values the attribute can take.
R _i	A classification rule that classifies an instance to belong to its associated class if a predicate consisting of a conjunction (AND) of conditions hold.
C _i	A set of conditions for rule R _i .
y _i	An associated target class for rule R _i .
cover(R _i)	A set of instances rule R _i covers in the training data. These are instances for which the conditions of R _i are true.
correct-cover(R _i)	A set of instances for which the rule's class prediction y _i agrees with the model's prediction (Formally, it is the set {x x ∈ cover(R _i) and y _i = M(x)} where M(x) is the class predicted by the classifier for the instance x.)
Incorrect-cover(R _i)	Set of instances that are incorrectly covered by R _i .
R	Rule set. Consists of a set of rules of the form R _i .
correct-cover(R)	Correct-cover of rule set R. It is defined as the union of the correct covers of R _i for each rule in R.
Cover(R)	Cover of rule set R. It is defined as the union of the covers of R _i for each rule in R.

- **Precision rule:** The precision of R_i is the ratio of the number of instances in correct-cover(R_i) to the number of instances in cover(R_i).

$$\text{Precision}(R_i) = \frac{\text{len}(\text{correct-cover}(R_i))}{\text{len}(\text{cover}(R_i))}$$

- **Length rule:** The length of R_i is the cardinality of the precondition set for the rule C_i .
- **Class coverage rule:** The class coverage of R_i is the ratio of the number of instances in $\text{correct-cover}(R_i)$ to the number of instances in the training set that the classifier has predicted to have label y_i .
- **Mutual information rule:** The mutual information rule (RMI) captures its mutual dependence with predicted class. It is derived using the MI, as follows.

```

x11 Number correctly classified = TP
x12 Number incorrectly classified = FP
x21 Number incorrectly not classified = FN
x22 Number correctly not classified = TN

row1 = TP + FP;
row2 = FN + TN;
col1 = TP + FN;
col2 = FP + TN;
total = TP + FP + FN + TN;

mi11=TP*log((TP * total) / (row1 * col1));
mi12=FP*log((FP * total) / (row1 * col2));
mi21=FN*log((FN * total) / (row2 * col1));
mi22=TN*log((TN * total) / (row2 * col2));
mutualInformation = mi11 + mi12 + mi21 + mi22;

comparison = TP >= (row1/total)*col1;

if comparison = True then achievedRMI = mutualInformation
else achievedRMI = -mutualInformation;

NormalizedRMI = achievedRMI / bestPossibleRMI

```

- **Set precision rule:** The precision of R is the ratio of the number of instances in $\text{correct-cover}(R)$ to the number of instances in $\text{cover}(R)$.

- **Set class coverage rule:** The class coverage of R is the ratio of the number of instances in $\text{correct-cover}(R)$ to the number of instances in the training set that the classifier has predicted to have label y_i where y_i is the class that the rules in R predict.

Now that you have looked at all the metrics that are part of this algorithm, let's dive into the approach MAGIE to generate if-then-else rules. The next section explains the various steps, including processing the data before creating rules and processing the rules for final output.

MAGIE Algorithm Approach

The approach of the algorithm is best described in Figure 10-1. The process is simple. First, generate instance-level rules and then combine them to represent class-level rules. The class level rules are then processed to remove duplicates and redundant rules. The final set is derived and is presented as the output. Next, let's look at each step.

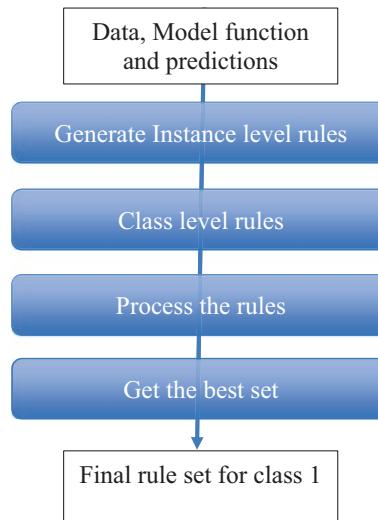


Figure 10-1. The approach of the MAGIE algorithm

Preprocessing the Input Data

This step preprocesses the input data happens to make each feature categorical. Categorical features are left unmodified. For numerical features, entropy-based binning is done to split the feature values into discrete bins. For instance, if the attribute age takes values from 10 to 85 (inclusive), then the binning step could produce the ranges ‘ $10 \leq \text{age} < 25$ ’, ‘ $25 \leq \text{age} < 60$ ’ and ‘ $60 \leq \text{age} \leq 85$ ’. After this step, the input data comprises only categorical features. Then original data set is divided into training and testing data sets.

Generating Instance Level Conditions

The algorithm iterates over each instance in the training data. For each instance, there is a particular class that the instance has been classified into. The marginal contribution of each feature-value used by the model in arriving at this classification is computed. The approach perturbs the training instance in question and trains a locally faithful linear model in the locality of the instance under consideration. The weights of the different features then approximate the marginal contribution values. The output of this step is a list of conditions. Each condition consists of a single feature and a value.

Learning Rules from Conditions

The output of the previous step of generating instance-level conditions is a list of conditions that are important at the instance level. There are a set of conditions that were important in classifying instances of that class for each class. We want to learn the rules for that class. Each rule has an associated coverage and precision.

Hence, the problem is the following. How to build rules having high precision and coverage from these base conditions? This could be done in a combinatorially prohibitive manner. For instance, first, evaluate all rules of length 1, then length 2, and so on. However, if the output of the previous step gives N conditions, then the complexity of this algorithm would be 2^N . This is not computationally feasible for larger data sets.

There should be an approach that can learn optimal combinations of conditions. The algorithm should take as input a notion of optimality and subsequently combine conditions to generate optimal rules. Another point that needs to be considered is that if there are any categorical variables in the data, we want the rules to allow these

categorical variables to take more than one value. For example, if we had a categorical variable called *country* and if ‘ $10 \leq \text{age} < 25$ ’, ‘*country* = US’ and ‘*country* = India’ are conditions for class 2, then one candidate rule could be ‘If $10 \leq \text{age} < 25$ AND *country* $\in \{\text{US}, \text{India}\}$ THEN predict class 2’. Therefore, the combinations should be not only ANDs of conditions but also ORs within a condition involving a categorical variable to allow such a variable to take on multiple possible values. A genetic algorithm learns these rules under the given conditions. The algorithm is run independently for each class. Hence, class-level rules are learned from class-level conditions. Each part of the genetic algorithm represents a possible rule. For example, if the number of conditions generated in the previous step (learning instance-level conditions) was 100, everyone in the population would be a bit string of length 100. One example string might be 100100000....000. The example described represents a rule of the form ‘If condition1 AND condition4 THEN predict class 1’. Each condition that involves a categorical variable has been allowed to take more than one value. The fitness function should capture several requirements.

- **Precision and coverage:** The mutual information rule (RMI) captures this notion of simultaneously optimizing for both precision and coverage. The high RMI rules are neither highly precise with low coverage nor overly generic and imprecise with high coverage.
- **Length:** The fitness function adds a length factor. The intuition here is that shorter rules are easier to interpret. And while it is true that rule length often inversely correlates with coverage, however making this parameter explicit leads to shorter, more interpretable rules.
- **Overlap:** An instance may be covered by one rule that says it should belong to class 1 and another rule that says it should belong to class 2. Only one of these is correct and minimizes ambiguity in the final rule set. Since rules should have high precision, meaning that they were correctly copying model behavior, the amount of overlap in the final rule set is minimized as we optimize precision.

The population with individuals that have a high probability of being “fit” is taken. These are individuals with only one bit set in the bit string (such as 1000, 0100, 0010, and 0001), followed by those with only two bits set (1100, 0110, 0101...) and so forth until the entire population size (i.e., 1,200 individuals) is reached. The algorithm runs for 600

generations. All the individuals of the last generation are finally selected as a rule set for one class. Hence, the output of this step is an exhaustive rule set for each class. Several rules do not add value to the set and need to be filtered out.

Postprocessing Rules

The rules generated in the previous step include several redundant rules. To eliminate them, sorting of the generated rules is done in descending order of precision. Then, for each rule R_i , it is checked whether $\text{correct-cover}(R_i) \subseteq \text{correct-cover}(R)$ and $\text{precision}(R_i) \leq \text{precision}(R)$, where R is a more precise rule not yet removed. Then R_i for the next step is not considered. Otherwise, the rule is retained for consideration. The original data set is split into training and testing subsets. The rules are learned by observing model behavior on the training data set. Then, for each rule, the precision of the testing data set is calculated. If the precision is less than the baseline precision on the test data, the rule is discarded at this step. This removes the noisy rules with a positive RMI in the training data set but low precision in the test data set. Baseline precision is the fraction of the instances in the test set for which the model predicts y_i , the class associated with the rule R_i .

Sorting Rules by Mutual Information

The previous section gives a combination of ORs (a subset of rules) among the ANDs (combining clauses into rules) generated by the genetic algorithm. Once the optimal set of rules for a class are derived, they are sorted to provide the user with the most relevant rules at the top and less relevant ones further down the list. This is done as follows. First, the rules are sorted in descending order of RMI. Then, the rule with the highest RMI is selected and added to the top of the list. Then, the rule with the next highest value of RMI is selected. The rule is compared to the list of already added rules. If it is similar to any of the rules already added, we discard this rule. Otherwise, it is added to the list of rules. The similarity is computed using the Jaccard similarity measure between the list of instances covered by the two rules. If the Jaccard similarity is greater than 50%, the rules are deemed to be similar. This process is repeated till we have a desired number of rules.

Now since you understand the major steps in the algorithm, let's look at the framework of the execution of the preceding steps. Figure 10-2 shows a more detailed framework of the MAGIE execution (see <https://github.com/viadee/magie>).

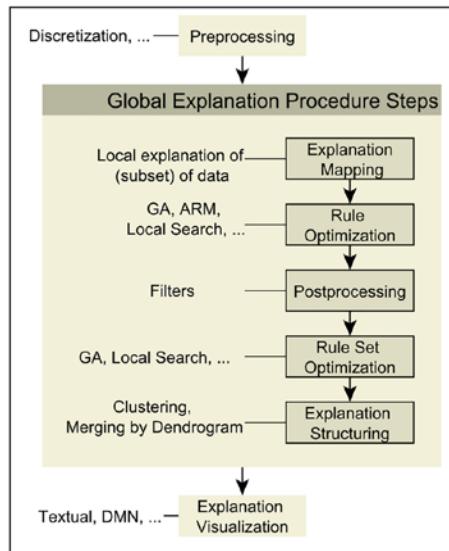


Figure 10-2. A detailed framework of MAGIE algorithm execution

- The **explanation mapping step** from Figure 10-2 corresponds to the initial creation of local explanations. These local explanations can be used as building blocks to create a global explanation. The term “explanation mapping” hence denotes the mapping of local explanations into the domain of global explanations. The framework can utilize these explanations to initialize optimization procedures and serve as the search space for further investigation of rules.
- The **optimization step** in Figure 10-2 refers to the two types of rule explanation optimization, which are executable by using the framework: first, the optimization of individual rules, and then the optimization of rule sets. Optimizers are coarsely divided into an initializer, a representation translator, an objective function, and an optimization procedure. The task of the initializer is to accept a forwarded set of rule explanations; for example, from an explanation mapping step, and initialize a population of rule explanations that are to be optimized. You might simply use the forwarded rules as an initial population or single conditions of the rule’s antecedent. The representation translator transforms the representation used

by the algorithm into a rule explanation (set). By doing so, the same optimization procedure optimizes rule sets and individual rules simply by providing a new representation translator.

- The **rule optimization step** in Figure 10-2 accepts a Rule explanation set as input. If a condition (i.e., a categorical predictor feature's value) exists within one of the rules, it is added to the overall search space of the optimization procedure. This search space can then be explored to find well-performing rules. The objective function should be applied to evaluate the quality of a rule. Currently, mutual information-based measures are utilized in the MAGIE algorithm, as explained in Table 10-1.
- The **rule set optimizers step** also accept a rule explanation set. The set of rules within the forwarded rule explanation set forms the search space in this case. Rule set optimizers aim is to reduce the number of rules within the given rule explanation set while maintaining a good coverage and quality of explanation.
- The **postprocessing step** denotes filtering steps and changes to rules and rule sets that do not utilize a search mechanism. For example, if two rules cover too many of the same data instances of the data set, only the “better” one is kept. Another example might be the comparison of a rule's metric with some defined baseline, such as the baseline precision. Finally, the contribution of a rule to a priorly created rule set might be evaluated.
- The **explanation structuring** step denotes the preparation of rule sets for displaying them to a user. The rule sets are created for each label value in separation. Thus, in the phase of explanation structuring, explanations of different labels might be clustered together. Rule explanations might also be joined in a dendrogram-like fashion which is the output of the MAGIE algorithm.

Finally, the structured explanations are to be visualized to the user.

GLocalX

GLocalX (GLObal to loCAL eXplainer) is a model-agnostic local-first explanation algorithm. GLocalX is based on the idea of deriving global explanations by inference on a set of logical rules representing local explanations. GLocalX aggregates local explanations expressed in the form of logical rules into a global explanation by iteratively “merging” the rules in a growing hierarchical structure while accounting for both fidelity (i.e., accuracy in emulating black-box predictions) and complexity of the rules. The merge procedure estimates a distance between explanations and yields a set of sorted candidate pairs to merge. Then, the pair with minimum distance satisfying constraints on both fidelity and complexity is processed to guarantee generalization and the updated explanation replaces the selected pairs. Constraint satisfaction is ensured on each merge. This guarantees high fidelity and low complexity on the final explanation yielded by GLocalX.

Empirical results over different black-box models and data sets indicate that GLocalX achieves both a high fidelity and a low complexity of the rule set representing the model explanation. Compared to transparent models that either optimize model complexity or fidelity, but not both, GLocalX simultaneously reaches high fidelity and low complexity. The high accuracy in prediction tasks also suggests that GLocalX might be used directly as a transparent model to replace global classifiers adopted in AI systems.

Local to Global Explanation Problem

Let $e_1, \dots, e_n \in E$ be a set of local explanations for a black-box classifier b defined in a human interpretable domain E . The local to global explanation problem consists in finding a g function yielding an explanation theory $E = g(e_1, \dots, e_n) \subset E$, such that E describes the overall logic according to which b makes decisions. To solve the local to global explanation problem, we need to generalize the local explanations, as they are accurate and faithful locally but not globally. Given a black-box classifier b adopted in an AI system, and a set of instances $X_{le} = \{x_1, \dots, x_n\}$ explained locally. Their local explanations $\{e_1, \dots, e_n\}$, the problem is solved by deriving an explanation theory $E = \{e_0 1, \dots, e_0 k\}$ by refining with an aggregation function g the local explanations into an explanation theory emulating the global decision logic of the black-box classifier b . Thus, the human interpretable domain E consists of a set of logical decision rules.

Local to Global Hierarchy of Explanation Theories

GLocalX method hierarchically merges local explanations into a global explanation theory. In particular, GLocalX takes as input a set of local explanations in the form of explanation theories $E = \{E_1, \dots, E_n\}$ where each theory $E_i = \{e_i\}$ is formed by a single explanation (i.e., $|E_i| = 1 \forall E_i \in E$). GLocalX iteratively merges the explanation theories and returns an explanation theory, $E = \{e_0 1, \dots, e_0 k\}$. It emulates the global behavior of the black-box classifier b , simultaneously maintaining the overall simple and interpretable model. At each iteration, GLocalX merges the closest pair of explanation theories E_i, E_j by using a notion of similarity between logical theories. The pairs are filtered out according to the merge quality criterion: if no pair satisfying the criterion is found, GLocalX halts prematurely without building the full hierarchy. The resulting hierarchy of explanation theories can be represented by using a tree-like diagram called a *dendrogram*. There are two key elements in the GLocalX approach: (i) similarity search, which allows to select which theories to merge and refine, and (ii) a merge function, which allows to refine the explanations.

Given a set of explanation theories E , a pairwise similarity function and a quality criterion, the logical theories are sorted by similarity in a queue Q_1 . Then, a batch of data is sampled to merge the candidate theories. Using batches instead of the whole training data set favors diverse merges, as the merge procedure has different behaviors according to the data at hand. In the merge loop, the queue is popped to find the most similar pair of theories whose merge satisfies a quality criterion and, the merge operation is run. If the merger is advantageous, the merged theory is kept.

As a quality criterion, the *Bayesian information criterion* (BIC) is used because it rewards models for their simplicity and accuracy. BIC has been successfully adopted in various techniques, such as clustering or adopting bisecting hierarchical refinement of the model. After a successful merge of the two explanations, E_i, E_j are replaced with the merged theory E_{i+j} . If no advantageous merge is found, GLocalX halts. This process is iterated until no more merges are possible. Finally, explanations with low fidelity are filtered out to reduce the output size. A α parameter indicates this per-class trimming threshold.

Finding Similar Theories

Selecting pairs of theories to merge requires the definition of a pairwise similarity function on logical explanation theories. To this aim, the similarity of two theories, E_1, E_2 is defined as the Jaccard similarity of their coverage on a given instance set X .

$$\text{similarity}_X(E_i, E_j) = \frac{|\text{coverage}(E_i, X) \cap \text{coverage}(E_j, X)|}{|\text{coverage}(E_i, X) \cup \text{coverage}(E_j, X)|}.$$

The larger is the shared coverage of E_i and E_j on X , the more similar the two logical explanation theories are. Coverage similarity is a two-faceted measure that captures the premise similarity and the coverage similarity. The former is straightforward: rules with similar premises have similar coverage. The latter balances the premise similarity to avoid rules with similar premises, but low coverage sway the similarity score.

The merge function allows GLocalX to generalize a set of explanation theories while balancing fidelity and complexity through approximate logical entailment. The merge involves two operators—join and cut—to simultaneously generalize and preserve a high level of fidelity. In the logical domain, generalization seldom involves premises relaxation or outright removal. Thus, GLocalX advances the state-of-the-art in exploiting this kind of generalization. Pushing for generalization may pull down fidelity. These contrasting behaviors are the focal point in a local to global setting and must be dealt with accordingly. This problem is tackled with a merge function that handles both rule generalization and fidelity. Suppose there are two explanation theories, $E_1 = \{e_1, e_2\}$ and $E_2 = \{e'_1, e'_2\}$, with e_1, e'_1 explaining a record x_1 and e_2, e'_2 explaining a record x_2 .

$$\begin{aligned} E_1 &= \{e_1 = \{\text{age} \geq 25, \text{job} = \text{unemployed}, \text{amount} \leq 10k\} \rightarrow \text{deny} \\ &\quad e_2 = \{\text{age} \geq 50, \text{job} = \text{office clerk}\} \rightarrow \text{deny}\} \\ E_2 &= \{e'_1 = \{\text{age} \geq 20, \text{job} = \text{manager}, \text{amount} > 8k\} \rightarrow \text{accept} \\ &\quad e'_2 = \{\text{age} \geq 40, \text{job} = \text{office clerk}, \text{amount} > 5k\} \rightarrow \text{deny}\} \end{aligned}$$

The resulting merge yields the following rules E_{1+2} :

$$\begin{aligned} E_{1+2} &= \{e''_1 = \{\text{age} \geq 25, \text{job} = \text{unemployed}, \text{amount} \leq 10k\} \rightarrow \text{deny} \\ &\quad e''_2 = \{\text{age} \in [20, 25], \text{job} = \text{manager}, \text{amount} \in [8k, 10k]\} \rightarrow \text{accept} \\ &\quad e''_3 = \{\text{age} \geq 40\} \rightarrow \text{deny}\} \end{aligned}$$

In E_{1+2} , rules with equal predictions from different explanation theories have been generalized by relaxing their premises (rule e'_3) while rules with different predictions have been specialized by further constraining them (rule e'_2). Specifically, e'_1, e'_2 results from a cut on e_1, e'_1 , that is $e_1 - e'_1$. More formally, given two explanation theories E_i, E_j , the merge function applies two operators on the explanation theories to derive a new theory approximately entailed by the two: the join and the cut operators. The former allows merging non-conflicting rules, while the latter allows merging conflicting rules.

A set of explanations $E = \{e_1, \dots, e_n\}$ part of a logical explanation theory is considered conflicting on an instance x if two or more of them cover an instance x but lead to two different outcomes. The merge of two logical explanation theories E_i and E_j applies the join operator on non-conflicting explanations and the cut operator on conflicting explanations iteratively on each instance in the batch. The resulting set of explanations composes the new explanation theory E_{i+j} . The candidate merge is then tested for considering the equilibrium between fidelity and complexity with the BIC computation.

In most cases, the model log-likelihood is computed as the rule's fidelity, and the model complexity as the average rule length. In the following, we provide details of the join and cut operators. The two operators move in opposite directions: join generalizes explanations, possibly at a fidelity cost, while the cut specializes explanations, possibly at a generalization cost. In other words, the former allows generalization while the latter regularizes it. Inspired by this, we define the join and the cut operators through an alternative representation of decision rules. Let X_{bi} be the set of subspaces on the feature i . Given a decision rule $r = P \rightarrow y$ we have that any $P_i \in X_{bi}$ is a subspace on the feature i . The premise P of the rule identifies a quasi-polyhedron defined as a subspace of $X_b(m)$.

$$P = \{P_1, \dots, P_m\} \in P(X_{b1} \times \dots \times X_{bm})$$

An instance x satisfies P if $\forall P_i \in P. x_i \in P_i$. Thus, x satisfies P if it lies in the subspace defined by the quasi-polyhedron. We define the operators join (\oplus) and cut ($-$) exploited by the merge function based on this quasi-polyhedron interpretation.

The join operator aims to generalize a set of non-conflicting explanations relaxing their premises, hence generalizing the associated rules. Given two quasi-polyhedra, P and Q , the join (\oplus) is defined as follows: $P \oplus Q = \{P_1 + Q_1, \dots, P_m + Q_m\}$.

where:

$$P_i + Q_i = \begin{cases} P_i \cup Q_i & \text{non-empty intersection} \\ [\min\{P_i \cup Q_i\}, \max\{P_i \cup Q_i\}] & \text{empty intersection} \\ \emptyset & P_i \text{ is empty} \vee Q_i \text{ empty} \end{cases}$$

Example. Consider the following two explanations:

$$\begin{aligned} e_1 &= \{\text{age} \geq 50, \text{job} = \text{office clerk}\} \rightarrow \text{deny} \\ e_2 &= \{\text{age} \geq 40\} \rightarrow \text{deny} \end{aligned}$$

The merge function applies the JOIN operator $e_1 \oplus e_2$ and returns

$$e'_1 = \{\text{age} \geq 40\} \rightarrow \text{deny}$$

The cut operator acts in a complementary fashion by slicing quasi-polyhedra. Here, the goal is to preserve the better rules and confine the lesser ones to subspaces with high fidelity. In other words, the aim is to remove overlaps between rules by subtracting them. This directly translates to subtracting quasi-polyhedra. Formally, given two quasi-polyhedra P and Q , the cut () is defined as follows.

$$P(-)Q = \{P_1 - Q_1, \dots, P_m - Q_m\}$$

where:

$$P_i - Q_i = \begin{cases} \{P_i, \emptyset\} & Q_i \text{ empty} \\ \{P_i, Q_i \setminus Q_i\} & \text{otherwise} \end{cases}$$

Note that, unlike the join operator, the cut operator is not symmetric, hence $P(-)Q = Q(-)P$. With the goal of preserving high-fidelity rules and restricting lower fidelity rules, it is straightforward to select subtracted and subtracting polyhedron, with the two rules being the one with higher and lower fidelity, respectively.

Consider the following two explanations: $e_1 = \{\text{age} \geq 25, \text{job} = \text{unemployed}, \text{amount} \geq 10K\} \rightarrow \text{deny}$ $e_2 = \{\text{age} \geq 20, \text{job} = \text{manager}, \text{amount} > 8K\} \rightarrow \text{accept}$ where the first one is the dominant one, that is, the one with highest fidelity. The merge function applies the cut operator $e_1 \ominus e_2$ and returns $e_1 = \{\text{age} \geq 25, \text{job} = \text{unemployed}, \text{amount} \geq 10K\} \rightarrow \text{deny}$ $e_0 = \{\text{age} \in [20, 25], \text{job} = \text{manager}, \text{amount} \in [8K, 10K]\} \rightarrow \text{accept}$. Note that e_1

is preserved as-is while the premises of e2 are further constrained to reduce overlap. Namely, age is constrained to remove overlap on $\text{age} \geq 25$ and amount is constrained to remove overlap on $\text{amount} \geq 10K$.

Join and cut produce sets of different cardinalities. While cut preserves the number of rules, join lessens it by subsuming it in a smaller, more general explanation set. The next section shows GLocalX in action using a sample data set and the Python code.

Code

Let's consider the following data set, which is the data of customer churns for a telecom company. The variables present in the data are state of the customer address, account length in no of days, area code, phone number, whether an international plan or not, voice mail plan, number of vmail messages, total day minutes, total day calls, total day charge, total eve calls, total eve charge, total night minutes, total night calls, total night charge, total intl minutes, total intl calls, total intl charge, customer service calls. The target variable is churn which contains True and False values. The True value is for customers who churned or left the services of the telecom company. The False value is for customers who have an active plan.

Let's look at the distribution of the target variable.

```
class_name = 'churn' #Target Variable Name
data[class_name] = data[class_name].replace({True: 1, False: 0})
data.shape

(3333, 21)

data.drop('phone number', axis=1, inplace=True)
data[class_name].value_counts()

0    2850
1    483
Name: churn, dtype: int64
```

For the sake of simplicity, let's make the classes evenly balanced.

```
import imblearn
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler
```

```

X = data.drop(class_name, axis=1)
y = data[class_name]

# define oversampling strategy
over = RandomOverSampler(sampling_strategy=0.2)
# fit and apply the transform
X, y = over.fit_resample(X, y)
# define undersampling strategy
under = RandomUnderSampler(sampling_strategy=0.9)
# fit and apply the transform
X, y = under.fit_resample(X, y)

data = pd.concat([X,y],axis=1)
#data.shape

data[class_name].value_counts()

0    633
1    570
Name: churn, dtype: int64

```

Let's look at the data types of different variables.

#	Column	Non-Null Count	Dtype
0	state	1203 non-null	object
1	account length	1203 non-null	int64
2	area code	1203 non-null	int64
3	international plan	1203 non-null	object
4	voice mail plan	1203 non-null	object
5	number vmail messages	1203 non-null	int64
6	total day minutes	1203 non-null	float64
7	total day calls	1203 non-null	int64
8	total day charge	1203 non-null	float64
9	total eve minutes	1203 non-null	float64
10	total eve calls	1203 non-null	int64
11	total eve charge	1203 non-null	float64
12	total night minutes	1203 non-null	float64

```

13 total night calls      1203 non-null    int64
14 total night charge    1203 non-null    float64
15 total intl minutes    1203 non-null    float64
16 total intl calls      1203 non-null    int64
17 total intl charge     1203 non-null    float64
18 customer service calls 1203 non-null    int64
19 churn                  1203 non-null    int64
dtypes: float64(8), int64(9), object(3)
memory usage: 188.1+ KB

```

Important Note! We need to handle categorical features before passing to LORE and GLocalX; otherwise, categorical variables are treated as continuous, and a decision rule is made on them using some threshold value.

If your data set has a very high dimension (too many columns), you can use SHAP to select the top 20–30 variables and then pass this data set for further process!

- **Data set:** Splitting generates explanations.
- **Train set:** Trains the black-box model.
- **Test set:** Tests the black-box model accuracy and then generates the local rules of them by LORE.
- **Validation set:** Validates the result of GLocalX on its global rules.

```

from scoped_rules.utils import data_processing

features = data_processing(data, class_name)
df, feature_names, class_values, numeric_columns, rdf, real_feature_names,
features_map = features.prepare_variables()
X_train, y_train, X_test, y_test, r_X_test, val, test = features.data_
splitting(test_size = 0.16, val_size = 0.05)

Length of train set is 959
Length of test set is 183
Length of val set is 61

```

This section converted all categorical variables to one-hot encoded.

Modeling our random forest classification model. This model is used as a black box for rules generation.

```
from sklearn.metrics import accuracy_score, f1_score, classification_report
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier

bb = RandomForestClassifier()
bb.fit(X_train, y_train)

def bb_predict(X):
    return bb.predict(X)

def bb_predict_proba(X):
    return bb.predict_proba(X)

y_pred = bb_predict(X_test)

print('Accuracy %.3f' % accuracy_score(y_test, y_pred))
print('F1-measure %.3f' % f1_score(y_test, y_pred))

Accuracy 0.880
F1-measure 0.872
```

LORE is a model-agnostic method able to provide interpretable and faithful explanations. LORE first learns a local interpretable predictor on a synthetic neighborhood generated by a genetic algorithm. Then it derives from the logic of the local interpretable predictor a meaningful explanation consisting of a decision rule, which explains the reasons for the decision.

We have used LORE because it is more accurate and stable than other local rules techniques such as anchors.

```
from scoped_rules.lore.lorem import LOREM
from scoped_rules.lore.explanation import *
from scoped_rules.lore.rule import *
from scoped_rules.lore.util import *
import concurrent.futures
import os
import time
```

```

lore_explainer = LOREM(r_X_test, bb_predict, feature_names, class_name,
class_values, numeric_columns, features_map,
                      neigh_type='geneticcp', categorical_use_prob=True,
                      continuous_fun_estimation=False,
                      size=1000, ocr=0.1, random_state=0, ngen=10,
                      bb_predict_proba=bb_predict_proba,
                      verbose=True)

print()

#If you want to check local explanation of any test-set instance

x = X_test[115] #115th instance from our 183 X_test instances
exp = lore_explainer.explain_instance(x, samples=300, use_weights=True,
metric=euclidean)
print()
print()
print(exp)

```

Output

calculating feature values

generating neighborhood - geneticcp

gen	nevals	avg	min	max
0	150	0.496654	0.496654	0.496654
1	130	0.767585	0.496654	0.993262
2	129	0.972472	0.496654	0.993301
3	122	0.992499	0.98517	0.993301
4	135	0.98904	0.496654	0.993301
5	118	0.99216	0.972332	0.993301
6	122	0.992392	0.97464	0.99328
7	128	0.992371	0.983344	0.99328
8	127	0.99234	0.984633	0.993287
9	132	0.992416	0.984333	0.993287
10	121	0.992546	0.987313	0.993287

```

gen      nevals      avg      min      max
0        150        0.5       0.5       0.5
1        135        0.500185   0.498573  0.509783
2        119        0.500508   0.498552  0.509982
3        131        0.501412   0.498649  0.51975
4        132        0.501696   0.497202  0.51975
5        123        0.502945   0.497272  0.51975
6        131        0.504018   0.498004  0.537397
7        132        0.505359   0.498784  0.537397
8        125        0.506727   0.498335  0.527432
9        123        0.508359   0.497412  0.55501
10       125        0.509345   0.497186  0.55501
synthetic neighborhood class counts {0: 188, 1: 160}
learning local decision tree
retrieving explanation

r = { customer service calls <= 3.02, voice mail plan = yes, total day
minutes <= 241.59, total day charge <= 42.59, total eve charge > 22.50 }
--> { churn: 0 }
c =  }

```

For global explanation, you need to generate local explanation rules for all test instances. Multiprocessing technique utilizes all the resources and parallelize the processes. The Python parallel processing library's concurrent.futures module generates the local explanation parallelly leveraging high number of CPU cores available.

```

def lore_rules(instance):
    exp = lore_explainer.explain_instance(X_test[instance], samples=300,
                                           use_weights=True)
    rule = ExplanationEncoder.default(instance,exp)
    return rule

def local_rules(lore_explainer, X_test):
    start_time = time.time()
    rules = []
    with concurrent.futures.ProcessPoolExecutor(max_workers=94) as
        executor:

```

```

results = executor.map(lore_rules,[x for x in range(len(X_test))])
for result in results:
    rules.append(result)
print()
print()
print(f'Total {len(rules)} local rules generated')
end_time = time.time()
print('time taken to generate local rules in sec: ', (end_time -
start_time))
return rules

rules = local_rules(lore_explainer, X_test)

```

GLocalX is a “local-first” model-agnostic explanation method. Starting from local explanations expressed in the form of local decision rules, GLocalX iteratively generalizes them into global explanations by hierarchically aggregating them (merging the significant rules).

The generated local rules are merged, and GLocalX returns the final global decision rule for each class of target variable.

```

from scoped_rules.global_interpret import glocalx_explanations

model = glocalx_explanations(rules, bb, feature_names, class_values, class_
name, val, test, data_path)

results = model.global_rules(batch_size = 32)

Local Rules json file stored at path: /mnt/data/scoped_rules_package/data/
local_rules.json
Val_set stored at path: /mnt/data/scoped_rules_package/data/val.csv
Test_set stored at path: /mnt/data/scoped_rules_package/data/test.csv

Merging Rules....
Merge Complete!
Time taken for global rules in sec: 13.0773446559906

```

```

{
    customer service calls < 3.478666663169861
    total day charge < 42.9187068939209
    state=NE: True
    total day minutes < 264.5166015625

    Class: 0
}

{
    total day charge >= 38.63331413269043

    Class: 1
}

```

Hence, you see how to generate a simple if-then rule using GLocalX. One major advantage of GLocalX is the flexibility it gives in terms of execution. You can modify models to cater to the needs of the data set. Since it is model-agnostic, you can use any type of model. It also provides flexibility to modify and merge rules in the final step. Sometimes we might want to code rules differently than the default output. For example, the default output can be encoded classes, whereas you can combine a few classes to represent rules as a group.

Skope-Rules

Complex machine learning models sometimes achieve high performance but lead to opaque decisions. Due to regulation or severe consequences on errors, more interpretability is often necessary. Here, we present skope-rules, a rule-based interpretable model.

Skope-rules, the proposed interpretable model, aims at learning decision rules for “scoping” a target class; that is, detecting instances of this class with high precision. The problem of generating such rules has been widely considered (see RuleFit, Slipper, LRI, MLRules).

However, this approach mainly differs in how decision rules are chosen: semantic deduplication based on variables composing each rule as opposed to L1-based feature selection (RuleFit).

Methodology

The methodology of the skope-rules is depicted in Figure 10-3. It is a simple three-step process to generate rules from this method.

- **Bagging estimator training:** Multiple decision tree classifiers and potentially regressors (if a sample weight is applied) are trained. Note that each node in this bagging estimator can be seen as a rule.
- **Performance filtering:** Out-of-bag precision and recall thresholds are applied to select the best rule.
- **Semantic deduplication:** A similarity filtering is applied to maintain enough diversity among the rules. The similarity measure of the two rules is based on the number of their common terms. A term is a variable name combined with a comparison operator (< or >).

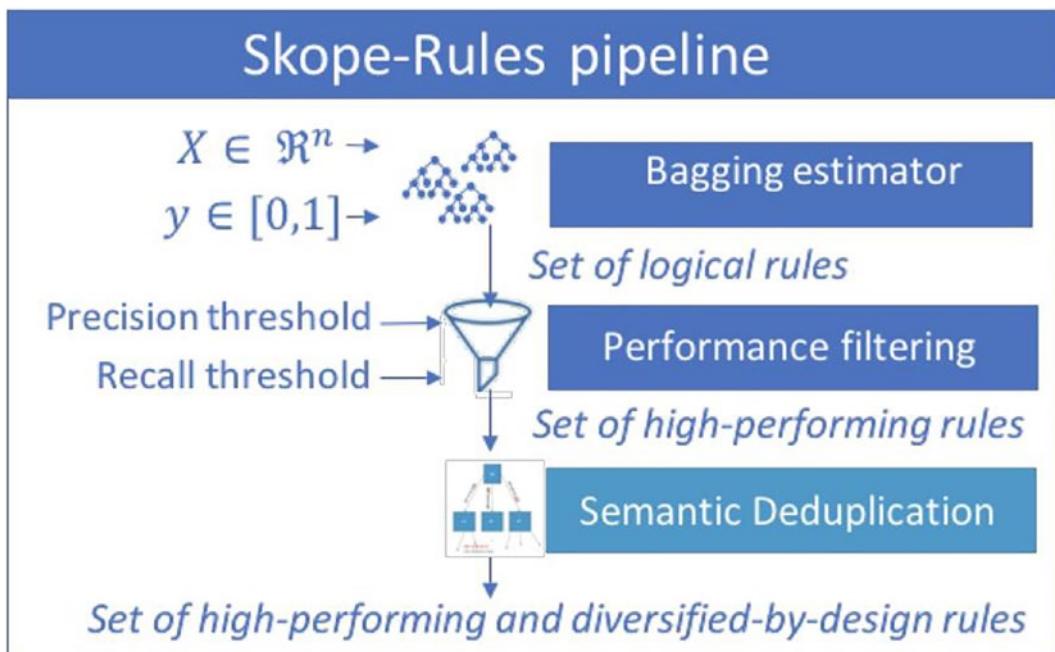


Figure 10-3. Skope-rules pipeline

Let's now look at a more detailed overview of the different steps. Figure 10-4 shows how the last step in semantic duplication works.

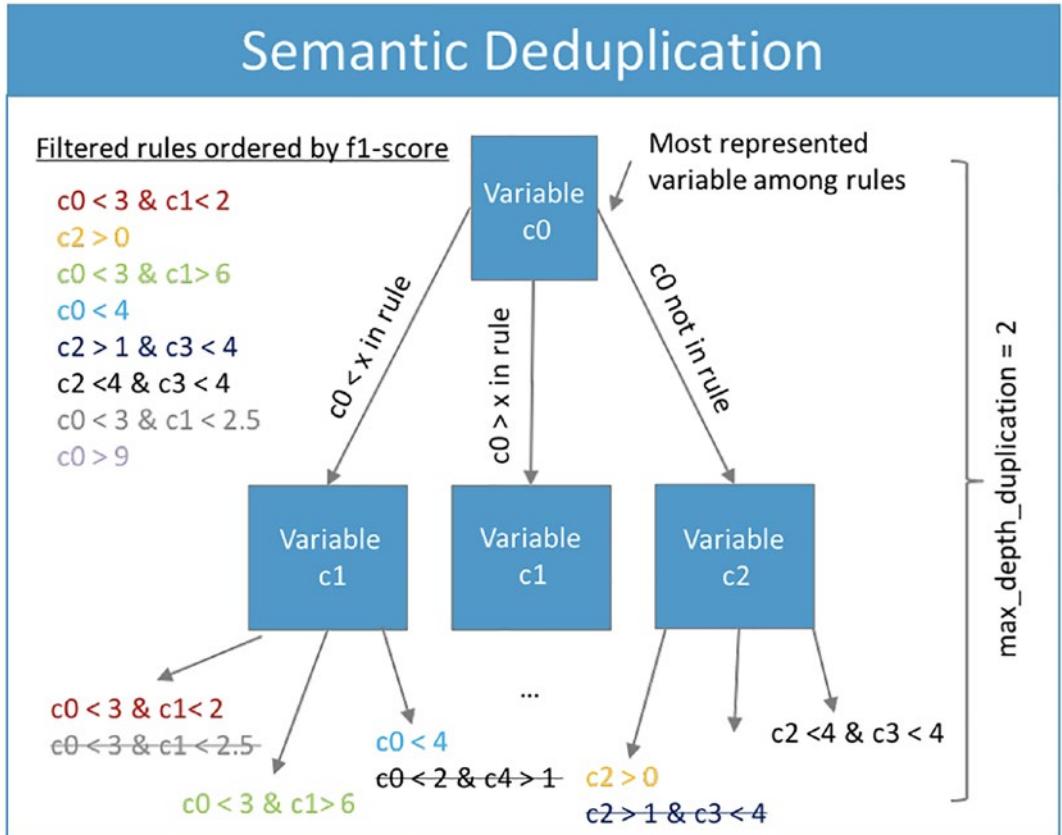


Figure 10-4. Skope-rules methodology

Implementation

The implementation of skope-rules is very simple and can be completed in a very simple piece of code depicted in Figure 10-5.

```

1 from skrules import SkopeRules
2
3 # Training
4 clf = SkopeRules(feature_names=X_train.columns)
5 clf.fit(X_train, y_train)
6
7 # Making predictions
8 y_score = clf.score_top_rules(X_test)
9
10 # Seeing the rules
11 clf.rules_

```

Figure 10-5. Skope-rules implementation example

Figure 10-6 shows an example of output from the skope-rules algorithm.

Example of output

debit_flows < 952 and credit_flows > 2411 and is_client_gold > 0.5,
 $(0.95, 0.16, 1)$

credit_balance > 327 and debit_flows < 523, (0.82, 0.33, 2)

Out-of-bag precision of the rule ↗
 Out-of-bag recall of the rule ↘
 This rule appeared in 2 trees ↓

Figure 10-6. Skope-rules output example

Skope-rules is suited for different applications and can be used as follows.

- **As a global interpretable model:** This is the natural use of skope-rules for a binary classification task. Rules isolate the 1s from 0s.
- **As a cluster describer:** In a clustering task, skope-rules describe each segment. Each cluster can be post-processed and approximated with a set of interpretable rules.

- **As a distribution queue describer:** Skope-rules is also relevant to describing how any subsample differs from a population. More precisely, it describes subsamples defined by a given variable's highest (or lowest) values. Skope-rules can be effective at understanding the highest (or lowest) prediction scores of another complex classifier.

Next, let's use the same data set we have been using since Chapter 9 and see how to implement a working example of skope-rules.

The response variable classes are coded as Will Earn and Won't Earn.

```
c=np.array(list3)
print(c)
print(list3)
list4=[]
list4.append('Will Earn')
list4.append('Wont Earn')

import sys
import six
sys.modules['sklearn.externals.six'] = six
import skrules
from skrules import SkopeRules

clf = SkopeRules(max_depth_duplication=2,
                  n_estimators=30,
                  precision_min=0.3,
                  recall_min=0.1,
                  feature_names=list)

for idx1, revenue in enumerate(list4):
    clf.fit(x_train, y_train == idx1)
    rules = clf.rules_[0:17]
    print("Rules for Revenue ",revenue)#Ask
    for rule in rules:
        print(rule)
```

```

print()
print(20*'=')
print()

```

Rules for Revenue Will Earn

```

('Month <= 6.5 and PageValues <= 1.2655065059661865', (0.9865938430983118,
0.6362471982068524, 1))
('Month > 6.5 and PageValues <= 0.0335247740149498', (0.9024152875257552,
0.2542760117477433, 4))
=====
```

Rules for Revenue Wont Earn

```

('BounceRates <= 0.0004056630132254213 and PageValues > 9.61924123764038',
(0.8133333333333334, 0.4299559471365639, 1))
('Administrative_Duration > 9.25 and PageValues <= 19.113906860351562
and PageValues > 0.7777977585792542', (0.3952254641909814,
0.2634836427939876, 1))
('BounceRates > 0.0010931899887509644 and PageValues > 8.887159824371338',
(0.4906716417910448, 0.2269197584124245, 1))
('BounceRates > 8.075000005192123e-05 and PageValues <= 15.39547348022461
and PageValues > 0.9448194801807404', (0.3780260707635009,
0.18076580587711488, 1))
('BounceRates <= 0.0004102485108887777 and PageValues <= 20.44874382019043
and PageValues > 2.515446901321411', (0.6148409893992933,
0.15038893690579083, 1))
```

You can see the output of the skope-rules as simple rules displayed for each class. Important variables for which rules are defined are PageValues and BounceRates for class Won't Earn. In the positive class, the Will Earn rules are made on the Month and PageValues variables.

Anchors

Anchors explain any black-box classification model's predictions by finding a rule that "anchors" the prediction sufficiently. A rule anchors a prediction if changes in other feature values do not affect the prediction. Anchors utilize reinforcement learning

techniques combined with a graph search algorithm to reduce the number of model calls (and hence the required runtime) to a minimum while still recovering from local optima. Ribeiro, Singh, and Guestrin proposed the algorithm in 2018. The same researchers that introduced the LIME algorithm.

Like its predecessor, the anchors approach deploys a *perturbation-based* strategy to generate *local* explanations for predictions of black-box machine learning models. However, instead of surrogate models used by LIME, the resulting explanations are expressed as easy-to-understand if-then rules, called *anchors*. These rules are reusable since they are *scoped*: anchors include the notion of coverage, stating precisely to which other, possibly unseen, instances they apply. Finding anchors involves an exploration or multi-armed bandit problem, which originates in the discipline of reinforcement learning. To this end, neighbors, or perturbations, are created and evaluated for every instance being explained. Doing so allows the approach to disregard the black box's structure and its internal parameters so that these can remain both unobserved and unaltered. Thus, the algorithm is *model-agnostic*, meaning it can be applied to *any* model class.

Figure 10-7 depicts both LIME and anchors locally explaining a complex binary classifier (predicts either + or -) using two exemplary instances. LIME's results do not indicate how faithful they are because LIME solely learns a linear decision boundary that best approximates the model. Given the same perturbation space, the anchors approach constructs explanations whose coverage is adapted to the model's behavior and clearly expresses their boundaries. Thus, they are faithful by design and state exactly for which instances they are valid. This property makes anchors particularly intuitive and easy to comprehend.

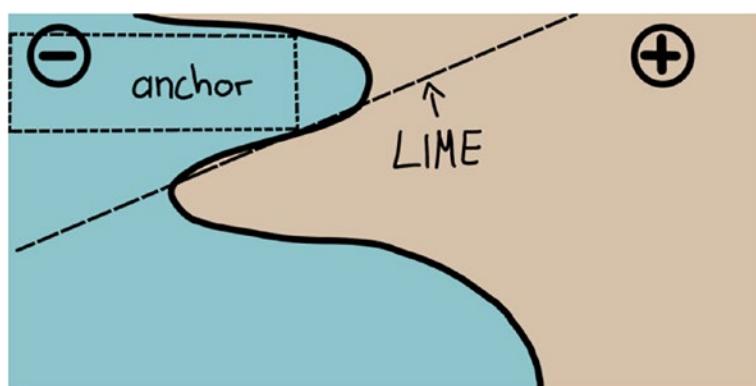


Figure 10-7. Representation of anchor compared to LIME

As mentioned, the algorithm's results or explanations come in the form of rules called anchors. Figure 10-8 illustrates such an anchor. For instance, suppose we are given a bivariate black-box model that predicts whether a passenger survives the *Titanic* disaster. Now we would like to know *why* the model predicts for one specific individual that it survives.

one exemplary individual and the model's prediction	
Feature	Value
Age	20
Sex	female
Class	first
TicketPrice	300\$
More attributes	...
Survived	true

Figure 10-8. Anchor example

The anchors algorithm provides the following result explanation.

```
IF SEX = female
AND Class = first
THEN PREDICT Survived = true
WITH PRECISION 97%
AND COVERAGE 15%
```

The example shows how anchors can provide essential insights into a model's prediction and its underlying reasoning. The result shows which attributes were considered by the model, which is the female sex and first class. Humans, being paramount for correctness, can use this rule to validate the model's behavior. The anchor also applies to 15% of the perturbation space's instances. In those cases, the explanation is 97% accurate, meaning the displayed predicates are almost exclusively responsible for the predicted outcome.

Anchor A is formally defined as follows.

$$\text{ED}_x(z|A)[\text{if}(x)=f(z)] \geq \tau, A(x)=1$$

- x represents the instance being explained (e.g., one row in a tabular data set).
- A is a set of predicates (i.e., the resulting rule or anchor), such that $A(x)=1$ when all feature predicates defined by A correspond to x 's feature values.
- f denotes the classification model to be explained (e.g., an artificial neural network model). It can be queried to predict a label for x and its perturbations.
- $Dx(\cdot|A)$ indicates the distribution of neighbors of x , matching A .
- $0 \leq \tau \leq 1$ specifies a precision threshold. Only rules that achieve a local fidelity of at least τ are considered valid results.

Finding Anchors

Although the mathematical description of anchors may seem clear and straightforward, constructing particular rules is infeasible. It would require evaluating the following.

$$\{f(x) = f(z) \mid f(x) = f(z)\} \text{ for all } z \in Dx(\cdot|A)$$

This is not possible in continuous or large input spaces. Therefore, the authors propose introducing the $0 \leq \delta \leq 10 \leq \delta \leq 1$ parameter to create a probabilistic definition. This way, samples are drawn until there is statistical confidence concerning their precision. The probabilistic definition reads as follows.

$$P(\text{prec}(A) \geq \tau) \geq 1 - \delta \text{ with } \text{prec}(A) = EDx(z|A)[\{f(x) = f(z)\}]$$

The previous two definitions are combined and extended by the notion of coverage. Its rationale consists of finding rules that apply to a large part of the model's input space. Coverage is formally defined as an anchor's probability of applying to its neighbors (i.e., its perturbation space).

$$\text{cov}(A) = ED(z)[A(z)]$$

Including this element leads to the anchor's final definition considering the maximization of coverage.

$$\text{Max cov}(A)$$

$$\text{As.t. } P(\text{prec}(A) \geq \tau) \geq 1 - \delta$$

Thus, the proceeding strives for a rule with the highest coverage among all eligible rules (all those that satisfy the precision threshold given the probabilistic definition). These rules are thought to be more important, as they describe a larger part of the model. Note that rules with more predicates tend to have higher precision than rules with fewer predicates. A rule that fixes every feature of xx reduces the evaluated neighborhood to identical instances. Thus, the model classifies all neighbors equally, and the rule's precision is 1. At the same time, a rule that fixes many features is overly specific and only applicable to a few instances. Hence, there is a *trade-off between precision and coverage*.

The anchors approach uses four main components to find explanations, as is shown in Figure 10-9.

- **Candidate generation:** Generates new explanation candidates. In the first round, one candidate per feature of x gets created and fixes the respective value of possible perturbations. In every other round, the best candidates of the previous round are extended by one feature predicate not yet contained therein.
- **Best candidate identification:** Candidate rules are to be compared in regard to which rule explains xx the best. To this end, perturbations that match the currently observed rule are created evaluated by calling the model. However, these calls need to be minimized as to limit computational overhead. This is why there is a pure-exploration multi-armed-bandit (MAB; KL-LUCB, to be precise) at the core of this component. MABs efficiently explore and exploit different strategies (called *arms* in an analogy to slot machines) using sequential selection. In the given setting, each candidate rule is seen as an arm that can be pulled. Each time it is pulled, respective neighbors are evaluated, and we thereby obtain more information about the candidate rule's payoff (precision in anchors' case). The precision thus states how well the rule describes the instance to be explained.
- **Candidate precision validation:** Takes more samples if there is no statistical confidence that the candidate exceeds the τ threshold.

- **Modified beam search:** All the components are assembled in a beam search, a graph search algorithm, and a variant of the breadth-first algorithm. It carries the B best candidates of each round over to the next one (where B is called the *beam width*). These B best rules are then created new rules. The beam search conducts at most featureCount(x) rounds, as each feature can only be included in a rule at most once. Thus, at every round, i, it generates candidates with exactly ii predicates and selects the B best thereof. Therefore, by setting B high, the algorithm more likely avoids local optima. In turn, this requires a high number of model calls and thereby increases the computational load.

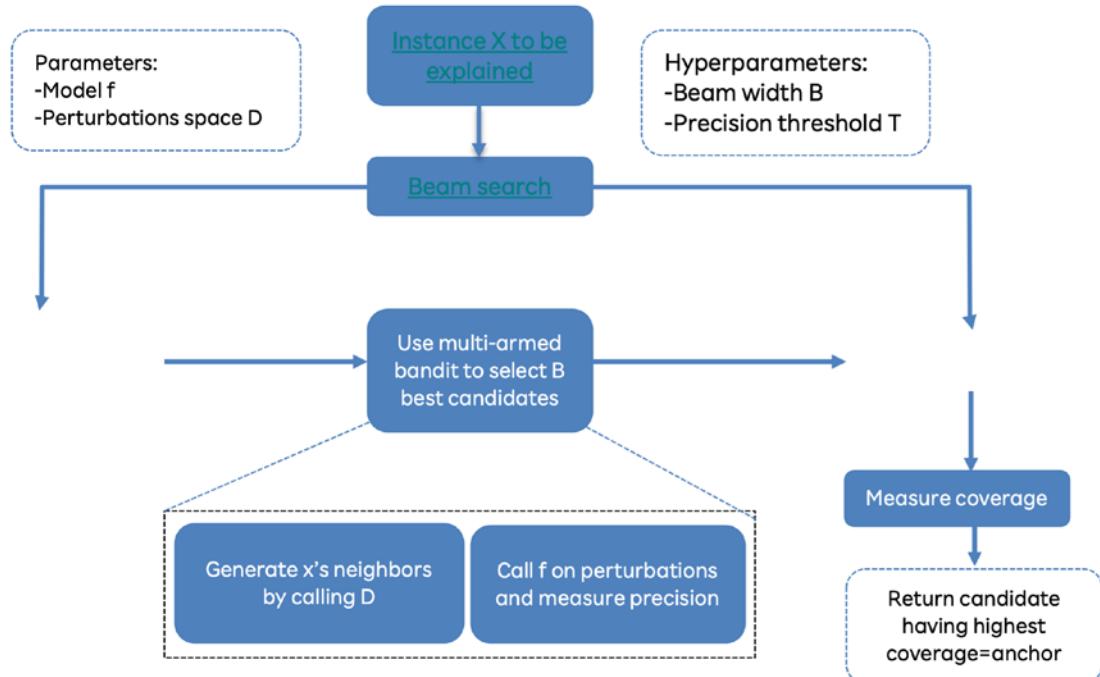


Figure 10-9. Anchors flow chart

Advantages

The anchors approach offers multiple advantages over LIME. First, the algorithm's output is easier to understand, as rules are **easy to interpret** (even for laypersons).

Furthermore, **anchors are subsettable** and even state a measure of importance by including the notion of coverage. Second, the anchors approach **works when model predictions are non-linear or complex** in an instance's neighborhood. As the approach deploys reinforcement learning techniques instead of fitting surrogate models, it is less likely to underfit it.

Apart from that, the algorithm is **model-agnostic** and thus applicable to any model.

Furthermore, it is **highly efficient as it can be parallelized** by using MABs that support batch sampling (e.g., BatchSAR).

Disadvantages

The algorithm suffers from a **highly configurable** and impactful setup, like most perturbation-based explainers. Hyperparameters such as the beam width or precision threshold need to be tuned to yield meaningful results. The perturbation function also needs to be explicitly designed for one domain/use case. Think of how tabular data gets perturbed (perturbation means adding noise to data) and apply the same concepts to image data. (Hint: These cannot be applied.) Luckily, default approaches may be used in some domains (e.g., tabular), facilitating an initial explanation setup.

Also, **many scenarios require discretization** as otherwise, results are too specific, have low coverage, and do not contribute to understanding the model. While discretization can help, it may also blur decision boundaries if used carelessly, thus having the exact opposite effect. Since there is no best discretization technique, users need to be aware of the data before deciding how to discretize data without obtaining poor results.

Constructing anchors requires **many calls to the ML model**, like all perturbation-based explainers. While the algorithm deploys MABs to minimize the number of calls, its runtime still depends on the model's performance and is highly variable.

Finally, the notion of **coverage is undefined in some domains**. For example, there is no obvious or universal definition of how superpixels in one image compare to such in other images.

Next, let's take a data set provided by default in the anchor algorithm and use it to generate simple rules. You can follow the code step by step to understand the anchor implementation.

Import anchor-related libraries.

```
from __future__ import print_function
import numpy as np
np.random.seed(1)
import sys
import sklearn
import sklearn.ensemble
%load_ext autoreload
%autoreload 2
from anchor import utils
from anchor import anchor_tabular
```

This data set is about predicting if a person makes more or less than \$50,000.

```
dataset = utils.load_dataset('adult', balance=True, dataset_folder=dataset_
folder, discretize=True)
```

Let's train a classifier for the following.

```
c = sklearn.ensemble.RandomForestClassifier(n_estimators=50, n_jobs=5)
c.fit(dataset.train, dataset.labels_train)
print('Train', sklearn.metrics.accuracy_score(dataset.labels_train,
c.predict(dataset.train)))
print('Test', sklearn.metrics.accuracy_score(dataset.labels_test,
c.predict(dataset.test)))
```

Getting an Anchor

Now let's start the explainer. We need the training data to perturb instances. `categorical_names` is a map from integer to list of strings, containing names for each value of the categorical features. Every feature that is not in this map is considered ordinal or continuous and thus discretized.

```
explainer = anchor_tabular.AnchorTabularExplainer(
    dataset.class_names,
    dataset.feature_names,
    dataset.train,
    dataset.categorical_names)
```

Next, we get an anchor for prediction number 0. An anchor is a sufficient condition; that is, when the anchor holds, the prediction should be the same as the prediction for this instance.

```
idx = 0
np.random.seed(1)
print('Prediction: ', explainer.class_names[c.predict(dataset.test[idx].reshape(1, -1))[0]])
exp = explainer.explain_instance(dataset.test[idx], c.predict,
threshold=0.95)

Prediction: b'>50K'

print('Anchor: %s' % (' AND '.join(exp.names())))
print('Precision: %.2f' % exp.precision())
print('Coverage: %.2f' % exp.coverage())

Anchor: Education = Bachelors AND Relationship = Husband AND
Occupation = Sales
Precision: 0.97
Coverage: 0.02
```

Note that we set threshold to 0.95, so we guarantee (with high probability) that precision is above 0.95; that is, that predictions on instances where the anchor holds is the same as the original prediction at least 95% of the time. Let's try it out on the test set.

```
# Get test examples where the anchor applies
fit_anchor = np.where(np.all(dataset.test[:, exp.features()] == dataset.
test[idx][exp.features()], axis=1))[0]
print('Anchor test precision: %.2f' % (np.mean(c.predict(dataset.test[fit_
anchor]) == c.predict(dataset.test[idx].reshape(1, -1)))))
print('Anchor test coverage: %.2f' % (fit_anchor.shape[0] / float(dataset.
test.shape[0])))

Anchor test precision: 0.97
Anchor test coverage: 0.02
```

Summary

This chapter is very important since the rule-based methods are not very common interpretability methods in circulation within the analytics industry. Most of the use cases still rely on SHAP and other feature importance-related methods; however, methods like MAGIE and GLocalX can bring in many benefits when it comes to understanding the behavior of each class within the modeling data set.

CHAPTER 11

Detailing Counterfactual Methods

The previous two chapters discussed feature importance methods and rule-based methods. Although the rule-based methods make life a lot easier for human stakeholders, a few gaps can be filled in model explanations. The outputs of both feature importance methods and rule-based methods are good for a model. Those methods can help in determining the behavior of a specific class in a classification model scenario. Counterfactual explanations are a step above the descriptive information and provide actions to make utilizing model results meaningful. This chapter discusses the actions provided by counterfactuals and how to derive more value from model explanations by implementing counterfactual explanations. We would like you to remember the word *actionable* to relate to counterfactual explanations. The output of counterfactuals can help derive meaningful actions from the outputs of the model decisions, which is much better than only having descriptive explanations of model decisions.

Counterfactual Explanations

A counterfactual explanation describes a causal situation in the form: “If X had not occurred, Y would not have occurred.” For example, “If I hadn’t taken a sip of this hot coffee, I wouldn’t have burned my tongue.” Event Y is that I burned my tongue; cause X is that I had a hot coffee. Counterfactuals requires imagining a hypothetical reality that contradicts the observed facts (for example, a world without hot coffee); hence, the name *counterfactual*. The ability to think in counterfactuals makes humans so smart compared to other animals.

In interpretable machine learning, counterfactual explanations explain predictions of individual instances. The “event” is the predicted outcome of an instance.

The “causes” are the feature values of this instance that were input to the model and “caused” a certain prediction.

Even if the relationship between the inputs and the predicted outcome might not be causal, we can see the inputs of a model as the cause of the prediction.

To simulate counterfactuals for predictions of machine learning models, we simply change the feature values of an instance before making the predictions, and we analyze how the prediction changes. We are interested in scenarios in which the prediction changes in a relevant way, like a flip in predicted class (for example, credit application accepted or rejected) or in which the prediction reaches a certain threshold (for example, the probability for cancer reaches 10%). A counterfactual explanation of a prediction describes the smallest change to the feature values that changes the prediction to a predefined output.

There are both model-agnostic and model-specific counterfactual explanation methods. This chapter focuses on model-agnostic methods that only work with the model inputs and outputs (and not the internal structure of specific models).

Unlike prototypes, counterfactuals do not have to be actual instances from the training data but can be a new combination of feature values.

Before discussing how to create counterfactuals, let's discuss some use cases.

Use Case 1: Banking Software

In this first example, Shankar applies for a loan and gets rejected by the (machine learning-powered) banking software. He wonders why his application was rejected and how he might improve his chances of getting a loan. The question of why can be formulated as a counterfactual: What is the smallest change to the features (income, number of credit cards, age, etc.) that would change the prediction from rejected to approved? One possible answer could be: If Shankar earned \$10,000 more per year, he would get the loan. Or if Shankar had fewer credit cards and had not defaulted on a loan five years ago, he would get the loan. Shankar will never know the reasons for the rejection, as the bank has no interest in transparency, but that is another story.

Use Case 2: Continuous Outcome

The second example explains a model that predicts a continuous outcome with counterfactual explanations. Garima wants to rent out her apartment, but she is unsure how much to charge for it, so she trains a machine learning model to predict the rent. Of course, since Garima is a data scientist, that is how she solves her problems. After entering all the details about size, location, whether pets are allowed, and so on, the model tells her that she can charge \$900. She expected \$1,000 or more, but she trusts her model and decides to play with the feature values of the apartment to see how she can improve the value of the apartment. She finds out that the apartment could be rented out for over \$1,000, if it were 15 m² larger. Interesting but non-actionable knowledge because she cannot enlarge her apartment. Finally, by tweaking only the feature values under her control (built-in kitchen—yes/no, pets allowed—yes/no, type of floor, etc.), she finds out if she allows pets and installs windows with better insulation, she can charge \$1,000. Garima had intuitively worked with counterfactuals to change the outcome.

Counterfactual Explanations at a Glance

Counterfactuals are human-friendly explanations because they usually focus on a small number of feature changes. Each counterfactual tells a different “story” of how a certain outcome was reached. One counterfactual might say to change feature A. Another counterfactual might say to leave A the same but change feature B, which is a contradiction. This issue of multiple truths can be addressed either by reporting all counterfactual explanations or by having criteria to evaluate counterfactuals and select the best one.

Speaking of criteria, how do we define a good counterfactual explanation? First, the user of a counterfactual explanation defines a relevant change in the prediction of an instance (= the alternative reality).

An obvious first requirement is that a counterfactual instance produces the predefined prediction as closely as possible. It is not always possible to find a counterfactual with a predefined prediction. For example, in a classification setting with two classes, a rare class, and a frequent class, the model might always classify an instance as the frequent class. Changing the feature values so that the predicted label would flip from the frequent class to the rare class might be impossible. We want, therefore, to relax the requirement that the prediction of the counterfactual must

exactly match the predefined outcome. In the classification example, we could look for a counterfactual where the predicted probability of the rare class is increased to 10% instead of the current 2%. What are the minimal changes in the features so that the predicted probability changes from 2% to 10% (or close to 10%)?

Another quality criterion is that a counterfactual should be as similar as possible to the instance regarding feature values. The distance between two instances can be measured, for example, with the Manhattan distance or the Gower distance if we have both discrete and continuous features. The counterfactual should not only be close to the original instance but should also change as few features as possible. We can simply count the number of changed features to measure how good a counterfactual explanation is in this metric.

Third, it is often desirable to generate multiple diverse counterfactual explanations so that the decision subject gets access to multiple viable ways of generating a different outcome. For instance, continuing the loan example, one counterfactual explanation might suggest only double the income to get a loan. In contrast, another counterfactual might suggest shifting to a nearby city and increasing the income by a small amount to get a loan. It could be noted that while the first counterfactual might be possible for some, the latter might be more actionable for some. Thus, besides providing a decision subject with different ways to get the desired outcome, diversity also enables “diverse” individuals to alter the features that are convenient for them.

The last requirement is that a counterfactual instance should have likely feature values. It would not make sense to generate a counterfactual explanation for the rent example where the size of an apartment is negative, or the number of rooms is set to 200. It is better when the counterfactual is likely according to the joint distribution of data; for example, a 20 m² apartment with ten rooms should not be regarded as a counterfactual explanation. Ideally, if the number of square meters is increased, the number of rooms should also be proposed.

The next section looks at how to generate counterfactual explanations using simple theories.

Generating Counterfactual Explanations

A simple and naive approach to generating counterfactual explanations is searching by trial and error. This approach involves randomly changing feature values of the instance of interest and stopping when the desired output is predicted. Like the example where

Anna tried to find a version of her apartment to charge more rent. But there are better approaches than trial and error. First, we define a loss function based on the criteria mentioned. This loss takes as input the instance of interest, a counterfactual, and the desired (counterfactual) outcome. Then, we can find the counterfactual explanation that minimizes this loss using an optimization algorithm. Many methods proceed in this way but differ in their definition of the loss function and optimization method.

To generate the counterfactuals, it is suggested to minimize the following loss.

$$L(x, x', y', \lambda) = \lambda \cdot (\hat{f}(x') - y')^2 + d(x, x')$$

Let's look at each component of the formula.

$$\lambda \cdot (\hat{f}(x') - y')^2$$

The first term is the quadratic distance between the model prediction for counterfactual x' and desired outcome y' , which the user must define in advance. The second term is the distance, d , between instance x to be explained and counterfactual x' . The loss measures how far the predicted outcome of the counterfactual is from the predefined outcome and how far the counterfactual is from the instance of interest.

$$\cdot d(x, x')$$

The distance function, d , is defined as the Manhattan distance weighted with the inverse median absolute deviation (MAD) of each feature.

$$d(x, x') = \sum_{j=1}^p \frac{|x_j - x'_j|}{MAD_j}$$

The total distance is the sum of all p feature-wise distances, that is, the absolute differences of feature values between instance x and counterfactual x' . The feature-wise distances are scaled by the inverse of the median absolute deviation of feature j over the data set defined as follows.

$$MAD_j = \text{median}_{i \in \{1, \dots, n\}} (|x_{i,j} - \text{median}_{l \in \{1, \dots, n\}} (x_{l,j})|)$$

The median of a vector is the value at which half of the vector values are greater and the other half smaller. The MAD is the equivalent of the variance of a feature. But, instead of using the mean as the center and summing over the square distances, we use the median as the center and sum over the absolute distances. The proposed distance function has the advantage over the Euclidean distance in that it is more robust to outliers. Scaling with the MAD is necessary to bring all the features to the same scale. It should not matter whether you measure the size of an apartment in square meters or square feet.

The parameter λ balances the distance in prediction (first term) against the distance in feature values (second term). The loss is solved for a given λ and returns a counterfactual x' . A higher value of λ means that we prefer counterfactuals with predictions close to the desired outcome y' . A lower value means that we prefer counterfactuals x' that are very similar to x in the feature values. If λ is very large, the instance with the prediction closest to y' is selected, regardless of how far it is away from x . Ultimately, the user must decide how to balance the requirement that the prediction for the counterfactual matches the desired outcome with the requirement that the counterfactual is similar to x .

To minimize this loss function, any suitable optimization algorithm can be used. If you have access to the gradients of the machine learning model, you can use gradient-based methods like Adam. The instance to be explained, x , the desired output, y' , and the tolerance parameter, ϵ , must be set in advance. The loss function is minimized for x' , and the (locally) optimal counterfactual, x' , returned while increasing λ until a sufficiently close solution is found (= within the tolerance parameter).

$$\arg \min_{x'} \max_{\lambda} L(x, x', y', \lambda).$$

Overall, the recipe for producing the counterfactuals is simple.

1. Select an instance, x , to be explained, the desired outcome, y' , a tolerance, ϵ , and a (low) initial value for λ .
2. Sample a random instance as an initial counterfactual.
3. Optimize the loss with the initially sampled counterfactual as a starting point.
4. While $|f(x') - y'| > \epsilon$,

- a. Increase λ .
 - b. Optimize the loss with the current counterfactual as starting point.
 - c. Return the counterfactual that minimizes the loss.
5. Repeat steps 2 to 4, and return the list of counterfactuals or the one that minimizes the loss.

The proposed method has some disadvantages. It only takes the first and second criteria into account, not the last two (“produce counterfactuals with only a few feature changes and likely feature values”). d does not prefer sparse solutions since increasing ten features by one gives the same distance to x as increasing one feature by ten. Unrealistic feature combinations are not penalized.

The method does not handle categorical features with many different levels well. The authors suggested running the method separately for each combination of feature values of the categorical features, but this leads to a combinatorial explosion if you have multiple categorical features with many values. For example, six categorical features with ten unique levels would mean one million runs.

Let's now look at another approach to these issues.

In this method, the following loss is minimized.

$$L(x, x', y', X^{obs}) = (o_1(\hat{f}(x'), y'), o_2(x, x'), o_3(x, x'), o_4(x', X^{obs}))$$

Each of the four objectives o_1 to o_4 corresponds to one of the four criteria mentioned. The first objective o_1 reflects that the counterfactual x' prediction should be as close as possible to our desired prediction y' . Therefore, we want to minimize the distance between $f(x')$ and y' , calculated by the Manhattan metric (L1 norm).

$$o_1(\hat{f}(x'), y') = \begin{cases} 0 & \text{if } \hat{f}(x') \in y' \\ \inf_{y' \in y'} |\hat{f}(x') - y'| & \text{else} \end{cases}$$

The second objective o_2 reflects that the counterfactual should be as similar as possible to instance x . It quantifies the distance between x' and x as the Gower distance.

$$o_2(x, x') = \frac{1}{p} \sum_{j=1}^p \delta_G(x_j, x'_j)$$

p is the number of features. The value of δ_G depends on the feature type of x_j .

$$\delta_G(x_j, x'_j) = \begin{cases} \frac{1}{\hat{R}_j} |x_j - x'_j| & \text{if } x_j \text{ numerical} \\ \mathbb{I}_{x_j \neq x'_j} & \text{if } x_j \text{ categorical} \end{cases}$$

Dividing the distance of a numeric feature j by R_j , the observed value range, scales δ_G for all features between 0 and 1.

The Gower distance can handle numerical and categorical features but does not count how many features were changed. Therefore, we count the number of features in a third objective, o_3 , using the L0 norm.

$$o_3(x, x') = ||x - x'||_0 = \sum_{j=1}^p \mathbb{I}_{x'_j \neq x_j}.$$

By minimizing o_3 , we aim for our third criteria—sparse feature changes.

The fourth objective, o_4 , reflects that the counterfactuals should have likely feature values/combinations. We can infer how “likely” a data point is using the training data or another data set. We denote this data set as X^{obs} . To estimate the likelihood, o_4 measures the average Gower distance between x' and the nearest observed data point $x^{[1]} \in X^{obs}$.

$$o_4(x', \mathbf{X}^{obs}) = \frac{1}{p} \sum_{j=1}^p \delta_G(x'_j, x_j^{[1]})$$

The two were the most general theories for generating counterfactual explanations. Apart from that, we would like to introduce various practical methods for generating counterfactuals from the data. These algorithms are discussed in the upcoming section.

- Counterfactual guided by prototypes
- DiCE
- MOC (Multi-Objective Counterfactuals)

Counterfactual Guided by Prototypes

This algorithm is an extension of the counterfactual explanation algorithm. It generates the counterfactuals using class prototypes.

The key insight of this formulation is the need to design an objective function that allows you to generate high-quality counterfactual instances.

A counterfactual instance should have the following desirable properties.

- The model prediction should be near the predefined output.
- The perturbation changing the original instance x to x' should be sparse.
- The counterfactual needs to be interpretable.
- The counterfactual needs to be found fast enough so that it can be applied in a real-life setting.

DiCE

DiCE can generate multiple counterfactuals that are diverse from each other but as close as possible to the original instance. Therefore, in the loss function, the following components are added.

- Distance between desired class and modeled class of a counterfactual
- Distance between different counterfactuals to ensure diversity between them
- Distance between counterfactual and original instance to ensure the feasibility of the counterfactual

In this case, sparsity is controlled on a post hoc basis. As a result, in this algorithm, you can select the features required to be changed to generate counterfactuals.

MOC (Multi-Objective Counterfactuals)

This algorithm tries to optimize most of the important properties. Its loss function involves the following components.

- The distance between desired class and modeled class of a counterfactual
- The similarity between a counterfactual and original instance
- The number of features being changed or sparsity
- Whether the counterfactual has likely feature values or a combination of values

Since the algorithm has all the important properties in its loss function, it produces very realistic counterfactuals. However, the time taken to generate counterfactuals for each instance is the longest for this algorithm as the objective function is too complicated to solve.

Comparison Between the Algorithms

This section compares various algorithms to determine and suggest to users a method to generate counterfactuals on their data. The comparison is made across various metrics such as time taken, sparsity of output, % of explanations generated, approach for categorical variables, similarity, and others. The metrics are mentioned in Table 11-1.

Table 11-1. Comparison of Various Methods of Generating Counterfactual Explanations from a Model

Metric/Algorithm	Counterfactual Explanations	Counterfactual Guided by Prototypes	MOC	DICE	CEML
Time per instance	7–12 mins	4 mins	30 mins	1 min	3–65 mins
Sparsity	High	High	Medium	Low	High
% counterfactual generated	28%	80%	70%	100%	100%
How does it tackle categorical variables?	Does not have any method to tackle	Has a method to tackle	Indicator function	Indicator function	Does not have any method to tackle
In data manifold?	No	Yes	Yes	No	No
Feature selection for perturbation	No	No	No	Yes	Yes
Similarity between original and counterfactual instance	Varies from one instance to another	Similar	Similar	Not Similar	Similar
Generates realistic counterfactuals	No	Yes	Yes	No	No
Generates multiple counterfactuals	No	No	Yes	Yes	No

The algorithms were tested on the same online shoppers' intention data set, mentioned at the start of Chapter 9. The machine was an 8 GB RAM Apple MacBook. No parallelization was applied. The results were generated instance by instance.

Notes:

- Time is tested on different ranges of probabilities (6 to 10 total instances were tested)

- Sparsity: High means it is changing lesser number of variables and vice-versa
- % reported as per the sample of ten observations used for testing
- Indicator function: any change in the value of the categorical variable is considered as distance 1

Based on the evaluation, we feel DiCE is an algorithm that can be used across many use cases. DiCE is covered next.

DiCE

Effective counterfactual explanations should satisfy two properties.

- The feasibility of the counterfactual actions given user context and constraints
- The diversity among the counterfactuals presented

Hence, the framework for generating and evaluating a diverse set of counterfactual explanations is based on determinantal point processes. To evaluate the actionability of counterfactuals, metrics should compare counterfactual-based methods to other local explanation methods.

The input is a trained machine learning model, f , and an instance, x . We would like to generate a set of k counterfactual examples, $\{c_1, c_2, \dots, c_k\}$, such that they all lead to a different decision than x .

The instance (x) and all CF examples ($\{c_1, c_2, \dots, c_k\}$) are d -dimensional. The machine learning model is assumed to be differentiable and static (does not change over time) and that the output is binary.

Our goal is to generate an actionable counterfactual set; that is, the user should find CF examples that they can act upon. To do so, we need individual CF examples to be feasible with respect to the original input but also need diversity among the generated counterfactuals to provide different ways of changing the outcome class. Diversity metrics to generate diverse counterfactuals can offer users multiple options.

At the same time, this method incorporates feasibility using the proximity constraint and introduces other user-defined constraints.

Diversity and Feasibility Constraints

Although diverse CF examples increase the chances that at least one example is actionable for the user, examples may change a large set of features or maximize diversity by considering big changes from the original input. This situation could be worsened when features are high-dimensional.

Diversity via determinantal point processes. The method captures diversity by building on determinantal point processes (DPP) adopted for solving subset selection problems with diversity constraints. The following are metrics based on the determinant of the kernel matrix given the counterfactuals.

$$\text{dpp_diversity} = \det(K),$$

$K_{ij} = \frac{1}{1 + \text{dist}(c_i, c_j)}$ and $\text{dist}(c_i, c_j)$ denotes a distance metric between the two counterfactual examples. In practice, to avoid ill-defined determinants, we add small random perturbations to the diagonal elements for computing the determinant.

Proximity

Intuitively, CF examples closest to the original input can be the most useful to a user. The method quantifies proximity as the (negative) vector distance between the original input and the CF example's features. This can be specified by a distance metric such as ℓ_1 -distance (optionally weighted by a user-provided custom weight for each feature). The proximity of a set of counterfactual examples is the mean proximity over the set.

$$\text{Proximity} := -\frac{1}{k} \sum_{i=1}^k \text{dist}(c_i, x).$$

Sparsity

Closely connected to proximity is the feasibility property of sparsity: the number of features a user needs to change to transition to the counterfactual class. Intuitively, a counterfactual example is more feasible if it makes changes to a fewer number of features. A counterfactual example may be close in feature space but may not be feasible due to real-world constraints. Thus, it makes sense to allow the user to

provide constraints on feature manipulation. They can be specified in two ways. First, as box constraints on feasible ranges for each feature, CF examples need to be searched. An example of such a constraint is “income cannot increase beyond \$200,000.” Alternatively, a user may specify the variables that can be changed.

Optimization

Based on the definitions of diversity and proximity, this method considers a combined loss function overall generated counterfactuals.

$$C(\mathbf{x}) = \arg \min_{\mathbf{c}_1, \dots, \mathbf{c}_k} \frac{1}{k} \sum_{i=1}^k \text{yloss}(f(\mathbf{c}_i), y) + \frac{\lambda_1}{k} \sum_{i=1}^k \text{dist}(\mathbf{c}_i, \mathbf{x}) - \lambda_2 \text{dpp_diversity}(\mathbf{c}_1, \dots, \mathbf{c}_k)$$

c_i is a counterfactual example (CF). k is the total number of CFs to be generated. $f(\cdot)$ is the ML model (a black box to end users). $\text{yloss}(\cdot)$ is a metric that minimizes the distance between $f(\cdot)$'s prediction for c_i s and the desired outcome y (usually 1 in our experiments). d is the total number of input features. x is the original input. $\text{dpp_diversity}(\cdot)$ is the diversity metric. λ_1 and λ_2 are hyperparameters that balance the three parts of the loss function. The method optimizes the preceding loss function using gradient descent. Ideally, we can achieve $f(c_i) = y$ for every counterfactual, but this may not always be possible because the objective is non-convex. Steps are run until the loss function converges, and the generated counterfactual is valid (belongs to the desired class).

Advantages

The interpretation of counterfactual explanations is very clear. If the feature values of an instance are changed according to the counterfactual, the prediction changes to the predefined prediction. There are no additional assumptions and no magic in the background. This also means it is not as dangerous as methods like LIME, where it is unclear how far we can extrapolate the local model for the interpretation.

The counterfactual method creates a new instance, but we can also summarize a counterfactual by reporting which feature values have changed. This gives two options for reporting results. You can either report the counterfactual instance or highlight which features have been changed between the instance of interest and the counterfactual instance.

The counterfactual method does not require access to the data or the model. It only requires access to the model's prediction function, which would also work via a web API. This is attractive for companies audited by third parties or offering explanations for users without disclosing the model or data. A company has an interest in protecting models and data because of trade secrets or data protection reasons. Counterfactual explanations offer a balance between explaining model predictions and protecting the interests of the model owner.

The method also works with systems that do not use machine learning. We can create counterfactuals for any system that receives inputs and returns outputs. The system that predicts apartment rents could also consist of handwritten rules, and counterfactual explanations would still work.

The counterfactual explanation method is relatively easy to implement since it is essentially a loss function (with a single or many objectives) that can be optimized with standard optimizer libraries. Additional details must be considered, such as limiting feature values to meaningful ranges (e.g., only positive apartment sizes).

Disadvantages

For each instance, you usually find multiple counterfactual explanations (Rashomon effect). This is inconvenient: most people prefer simple explanations over the complexity of the real world. It is also a practical challenge. Let's say we generated 23 counterfactual explanations for one instance. Are we reporting them all? Only the best? What if they are all relatively "good" but very different? These questions must be answered anew for each project. It can also be advantageous to have multiple counterfactual explanations because humans can select the ones that correspond to their previous knowledge.

Let's look at an example. Suppose that a hotel manager analyzes customer bookings and wishes to analyze which customers are more likely to cancel their booking. Figure 11-1 shows an instance where a customer did not cancel their hotel booking. According to this case, the customer belongs to the **Offline TA/TO** market segment. The customer type is **Contract**, the Distribution Channel is **TA/TO**, and Required Car Parking Spaces is **0**.

CHAPTER 11 DETAILING COUNTERFACTUAL METHODS

Diverse Counterfactuals found! total time taken: 00 min 05 sec Query instance (original outcome : 0)						
	LeadTime	MarketSegment	CustomerType	DistributionChannel	RequiredCarParkingSpaces	IsCanceled
0	93.0	Offline TA/TO	Contract	TA/TO	0.0	0.458217
Diverse Counterfactual set without sparsity correction since only metadata about each feature is available (new outcome : 1)						
	LeadTime	MarketSegment	CustomerType	DistributionChannel	RequiredCarParkingSpaces	IsCanceled
0	737.0	Complementary	Contract	TA/TO	0.0	0.844
1	191.0	Groups	Transient-Party	TA/TO	0.0	0.800
2	180.0	Offline TA/TO	Group	TA/TO	0.0	0.602
3	354.0	Groups	Transient	TA/TO	0.0	0.961

Figure 11-1. Sample counterfactual output

However, a hotel manager looking to minimize cancellation bookings would ideally like to identify the types of customers who would be likely to cancel their bookings. This is where DiCE comes in.

The following is observed when looking at the counterfactual explanations—instances where a customer did cancel their bookings.

- **Lead time** is significantly higher across the four examples.
- One customer still belongs to the **Offline TA/TO** market segment.
- One of the four cancellations is still a **contract** customer.
- The **distribution channel** and number of **required car parking spaces** remain the same.

Figure 11-2 is a sample of the data set.

Out[8]:	IsCanceled	LeadTime	MarketSegment	DistributionChannel	CustomerType	RequiredCarParkingSpaces
0	0	342	Direct	Direct	Transient	0
1	0	737	Direct	Direct	Transient	0
2	0	7	Direct	Direct	Transient	0
3	0	13	Corporate	Corporate	Transient	0
4	0	14	Online TA	TA/TO	Transient	0
...
40055	0	212	Offline TA/TO	TA/TO	Transient	0
40056	0	169	Direct	Direct	Transient-Party	0
40057	0	204	Direct	Direct	Transient	0
40058	0	211	Offline TA/TO	TA/TO	Contract	0
40059	0	161	Offline TA/TO	TA/TO	Transient	0

40060 rows × 6 columns

Figure 11-2. Data set for sample code

The continuous features in the data set are defined.

```
# Data set for training an ML model
d = dice_ml.Data(dataframe=data_set,
continuous_features=['LeadTime','RequiredCarParkingSpaces'],
outcome_name='IsCanceled')
```

DiCE can be installed from repo-<https://github.com/interpretml/DiCE>.

The neural network model is trained across **30** epochs using the **binary_crossentropy** loss and the Adam Optimizer. TensorFlow version 2.3.1 is used.

```
from tensorflow.keras.models import Sequential
from tensorflow.python.keras.layers import Dense
from tensorflow.python.keras.wrappers.scikit_learn import
KerasRegressor
sess = tf.InteractiveSession()
# Generating train and test data
train, _ = d.split_data(d.normalize_data(d.one_hot_encoded_data))
X_train = train.loc[:, train.columns != 'IsCanceled']
y_train = train.loc[:, train.columns == 'IsCanceled']# Fitting a dense
neural network model
ann_model = Sequential()
ann_model.add(Dense(6, input_shape=(X_train.shape[1],), activation=tf.
nn.relu))
ann_model.add(Dense(1, activation=tf.nn.sigmoid))
ann_model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
history=ann_model.fit(X_train, y_train, validation_split=0.20, epochs=30,
verbose=0, class_weight={0:1,1:2})
history
```

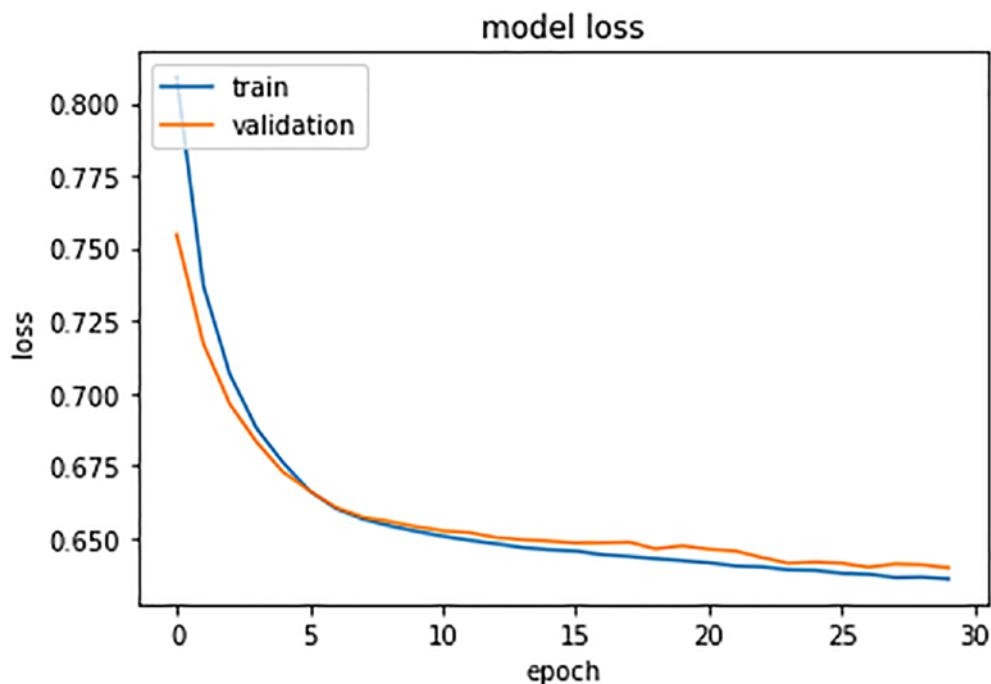


Figure 11-3. Plot of the model loss

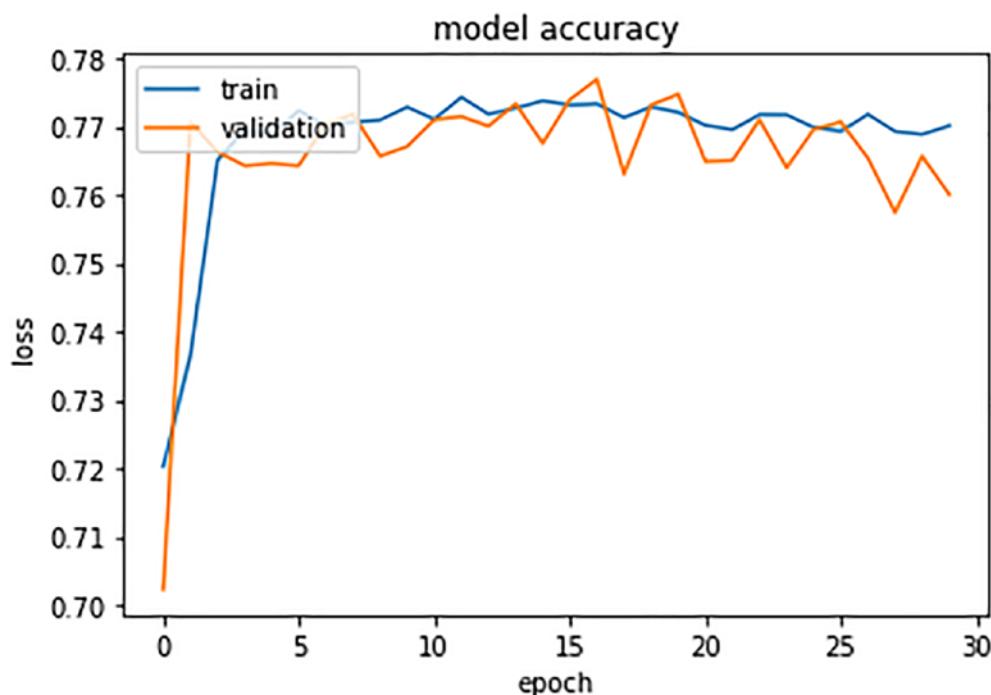


Figure 11-4. A plot of the training and validation accuracy

The model is now stored as a DiCE model for explanation purposes. The features are now stored in a special format compatible with DiCE to generate a DiCE explanation instance.

```
new_d = dice_ml.Data(features={
    'LeadTime':[0, 737],
    'MarketSegment': ['Complementary', 'Corporate', 'Direct', 'Groups',
    'Offline TA/T0', 'Online TA'],
    'CustomerType': ['Contract', 'Group', 'Transient', 'Transient-Party'],
    'DistributionChannel': ['Corporate', 'Direct', 'TA/T0', 'Undefined'],
    'RequiredCarParkingSpaces': [0, 8]},
    outcome_name='IsCanceled')
```

A DiCE explanation instance is formed.

```
>>> exp = dice_ml.Dice(new_d,m)
>>> exp<dice_ml.dice_interfaces.dice_tensorflow1.DiceTensorFlow1 at
0x7f22fb2da630>
```

Let's take an example of a customer with the following attributes, included in what is called a *query instance*.

```
query_instance = {'LeadTime': 68,
    'MarketSegment': 'Online TA',
    'CustomerType': 'Transient',
    'DistributionChannel': 'TA/T0',
    'RequiredCarParkingSpaces': 0}
```

Counterfactual examples are generated as follows.

```
# Generate counterfactual examples
dice_exp = exp.generate_counterfactuals(query_instance, total_CFs=4,
desired_class="opposite")
# Visualize counterfactual explanation
dice_exp.visualize_as_dataframe()
```

CHAPTER 11 DETAILING COUNTERFACTUAL METHODS

Diverse Counterfactuals found! total time taken: 00 min 05 sec						
Query instance (original outcome : 1)						
	LeadTime	MarketSegment	CustomerType	DistributionChannel	RequiredCarParkingSpaces	IsCanceled
0	68.0	Online TA	Transient	TA/TO	0.0	0.666167

Diverse Counterfactual set without sparsity correction since only metadata about each feature is available (new outcome : 0)						
	LeadTime	MarketSegment	CustomerType	DistributionChannel	RequiredCarParkingSpaces	IsCanceled
0	0.0	Online TA	Group	TA/TO	0.0	0.144
1	0.0	Online TA	Transient	Direct	7.0	0.000
2	70.0	Direct	Transient-Party	Corporate	2.0	0.000
3	737.0	Corporate	Transient	TA/TO	8.0	0.000

Figure 11-5. Output sample

Here, a case with an original outcome of **1** (i.e., a cancellation) is generated. It is also observed that counterexamples where a customer does not cancel their hotel booking are also generated. Conversely, here is a query instance where the customer **did** cancel their hotel booking.

```
query_instance_2 = {'LeadTime': 93,
'MarketSegment': 'Offline TA/TO',
'CustomerType': 'Contract',
'DistributionChannel': 'TA/TO',
'RequiredCarParkingSpaces': 0}
```

Again, counterfactual examples are generated.

```
# Generate counterfactual examples
dice_exp_2 = exp.generate_counterfactuals(query_instance_2, total_CFs=4,
desired_class="opposite")
# Visualize counterfactual explanation
dice_exp_2.visualize_as_dataframe()
```

Diverse Counterfactuals found! total time taken: 00 min 05 sec						
Query instance (original outcome : 0)						
	LeadTime	MarketSegment	CustomerType	DistributionChannel	RequiredCarParkingSpaces	IsCanceled
0	93.0	Offline TA/TO	Contract	TA/TO	0.0	0.458217

Diverse Counterfactual set without sparsity correction since only metadata about each feature is available (new outcome : 1)						
	LeadTime	MarketSegment	CustomerType	DistributionChannel	RequiredCarParkingSpaces	IsCanceled
0	737.0	Complementary	Contract	TA/TO	0.0	0.844
1	191.0	Groups	Transient-Party	TA/TO	0.0	0.800
2	180.0	Offline TA/TO	Group	TA/TO	0.0	0.602
3	354.0	Groups	Transient	TA/TO	0.0	0.961

Figure 11-6. Output sample

These are just some examples of how counterfactual explanations can be generated.

Summary

This chapter covered counterfactual explanations. The importance of counterfactual explanations comes into play when businesses look at the models to generate actionable insights. In business problems scenarios, not always the model results are intuitive. Most results are descriptive (i.e., good to have). Counterfactuals help derive actions. An insurance company manager who built a model to predict churn for the next premium cycle can utilize counterfactuals to understand each customer's changes to make them pay the premium. In such scenarios, counterfactuals find a lot of benefits. However, counterfactuals do not always give reasonable explanations.

You need to understand that counterfactuals run on the model's data. Most of the features in various data set are not in control of the business units. For example, when building a churn model, the features are age, income, pin code, financial status, previous premium paid, conversations with customer care, the name of the agent who sold the policy, or other policies. Even when we build a churn model and understand what the model is doing for each instance, we won't be able to change the age and income of the customer.

But factors like a conversation with customer care are a few features we can control. after running a counterfactual on this data, the company manager gets output in the form that "change customer care satisfaction score." This is an indication to assign this customer to the best executive. This might increase the chances of customers paying the premium in the next cycle. The field of counterfactual is not new; however, its implementation in the industry is very sparse. We recommend you find use cases within their organization or implement and test use cases of counterfactuals.

CHAPTER 12

Detailing Image Interpretability Methods

This chapter focuses on interpretation methods for image models. Deep learning has been very successful, especially in tasks that involve images and texts, such as image classification and language translation. The success story of deep neural networks began in 2012, when a deep learning approach won the ImageNet image classification challenge. Since then, we have witnessed an explosion of deep neural network architectures, with a trend toward deeper networks with more and more weight parameters.

To make image predictions with a neural network, the data input is passed through many layers of multiplication with the learned weights and through non-linear transformations. A single prediction can involve millions of mathematical operations depending on the architecture of the neural network. There is no chance that we humans can follow the exact mapping from data input to prediction. We would have to consider millions of weights that interact in a complex way to understand a prediction by a neural network.

To interpret the behavior and predictions of neural networks, we need specific interpretation methods. This book assumes that you are familiar with deep learning, including convolutional neural networks. You can certainly use model-agnostic methods, such as local models or partial dependence plots. Still, there are two reasons why it makes sense to consider interpretation methods developed specifically for neural networks. First, neural networks learn features and concepts in their hidden layers, and we need special tools to uncover them. Second, the gradient can be utilized to implement interpretation methods that are more computationally efficient than model-agnostic methods that look at the model “from the outside.” Also, most other methods in this book are intended for the interpretation of models for tabular data. Image and text data require different methods.

Image Interpretation Using LIME

Let's start with the LIME method for model explainability. It is a popular method because it can be used for tabular, image, and text data. Using an example, let's explore how LIME interprets images. The problem statement is shown in Figure 12-1.

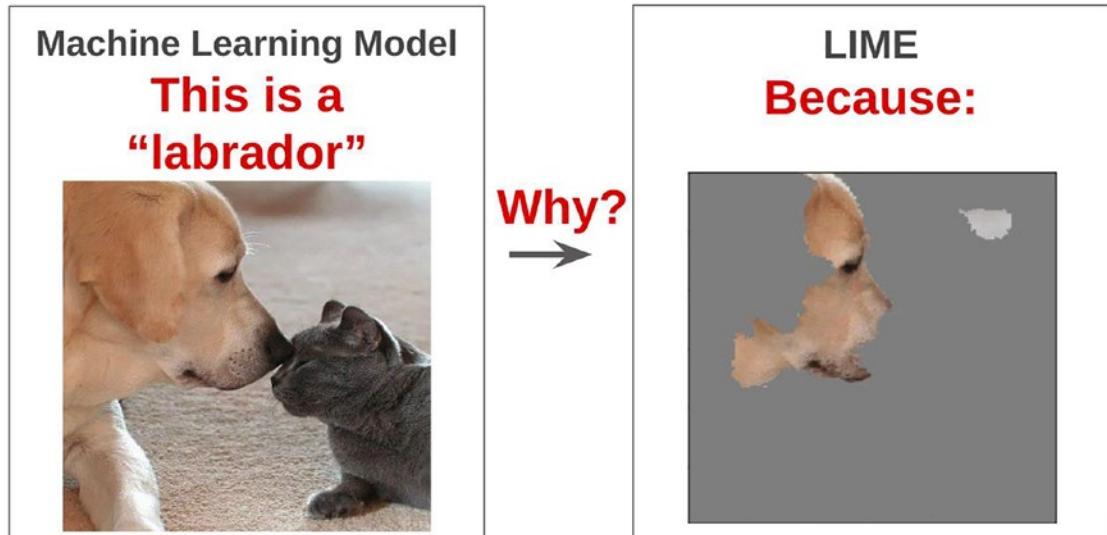


Figure 12-1. Problem description of image interpretation

LIME for images works differently than LIME for tabular data and text. Intuitively, it would not make much sense to perturb individual pixels since more than one pixel contributes to one class. Randomly changing individual pixels would probably not change the predictions by much. Therefore, variations of the images are created by segmenting the image into superpixels and turning them off or on. Superpixels are interconnected pixels with similar colors and can be turned off by replacing each pixel with a user-defined color such as gray. The user can also specify a probability for turning off a superpixel in each permutation.

Let's learn the concept of image interpretation using the following sample code. First, read the image and use the pretrained InceptionV3 model available in Keras to predict the image class.

```
# Read Image

import numpy as np
import skimage
Xi = skimage.io.imread("https://arteagac.github.io/blog/lime_image/img/cat-and-dog.jpg")
Xi = skimage.transform.resize(Xi, (299,299))
Xi = (Xi - 0.5)*2 #Inception pre-processing

skimage.io.imshow(Xi/2+0.5) # Show image before inception preprocessing

#Predict class for image using InceptionV3
import keras
from keras.applications.imagenet_utils import decode_predictions
np.random.seed(222)
inceptionV3_model = keras.applications.inception_v3.InceptionV3() #Load pretrained model
preds = inceptionV3_model.predict(Xi[np.newaxis,:,:,:])
top_pred_classes = preds[0].argsort()[-5:][::-1] # Save ids of top 5 classes
decode_predictions(preds)[0] #Print top 5 classes
```

This script loads the input image in the Xi variable and prints the top five classes (and probabilities) for the image, as shown in Figure 12-2.

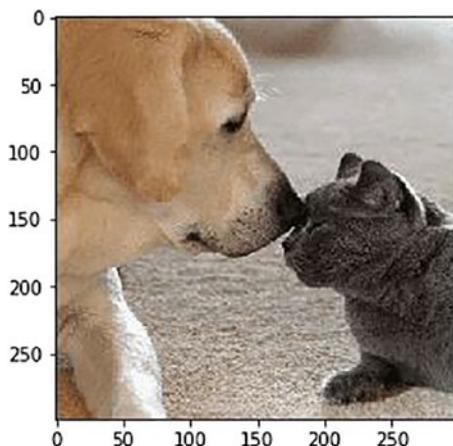


Figure 12-2. Image used for the problem

- Labrador Retriever (82.2%)
- Golden Retriever (1.5%)
- American Staffordshire Terrier (0.9%)
- Bull Mastiff (0.8%)
- Great Dane (0.7%)

With this information, the input image, and the pretrained InceptionV3 model, we can generate explanations with LIME. This example generates explanations for the Labrador Retriever class.

LIME creates explanations by generating a new data set of random perturbations (with their respective predictions) around the instance being explained and then fitting a weighted local surrogate model. This local model is usually a simpler model with intrinsic interpretability, such as a linear regression model.

Step 1. Generate Random Perturbations for Input Image

For images, LIME generates perturbations by turning on and off some of the superpixels in the image. The following script uses the quick-shift segmentation algorithm to compute the superpixels in the image. In addition, it generates an array of 150 perturbations, where each is a vector with zeros and ones that represent whether the superpixel is on or off.

```
#Generate segmentation for image
import skimage.segmentation
superpixels = skimage.segmentation.quickshift(Xi, kernel_size=4,max_
dist=200, ratio=0.2)
num_superpixels = np.unique(superpixels).shape[0]
skimage.io.imshow(skimage.segmentation.mark_boundaries(Xi/2+0.5,
superpixels))

#Generate perturbations
num_perturb = 150
perturbations = np.random.binomial(1, 0.5, size=(num_perturb, num_
superpixels))
```

```
#Create function to apply perturbations to images
import copy
def perturb_image(img,perturbation,segments):
    active_pixels = np.where(perturbation == 1)[0]
    mask = np.zeros(segments.shape)
    for active in active_pixels:
        mask[segments == active] = 1
    perturbed_image = copy.deepcopy(img)
    perturbed_image = perturbed_image*mask[:, :, np.newaxis]
    return perturbed_image

#Show example of perturbations
print(perturbations[0])
skimage.io.imshow(perturb_image(Xi/2+0.5,perturbations[0],superpixels))
```

Computing the superpixels in the image results in what's shown in Figure 12-3.

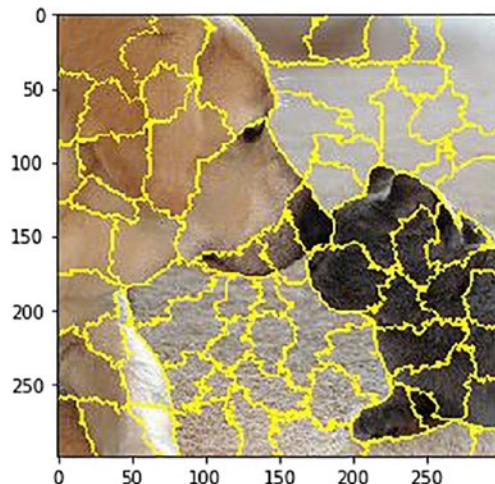


Figure 12-3. Output of superpixels

Figure 12-4 shows examples of perturbation vectors and perturbed images.

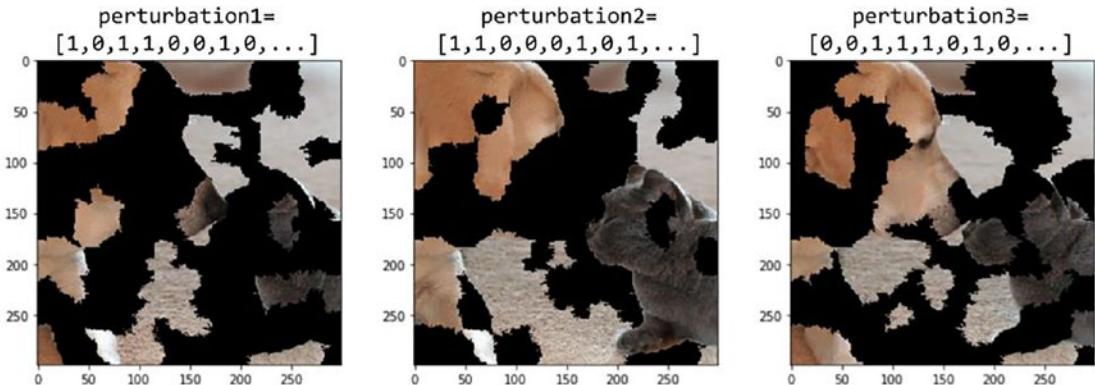


Figure 12-4. Output of the perturbed images

Step 2. Predict Class for Perturbations

The following script uses the `inceptionV3_model` to predict the class of each of the perturbed images. The shape of the predictions is (150,1000), which means that for each of the 150 images, there is the probability of belonging to the 1,000 classes in InceptionV3. From these 1,000 classes, we use only the Labrador class in further steps since it is the prediction we want to explain. In this example, 150 perturbations were used. However, for real applications, a larger number of perturbations produce more reliable explanations.

```
predictions = []
for pert in perturbations:
    perturbed_img = perturb_image(Xi,pert,superpixels)
    pred = inceptionV3_model.predict(perturbed_img[np.newaxis,:,:,:])
    predictions.append(pred)
predictions = np.array(predictions)
print(predictions.shape)
```

Now we have everything to fit a linear model using the perturbations as input features X and the predictions for Labrador `predictions[labrador]` as output y. However, before we fit a linear model, LIME needs to give more weight (importance) to images that are closer to the image being explained.

Step 3. Compute Weights (Importance) For the Perturbations

We use a distance metric to evaluate how far each perturbation is from the original image. The original image is a perturbation with all the superpixels active (all elements in one). Given that the perturbations are multidimensional vectors, the cosine distance is a metric that can be used for this purpose. After the cosine distance has been computed, a kernel function translates distance to a value between 0 and 1 (a weight). At the end of this process, we weight (importance) each perturbation in the data set.

```
#Compute distances to original image
import sklearn.metrics
original_image = np.ones(num_superpixels)[np.newaxis,:,:] #Perturbation with
all superpixels enabled
distances = sklearn.metrics.pairwise_distances(perturbations,original_
image, metric='cosine').ravel()
print(distances.shape)

#Transform distances to a value between 0 an 1 (weights) using a kernel
function
kernel_width = 0.25
weights = np.sqrt(np.exp(-(distances**2)/kernel_width**2)) #Kernel function
print(weights.shape)
```

Step 4. Fit an Explainable Linear Model Using the Perturbations, Predictions, and Weights

We fit a weighted linear model using the information obtained in the previous steps. We get a coefficient for each superpixel in the image that represents how strong is the effect of the superpixel in the Labrador Retriever prediction.

```
#Estimate linear model
from sklearn.linear_model import LinearRegression
class_to_explain = top_pred_classes[0] #Labrador class
simpler_model = LinearRegression()
simpler_model.fit(X=perturbations, y=predictions[:, :, class_to_explain],
sample_weight=weights)
```

```

coeff = simpler_model.coef_[0]

#Use coefficients from linear model to extract top features
num_top_features = 4
top_features = np.argsort(coeff)[-num_top_features:]

#Show only the superpixels corresponding to the top features
mask = np.zeros(num_superpixels)
mask[top_features]= True #Activate top superpixels
skimage.io.imshow(perturb_image(Xi/2+0.5,mask,superpixels))

```

We need to sort these coefficients to determine the most important superpixels (`top_features`) for the Labrador Retriever prediction. Even though we used the magnitude of the coefficients to determine the most important features, other alternatives such as forward or backward elimination can be used for feature importance selection.

Figure 12-5 shows the top superpixels after computing.

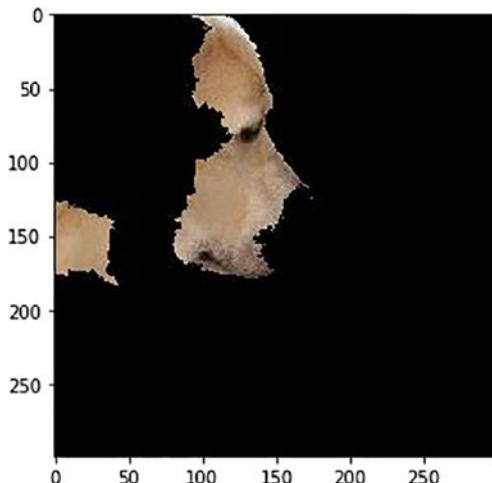


Figure 12-5. Top superpixels of the input image

This is what LIME returns as an explanation. The area of the image (superpixels) that have a stronger association with the Labrador Retriever prediction. This explanation suggests that the pretrained InceptionV3 model is doing a good job predicting the Labrador Retriever class for the given image. This example shows how LIME can increase confidence in a machine learning model by understanding why certain predictions are returned.

Image Interpretation Using Pixel Attribution (Saliency Maps)

Pixel attribution methods highlight the pixels that were relevant for certain image classification by a neural network. Figure 12-6 is an example of an explanation.

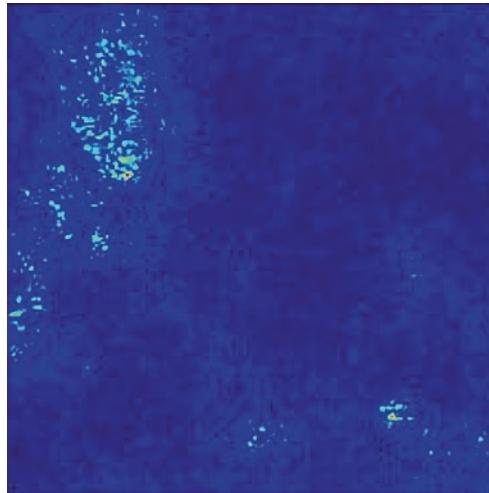


Figure 12-6. Pixel attribution sample image

Pixel attribution methods can be found under various names: sensitivity map, saliency map, pixel attribution map, gradient-based attribution methods, feature relevance, feature attribution, and feature contribution.

Pixel attribution is a special case of feature attribution for images. Feature attribution explains individual predictions by attributing each input feature according to how it changed the prediction (negatively or positively). The features can be input pixels, tabular data, or words. SHAP, Shapley values, and LIME are examples of general feature attribution methods. We consider neural networks that output as prediction a vector of length, C, which includes regression where C=1. The output of the neural network for image, I, is called $S(I)=[S_1(I), \dots, S_C(I)]$. All these methods take as input $x \in R^p$ (can be image pixels, tabular data, words, ...) with p features and output as explanation a relevance value for each of the p input features: $R_c = [R_{c1}, \dots, R_{cp}]$. The c indicates the relevance for the c-th output $S_C(I)$.

There is a confusing amount of pixel attribution approaches. It helps to understand that there are two different types of attribution methods.

- **Occlusion- or perturbation-based:** Methods like SHAP and LIME manipulate parts of the image to generate explanations (model-agnostic).
- **Gradient-based:** Many methods compute the prediction gradient (or classification score) with respect to the input features. The gradient-based methods (of which there are many) mostly differ in how the gradient is computed.

Both approaches have in common that the explanation has the same size as the input image (or can be meaningfully projected onto it). They assign each pixel a value that can be interpreted as the relevance of the pixel to the prediction or classification of that image.

Another useful categorization for pixel attribution methods is the baseline question.

Gradient-only methods tell whether a change in a pixel would change the prediction. Vanilla Gradient and Grad-CAM are two examples. The interpretation of the gradient-only attribution is: If we were to change this pixel, the predicted class probability would go up (for positive gradient) or down (for negative gradient). The larger the absolute value of the gradient, the stronger the effect of a change at this pixel.

The next few sections explain image interpretation with activation maps.

Image Interpretation Using Class Activation Maps

Class activation maps (CAM) were introduced in the paper “Learning Deep Features for Discriminative Localization” using global average pooling in CNNs. A CAM in a particular category indicates the discriminative region used by CNN to identify the category.

It has been observed that convolution units of various layers of a convolutional neural network act as an object detector even though no such prior about the location of the object is provided while training the network for a classification task. Even though convolution has this remarkable property, it is lost when using a fully connected layer for the classification task. To avoid using a fully connected network, some architectures like Network in Network (NiN) and GoogLeNet are fully convolutional neural networks. Global *average pooling* (GAP) is a commonly used layer in such architectures. It is

mainly used as a regularizer to prevent overfitting while training. The class activation map simply indicates the discriminative region in the image, which the CNN uses to classify that image in a particular category. For this technique, the network consists of ConvNet, and right before the softmax layer (for multiclass classification), global average pooling is performed on the convolutional feature maps. The output of this layer is used as a feature for a fully connected layer that produces the desired classification output. Given this simple connectivity structure, we can identify the importance of the image regions by projecting back the weights of the output layer onto the convolutional feature maps.

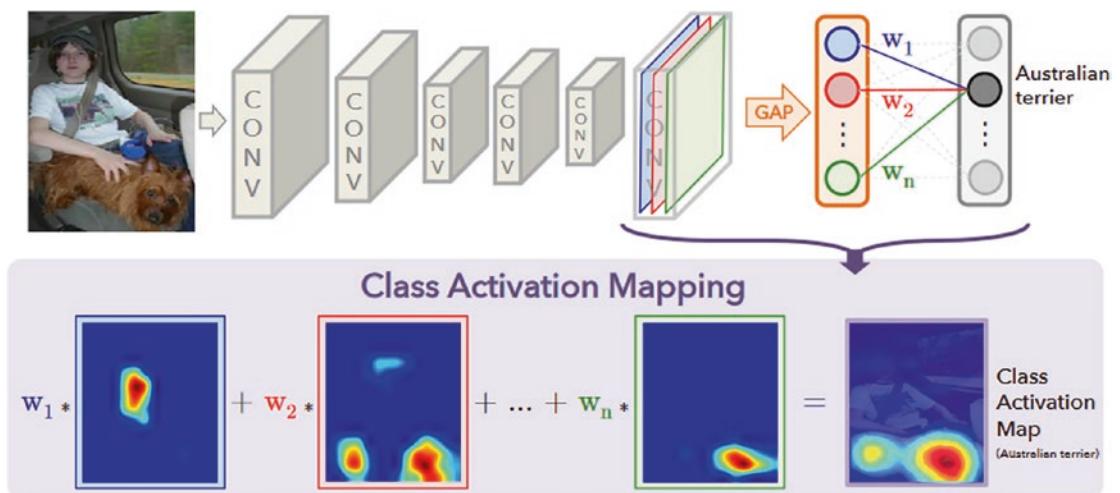


Figure 12-7. The sample network architecture ideal for CAM

The method to implement this is detailed in the following steps.

Step 1. Modify the Model

Suppose you have built your deep classifier with Conv blocks and a few fully connected layers. You must modify this architecture such that there aren't any fully connected layers. Use the GlobalAveragePooling2D layer between the output layer (softmax/sigmoid) and the last convolutional block. The CAM model provides a required modification to the cat and dog classifier. Here we used pretrained VGG16 model to simulate the trained classifier.

```
def CAMmodel():
    ## Simulating my pretrained dog and cat classifier.
    vgg = VGG16(include_top=False, weights='imagenet')
    vgg.trainable = False
    ## Flatten the layer so that it's not nested in the sequential model.
    vgg_flat = flatten_model(vgg)
    ## Insert GAP
    vgg_flat.append(keras.layers.GlobalAveragePooling2D())
    vgg_flat.append(keras.layers.Dense(1, activation='sigmoid'))

    model = keras.models.Sequential(vgg_flat)
    return model
```

A simple utility flatten_model returns the list of layers in my pretrained model. This is done so that the layers are not nested when modified using the Sequential model, and the last convolutional layer can be accessed and used as an output. We appended GlobalAveragePooling2D and Dense in the returned array from flatten_model. Finally, the Sequential model is returned.

```
def flatten_model(model_nested):
    ...
    Utility to flatten pretrained model
    ...

    layers_flat = []
    for layer in model_nested.layers:
        try:
            layers_flat.extend(layer.layers)
        except AttributeError:
            layers_flat.append(layer)

    return layers_flat
```

Next, we call `model.build()` with the appropriate model input shape.

```
keras.backend.clear_session()
model = CAMmodel()
model.build((None, None, None, 3)) # Note
model.summary()
```

Step 2. Retrain the Model with CAMLogger Callback

Since a new layer was introduced, we must retrain the model. But we don't need to retrain the entire model. We can freeze the convolutional blocks by using `vgg.trainable=False`. When we check model performance, we see a decline in the model performance in terms of both training and validation accuracy. Thus, for the implementation of CAM, we must modify our architecture and thus a decline in model performance.

Step 3. Use CAMLogger to See the Class Activation Map

In the `__init__` for the CAM class, we initialize `cammodel`. Notice there are two outputs from this CAM model.

Output from the last convolutional layer (`block5_conv3`). The model prediction (softmax/sigmoid).

```
class CAM:
    def __init__(self, model, layerName):
        self.model = model
        self.layerName = layerName
        ## Prepare cammodel
        last_conv_layer = self.model.get_layer(self.layerName).output
        self.cammodel = keras.models.Model(inputs=self.model.input,
                                           outputs=[last_conv_layer, self.
                                           model.output])

    def compute_heatmap(self, image, classIdx):
        ## Get the output of last conv layer and model prediction
        [conv_outputs, predictions] = self.cammodel.predict(image)
        conv_outputs = conv_outputs[0, :, :, :]
        conv_outputs = np.rollaxis(conv_outputs, 2)
        ## Get class weights between
        class_weights = self.model.layers[-1].get_weights()[0]
        ## Create the class activation map.
        caml = np.zeros(shape = conv_outputs.shape[1:3], dtype=np.float32)
        for i, w in enumerate(class_weights[:]):
            caml += w * conv_outputs[i, :, :]      caml /= np.max(caml)
```

```

cam1 = cv2.resize(cam1, (image.shape[1], image.shape[2]))
## Prepare heatmap
heatmap = cv2.applyColorMap(np.uint8(255*cam1), cv2.COLORMAP_JET)
heatmap[np.where(cam1 < 0.2)] = 0    return heatmap

def overlay_heatmap(self, heatmap, image):
    img = heatmap*0.5 + image
    img = img*255
    img = img.astype('uint8')
    return (heatmap, img)

```

The compute_heatmap method is responsible for generating the heatmap, which is the discriminative region used by CNN to identify the category (class of image).

- cammodel.predict() on the input image gives the feature map of the last convolutional layer of shape (1,7,7,512).
- We also extract the weights of the output layer of shape (512,1).
- The dot product of the extracted weights from the final layer and the feature map is calculated to produce the class activation map.

Now let's wrap everything in a callback. The CAMLogger callback integrates wandb.log() method to log the generated activation maps onto the W&B run page. The heatmap returned from the CAM is finally overlaid on the original image by calling overlay_heatmap() method.

Step 4. Draw Conclusions from the CAM

You can draw a lot of conclusions from the plots shown in Figure 12-8. If the prediction score is greater than 0.5, the network classifies the image as a dog; otherwise, it is classified as a cat. CAM charts have their corresponding class activation maps.

Let's go through some observations.

- The model classifies the images as dogs by looking at the facial region in the image. In some images, it's able to look at the entire body, except the paws.
- The model classifies the images as cats by looking at the ears, paws, and whiskers.

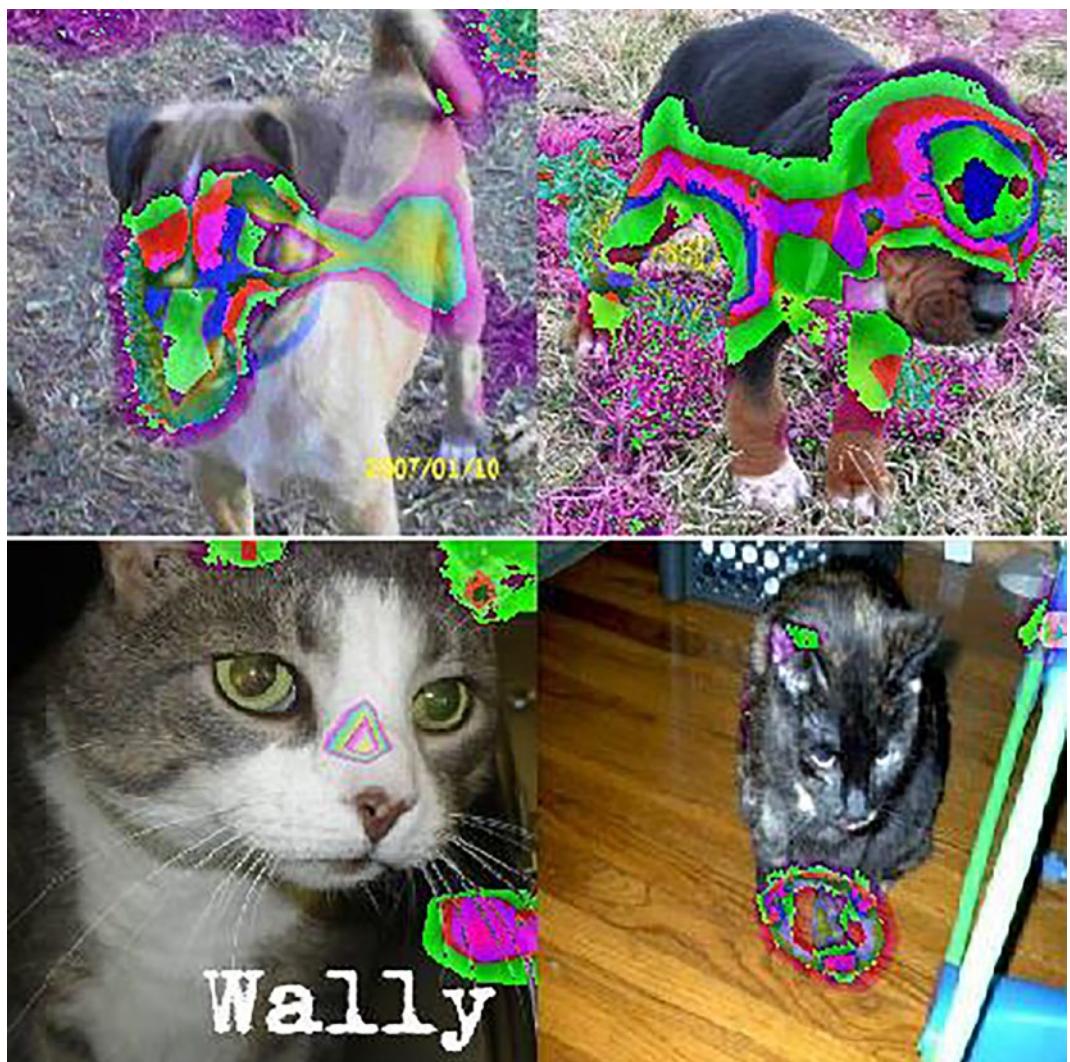


Figure 12-8. Output of the CAM model looking at dogs' faces and cats' whiskers, paws, and ears. In a misclassified image, the model is not looking where it should be looking. Thus by using CAM, we are able to interpret the reason behind this misclassification, which is really cool

Image Interpretation Using Gradient-Weighted Class Activation Maps

Grad-CAM is a popular technique for visualizing where a convolutional neural network model is looking. Grad-CAM is class-specific, meaning it can produce a separate visualization for every class present in the image, as shown in Figure 12-9.

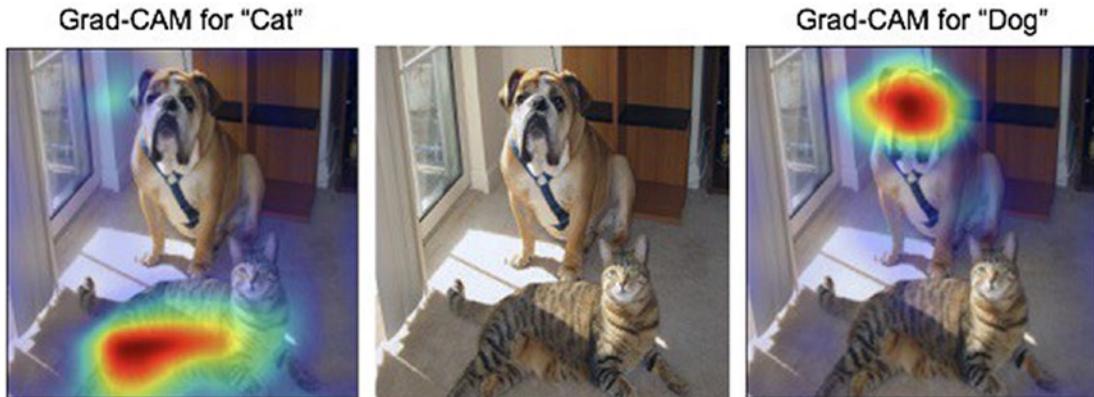


Figure 12-9. Sample Grad-CAM image

Grad-CAM is a popular technique for visualizing where a convolutional neural network model is looking. It is class-specific, meaning it can produce a separate visualization for every class present in the image. Grad-CAM can be used for weakly-supervised localization—that is, determining the location of objects using a model trained only on whole-image labels rather than explicit location annotations. It can also be used for weakly-supervised segmentation. The model predicts all the pixels that belong to particular objects without requiring pixel-level labels for training. Finally, Grad-CAM can be used to better understand a model, for example, by providing insight into model failure. The following are a few noteworthy points about Grad-CAM.

- **Grad-CAM as post hoc attention:** Grad-CAM is a form of post hoc attention, meaning that it is a method for producing heatmaps that are applied to a trained neural network, and the parameters are fixed. This is distinct from trainable attention, which involves learning how to produce attention maps (heatmaps) during training by learning parameters.

- Grad-CAM as a **generalization** of CAM: Grad-CAM does not require a particular CNN architecture. Grad-CAM is a generalization of CAM, a method that does require using a particular architecture. CAM requires an architecture that applies GAP to the final convolutional feature maps, followed by a single fully connected layer that produces the predictions.

The basic idea behind Grad-CAM is the same as the basic idea behind CAM: we want to exploit the spatial information preserved through convolutional layers to understand which parts of an input image were important for a classification decision. Similar to CAM, Grad-CAM uses the feature maps produced by the last convolutional layer of a CNN. The authors of Grad-CAM argue, “We can expect the last convolutional layers to have the best compromise between high-level semantics and detailed spatial information.”

Figure 12-10 illustrates the parts of a neural network model relevant to Grad-CAM.

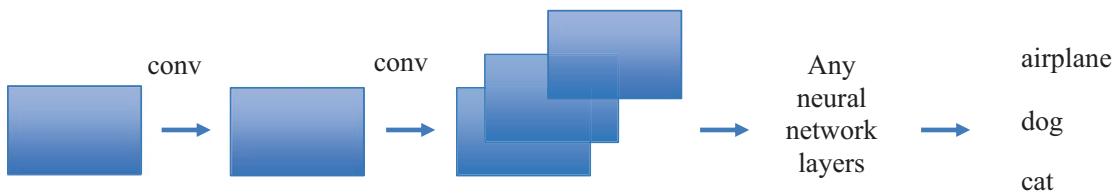


Figure 12-10. Illustration of Grad-CAM flow

The CNN is composed of some convolutional layers (shown as “conv” in the sketch). The feature maps produced by the final convolutional layer are shown as A1, A2, and A3, the same as in the CAM sketch. At this point, for CAM, we would need to do global average pooling followed by a fully connected layer. For Grad-CAM, we can do anything—for example, multiple fully connected layers, which is “any neural network layers” in the sketch. The only requirement is that the layers we insert after A1, A2, and A3 must be differentiable to get a gradient. Finally, there are the classification outputs for *airplane*, *dog*, *cat*, *person*, and so forth.

The difference between CAM and Grad-CAM is how the feature maps A1, A2, and A3 are weighted to make the final heatmap. In CAM, we weigh these feature maps using weights taken out of the last fully connected layer of the network. In Grad-CAM, we weight the feature maps using “alpha values” that are calculated based on gradients. Therefore, Grad-CAM does not require a particular architecture because we can calculate gradients through any neural network layer we want. The *Grad* in Grad-CAM stands for *gradient*.

The output of Grad-CAM is a *class-discriminative localization map*, a heatmap where the hot part corresponds to a particular class. Grad-CAM uses the gradients of any target concept (e.g., logits for *dog* or even a caption), flowing into the final convolutional layer to produce a coarse localization map highlighting the important regions in the image for predicting the concept.

Using Grad-CAM, you can visually validate where your network is looking, verifying that it is indeed looking at the correct patterns in the image and activating around those patterns.

If the network is *not* activating around the proper patterns/objects in the image, then you know the following.

- Your network hasn't properly learned the underlying patterns in your data set.
- Your training procedure needs to be revisited.
- You may need to collect additional data.
- And most importantly, *your model is not ready for deployment*.

Grad-CAM is a tool that should be in any deep learning practitioner's toolbox. Deep learning models are often criticized for being black-box algorithms where we don't know what is going on under the hood. Using a gradient camera (i.e., Grad-CAM), deep learning practitioners can visualize CNN layer activation heatmaps with Keras/TensorFlow. Visualizations like this allow you to peek at what the "black box" is doing.

To use the Grad-CAM implementation, you need to configure your system with a few software packages, including TensorFlow (2.0 recommended), OpenCV, or Imutils.

Let's discuss the implementation of Grad-CAM now.

To begin, we first need a model to run the forward pass. We use VGG16 pretrained on ImageNet. You can use any model because Grad-CAM, unlike CAM, doesn't require a specific architecture and is compatible with any convolutional neural network.

```
model = VGG16(weights='imagenet')
```

After we define the model, we load a sample image (see Figure 12-11) and preprocess it so that it is compatible with the model.

```
def preprocess(img):
    img = img_to_array(img)
    img = np.expand_dims(img, axis=0)
```

```

img = preprocess_input(img)
return img
image_1 = preprocess(image)

```

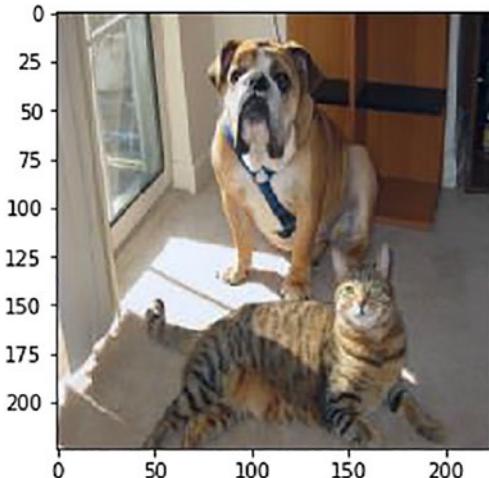


Figure 12-11. Input image for Grad-CAM

Then we use the model to make predictions on the sample image and decode the top three predictions. As you can see in the following images, we are considering the top three predictions from the model. The top model prediction is for a boxer.

```

predict = model.predict(image_1)
print(decode_predictions(predict,top=3))
target_class = np.argmax(predict[0])
print("Target Class = %d"%target_class)

```

```
[[('n02108089', 'boxer', 0.29122108), ('n02108422', 'bulldog', 0.19912729), ('n02129604', 'tiger', 0.1005028)]]
Target Class = 242
```

In the next step, we find the gradients of the target class score y_c with respect to the feature maps A_k of the last convolutional layer. Intuitively it tells how important each channel is regarding the target class. The grads variable returns a tensor, which is used in the following steps.

```

last_conv = model.get_layer('block5_conv3')
grads = K.gradients(model.output[:,242],last_conv.output)[0]

```

The gradients thus obtained are then global average pooled to obtain the neuron important weights a_k corresponding to the target class. This returns a tensor that is passed to the Keras function that takes the image as input and returns the pooled_grads along with activation maps from the last convolution layer.

```
pooled_grads = K.mean(grads, axis=(0,1,2))
iterate = K.function([model.input], [pooled_grads, last_conv.output[0]])
pooled_grads_value, conv_layer_output = iterate([image_1])
```

After that, we multiply each activation map with corresponding pooled gradients which acts as weights determining how important each channel is with regard to the target class. We then take the mean of all the activation maps along the channels and the result obtained is the final class-discriminative saliency map (see Figure 12-12).

```
for i in range(512):
    conv_layer_output[:, :, i] *= pooled_grads_value[i]
heatmap = np.mean(conv_layer_output, axis=-1)
```

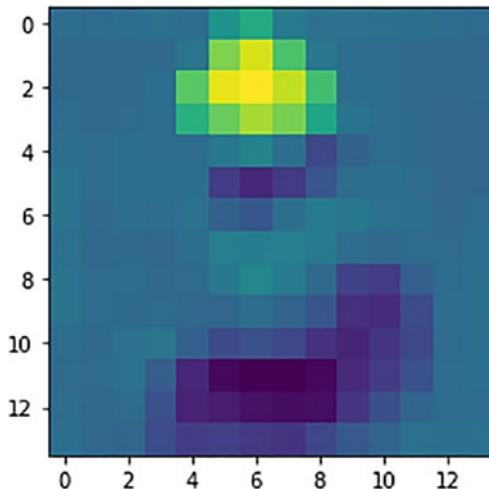


Figure 12-12. Class-discriminative map

Then we apply ReLU on the resulting heatmap to only keep the features that positively influence the output map. But we see that we don't have many negative intensities in the heatmap, and hence there isn't much change in the heatmap after applying ReLU (see Figure 12-13).

```
for x in range(heatmap.shape[0]):  
    for y in range(heatmap.shape[1]):  
        heatmap[x,y] = np.max(heatmap[x,y],0)
```

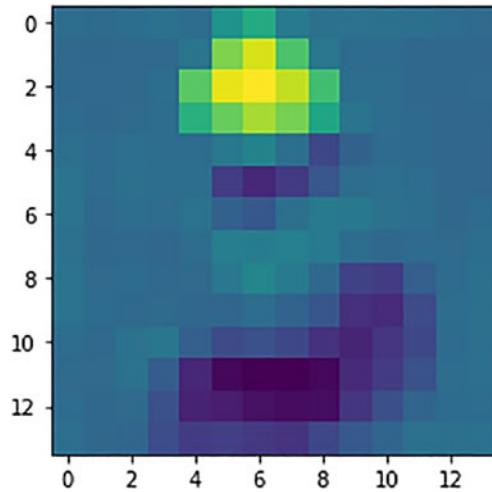


Figure 12-13. Class-discriminative map after ReLU

We then divide each intensity value of the heatmap with the maximum intensity value to normalize the heatmap such that all values fall between 0 and 1 (see Figure 12-14).

```
heatmap = np.maximum(heatmap,0)  
heatmap /= np.max(heatmap)  
plt.imshow(heatmap)
```

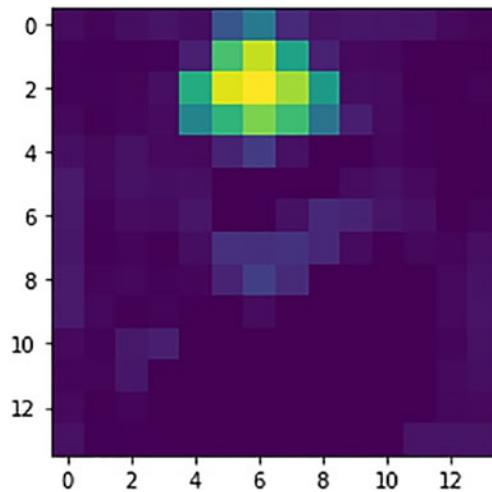


Figure 12-14. Class-discriminative saliency map after normalization

Finally, we upsample the resulting heatmap to match the dimensions of the input image and overlay it on the input image in order to see the results, as shown in Figure 12-15.

```
upsample = resize(heatmap, (224,224),preserve_range=True)
plt.imshow(image)
plt.imshow(upsample,alpha=0.5)
plt.show()
```

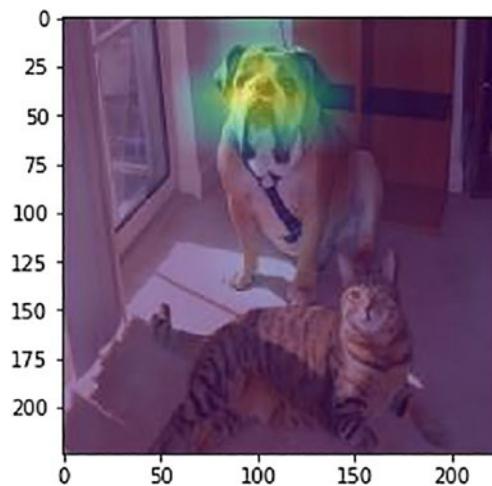


Figure 12-15. Final saliency map with bulldog as the target class

Summary

In this chapter, you learned a new technique for interpreting convolutional neural networks, which is state-of-the-art architecture, especially for image-related tasks. Grad-CAM improves on its predecessor, CAM, and provides better localization and clear class-discriminative saliency maps, which guide and demystify the complexity behind the black-box-like models. The research in interpretable machine learning is advancing at a faster pace and is proving to be crucial in building customer trust and improving the models. You also studied how to interpret images using basic techniques such as LIME.

The next chapter discusses text interpretation methods.

CHAPTER 13

Explaining Text Classification Models

Text documents can be described by several abstract concepts such as semantic category, writing style, or sentiment. Machine learning (ML) models have been trained to automatically map documents to these abstract concepts, allowing to annotate very large text collections, more than could be processed by a human in a lifetime. Besides predicting the text's category very accurately, it is also highly desirable to understand how and why the categorization process takes place.

In the field of natural language processing models, it's always the case that the dimension of the features is huge, and hence explaining the models using feature importance becomes much more complicated. Techniques like LIME and SHAP explain how an NLP model works and makes decisions to end users and data science practitioners. This chapter uses a few examples to see how we can utilize the two mentioned methods to explain a model built on text data.

To start, let's quickly build a text classification model. We do not discuss the model because it is out of scope for this chapter. We build the model and then quickly dive into generating explanations to explain the model decisions.

We used the Stack Overflow tags classification data set. The following is a brief description of the data set.

- Question ID
- Creation date
- Closed date, if applicable
- Score
- Owner user ID

- Number of answers
- Tags

This data set was extracted from the Stack Overflow database. It contains 12,583,347 non-deleted questions and 3,654,954 deleted ones.

This is all public data within the Stack Exchange data dump, which is much more comprehensive (including question and answer text) and requires much more computational overhead to download and process. This data set is designed to be easy to read in and start analyzing.

Data Preprocessing, Feature Engineering, and Logistic Regression Model on the Data

The following are the different imports required for building this model.

```
import pandas as pd
import numpy as np
import sklearn
import sklearn.ensemble
import sklearn.metrics
from sklearn.utils import shuffle
from __future__ import print_function
from io import StringIO
import re
from bs4 import BeautifulSoup
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score, precision_score,
recall_score
import lime
from lime import lime_text
from lime.lime_text import LimeTextExplainer
from sklearn.pipeline import make_pipeline
```

Now read the data set and perform some data modifications required specifically for the model. You can follow the same code to reproduce results in your own system or environment.

```
df = pd.read_csv('stack-overflow-data.csv')

df = df[pd.notnull(df['tags'])]
df = df.sample(frac=0.5, random_state=99).reset_index(drop=True)
df = shuffle(df, random_state=22)
df = df.reset_index(drop=True)
df['class_label'] = df['tags'].factorize()[0]
class_label_df = df[['tags', 'class_label']].drop_duplicates().sort_
values('class_label')
label_to_id = dict(class_label_df.values)
id_to_label = dict(class_label_df[['class_label', 'tags']].values)

REPLACE_BY_SPACE_RE = re.compile('[/(){}\\[\\]\\|@,;]')
BAD_SYMBOLS_RE = re.compile('[^0-9a-z #+_]')
# STOPWORDS = set(stopwords.words('english'))

def clean_text(text):
    """
    text: a string
    return: modified initial string
    """
    text = BeautifulSoup(text, "lxml").text # HTML decoding.
    BeautifulSoup's text attribute return a string stripped of any
    HTML tags and metadata.
    text = text.lower() # lowercase text
    text = REPLACE_BY_SPACE_RE.sub(' ', text) # replace REPLACE_BY_
    SPACE_RE symbols by space in text. substitute the matched string
    in REPLACE_BY_SPACE_RE with space.
    text = BAD_SYMBOLS_RE.sub('', text) # remove symbols which are in
    BAD_SYMBOLS_RE from text. substitute the matched string in BAD_
    SYMBOLS_RE with nothing.
```

```

#     text = ' '.join(word for word in text.split() if word not in
#                      STOPWORDS) # remove stopwors from text
#     return text

df['post'] = df['post'].apply(clean_text)

list_corpus = df["post"].tolist()
list_labels = df["class_label"].tolist()
X_train, X_test, y_train, y_test = train_test_split(list_corpus, list_
labels, test_size=0.2, random_state=40)

vectorizer = CountVectorizer(analyzer='word', token_pattern=r'\w{1,}', 
ngram_range=(1, 3), stop_words = 'english', binary=True)
train_vectors = vectorizer.fit_transform(X_train)
test_vectors = vectorizer.transform(X_test)

logreg = LogisticRegression(n_jobs=1, C=1e5)
logreg.fit(train_vectors, y_train)
pred = logreg.predict(test_vectors)
accuracy = accuracy_score(y_test, pred)
precision = precision_score(y_test, pred, average='weighted')
recall = recall_score(y_test, pred, average='weighted')
f1 = f1_score(y_test, pred, average='weighted')

```

Interpreting Text Predictions with LIME

This section explains a text model using a popular technique called LIME, which you studied in Chapter 9. It is an interpretability surrogate model used on black-box models (model-agnostic). It provides interpretability for a single observation prediction (local). To recap, let's see how LIME works.

1. Choose the single prediction.
2. LIME creates permutations of your data at this instance and collects the black-box model results.

3. It then gives weights to the new samples based on how closely they match the data of the original prediction.
4. A new, less complex, interpretable model is trained on the data variations created using the weights attached to each variation.
5. This local interpretable model can explain the prediction.

With LIME, you can create visualizations like the following to help you understand which words in the text have the greatest influence on the model's final prediction. In Figure 13-1, the words *killed*, *crash*, and *helicopter* all contribute to the final prediction score of 0.89, making the model quite confident in its disaster class prediction.

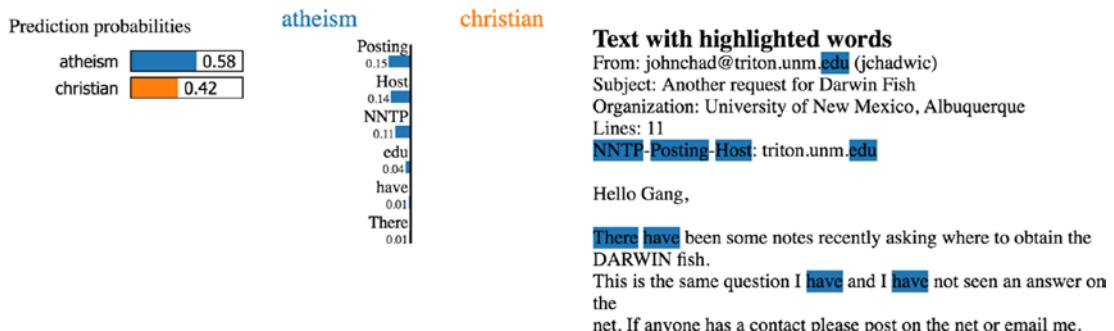


Figure 13-1. A description of how LIME works for text data

Now that you understand how LIME works for text data, let's proceed to the code snippet that can help implement LIME.

```
c = make_pipeline(vectorizer, logreg)
class_names=list(df.tags.unique())
explainer = LimeTextExplainer(class_names=class_names)

idx = 1877
exp = explainer.explain_instance(X_test[idx], c.predict_proba, num_
features=6, labels=[4, 8])
print('Document id: %d' % idx)
print('Predicted class =', class_names[logreg.predict(test_vectors[idx])].
reshape(1, 1)[0,0]])
print('True class: %s' % class_names[y_test[idx]])
```

```
Document id: 1877
Predicted class = sql
True class: sql
```

We randomly select a document in the test set, which happens to be labeled SQL, and the model predicts it as SQL as well. Using this document, we generate explanations for label 4, which is SQL, and label 8, which is Python.

```
print ('Explanation for class %s' % class_names[4])
print ('\n'.join(map(str, exp.as_list(label=4))))
```

```
Explanation for class sql
('sql', 0.6326626857717909)
('date', 0.11562405901266143)
('query', 0.08730712530365084)
('values', 0.052699789384404436)
('execute', -0.04424623466659338)
('s', -0.04103939933741401)
```

```
print ('Explanation for class %s' % class_names[8])
print ('\n'.join(map(str, exp.as_list(label=8))))
```

```
Explanation for class python
('sql', -0.14435732364139484)
('query', -0.07761100453729707)
('range', 0.07204575624937577)
('1', 0.032041933794168365)
('date', -0.021905004284891793)
('d', 0.021236716325567367)
```

This document has the highest explanation for label *sql*. We also notice that the positive and negative signs are with respect to a particular label; for example, the word *sql* is positive toward the *sql* class while negative toward the *python* class, and vice versa.

We are going to generate labels for the top two classes for this document.

```
exp = explainer.explain_instance(X_test[idx], c.predict_proba, num_
features=6, top_labels=2)
print(exp.available_labels())
```

[4, 8]

It gives *sql* and *python*.

```
exp.show_in_notebook(text=False)
```

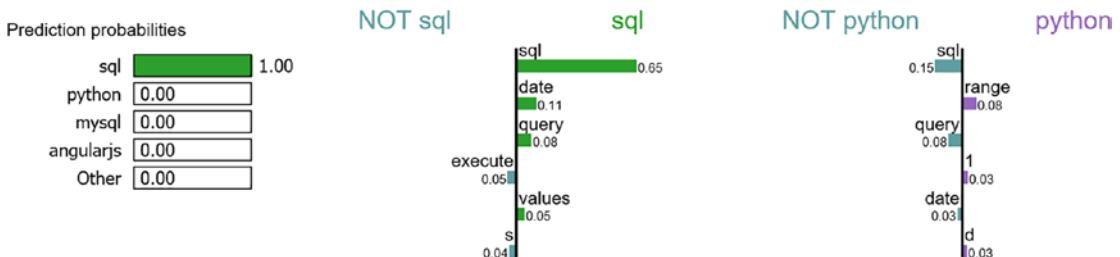


Figure 13-2. LIME output of the model

The following explains the visualization in Figure 13-2.

- For this document, the word *sql* has the highest positive score for the *sql* class.
- Our model predicts this document should be labeled as *sql* with the probability of 100%.
- If we remove the word *sql* from the document, we expect the model to predict the *sql* label with the probability at $100\% - 65\% = 35\%$.
- On the other hand, the word *sql* is negative for the *python* class, and the model has learned that word *range* has a small positive score for the *python* class.

We may want to zoom in and study the explanations for the ***sql*** class, as well as the document itself.

```
exp.show_in_notebook(text=y_test[idx], labels=(4,))
```

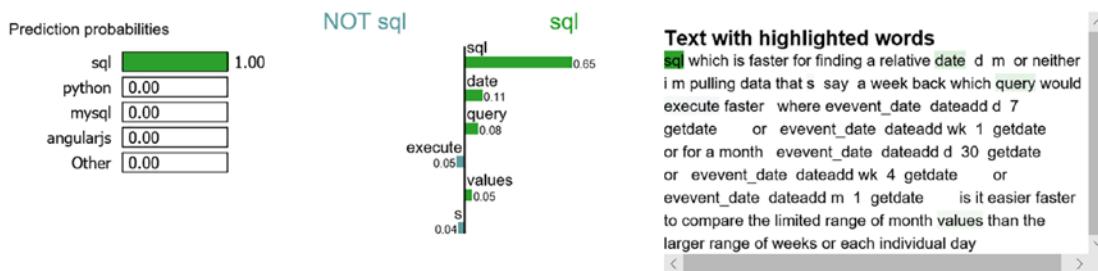


Figure 13-3. LIME output

When it comes to using LIME for NLP, the main disadvantage is that it is unstable. Different sampling around the same local data can lead to very different explanation results and that LIME only provides local interpretability. Furthermore, since LIME creates a linear local model around the instance of interest, it assumes that the local instant can be interpreted linearly, which is not always the case. The first issue can be quite dissuasive in terms of using LIME for interpretability as it is difficult to trust a model that can produce different results on the same explanation when sampling the variations changes.

However, in terms of local interpretability, most of the feature weights make sense in our own understanding of disaster-related words. The LIME explainer itself has a short runtime meaning it is easy to produce many explanations for different local predictions in a short space of time if needed.

Interpreting Text Predictions with SHAP

You studied SHAP in Chapter 9. Let's now look at how to generate explanations on text data using SHAP. There are several different types of visualizations that you can create with SHAP for understanding model classifications.

Using SHAP returns an attribution value for each feature in the model, indicating how much that feature contributed to the prediction. This is pretty straightforward for structured data, but how would it work for text? In a bag of words model (sample implementation provided in the following example), SHAP treats each word in the

vocabulary as an individual feature. We can then map the attribution values to the indices in the vocabulary to see the words that contributed most (and least) to the model's predictions. In the next section, we create a SHAP explainer object. There are a couple of types of explainers. We use DeepExplainer since we have a deep model. We instantiate it by passing it the model and a subset of the training data.

The following are the necessary imports.

```
from sklearn.preprocessing import MultiLabelBinarizer
import tensorflow as tf
from tensorflow.keras.preprocessing import text
import keras.backend.tensorflow_backend as K
K.set_session
import shap

tags_split = [tags.split(',') for tags in df['tags'].values]
tag_encoder = MultiLabelBinarizer()
tags_encoded = tag_encoder.fit_transform(tags_split)
num_tags = len(tags_encoded[0])
train_size = int(len(df) * .8)

y_train = tags_encoded[: train_size]
y_test = tags_encoded[train_size:]

class TextPreprocessor(object):
    def __init__(self, vocab_size):
        self._vocab_size = vocab_size
        self._tokenizer = None
    def create_tokenizer(self, text_list):
        tokenizer = text.Tokenizer(num_words = self._vocab_size)
        tokenizer.fit_on_texts(text_list)
        self._tokenizer = tokenizer
    def transform_text(self, text_list):
        text_matrix = self._tokenizer.texts_to_matrix(text_list)
        return text_matrix
```

```
VOCAB_SIZE = 500
train_post = df['post'].values[: train_size]
test_post = df['post'].values[train_size: ]
processor = TextPreprocessor(VOCAB_SIZE)
processor.create_tokenizer(train_post)
X_train = processor.transform_text(train_post)
X_test = processor.transform_text(test_post)

def create_model(vocab_size, num_tags):
    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.Dense(50, input_shape = (VOCAB_SIZE,), activation='relu'))
    model.add(tf.keras.layers.Dense(25, activation='relu'))
    model.add(tf.keras.layers.Dense(num_tags, activation='sigmoid'))
    model.compile(loss = 'binary_crossentropy', optimizer='adam', metrics = ['accuracy'])
    return model

model = create_model(VOCAB_SIZE, num_tags)
model.fit(X_train, y_train, epochs = 2, batch_size=128, validation_split=0.1)
print('Eval loss/accuracy:{}'.format(model.evaluate(X_test, y_test, batch_size = 128)))
```

After the model is trained, we use the first 200 training documents as our background data set to integrate and create a SHAP explainer object.

- We get the attribution values for individual predictions on a subset of the test set.
- Transform the index to words.
- Use SHAP's summary_plot method to show the top features impacting model predictions.

```
attrib_data = X_train[:200]
explainer = shap.DeepExplainer(model, attrib_data)
num_explanations = 20
```

```

shap_vals = explainer.shap_values(X_test[:num_explanations])
words = processor._tokenizer.word_index
word_lookup = list()
for i in words.keys():
    word_lookup.append(i)
word_lookup = [''] + word_lookup
shap.summary_plot(shap_vals, feature_names=word_lookup, class_names=tag_encoder.classes_)

```

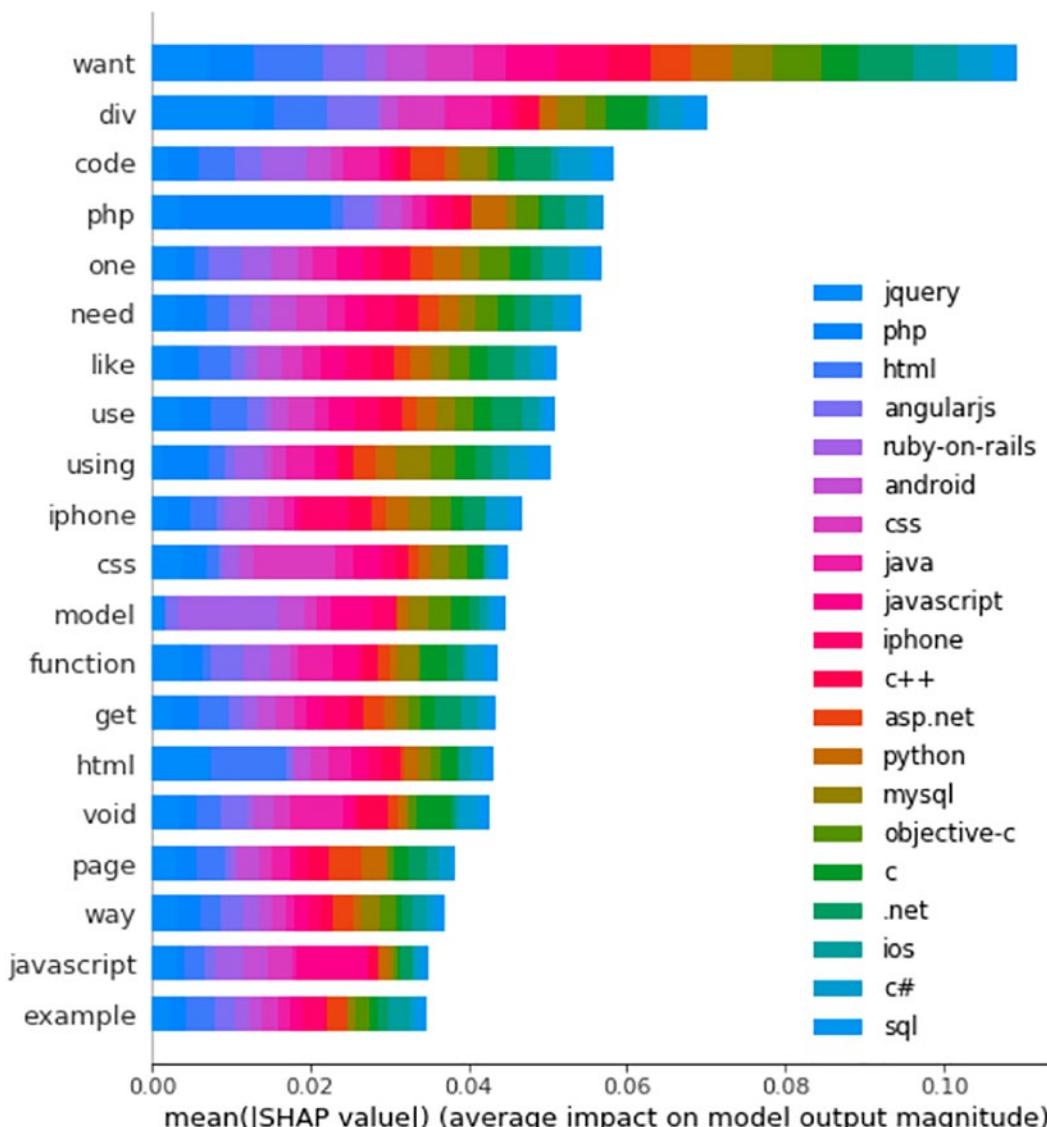


Figure 13-4. SHAP output of the model for text classification

- The word *want* is the biggest signal word used in the model, contributing most to ***jquery*** class predictions.
- The word *php* is the fourth biggest signal word used in the model, contributing most to the ***php*** class.
- On the other hand, the word *php* is likely to have a negative signal to the other class because it is unlikely to see *php* appear in a ***python*** document.

This is how we use SHAP for explaining text models. In the example, we highlighted the most important words for the model.

Explaining Text Models with Sentence Highlighting

Sentence highlighting consists of assigning a score to every word based on the importance of that word in the final prediction. Formally, a sentence highlighting (SH) is modeled as a vector s that explains a classification $y = b(x)$ of a black-box b on x . The dimensions of s are the words present in the sentence x we want to explain, and the s_i value is the saliency value of the word i . The greater the value of s_i , the greater the importance of that word. A positive value indicates a positive contribution toward y , while a negative one means that the word has contributed negatively.

ELI5 is a Python library that allows to visualize and debug various machine learning models using unified API. It has built-in support for several ML frameworks and provides a way to explain black-box models.

We are working with the Stack Overflow data set to explain how ELI5 works for the text data. This data set contains posts and the corresponding tag for the post.

Figure 13-5 shows the data set.

	post	tags
0	what is causing this behavior in our c# datet...	c#
1	have dynamic html load as if it was in an ifra...	asp.net
2	how to convert a float value in to min:sec i ...	objective-c
3	.net framework 4 redistributable just wonderi...	.net
4	trying to calculate and print the mean and its...	python

Figure 13-5. Sample data for text modeling

This is a balanced data set and thus suited well for our purpose of understanding.

First, use a simple scikit-learn pipeline to build a text classifier, which we interpret later. This pipeline uses a simple count vectorizer and logistic regression.

```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegressionCV
from sklearn.pipeline import make_pipeline# Creating train-test Split
X = soda[['post']]
y = soda[['tags']]X_train, X_test, y_train, y_test = train_test_split(X,
y, test_size=0.2, random_state=0)# fitting the classifier
vec = CountVectorizer()
clf = LogisticRegressionCV()
pipe = make_pipeline(vec, clf)
pipe.fit(X_train.post, y_train.tags)
```

The following are the results (also see Figure 13-6).

```
from sklearn import metricsdef print_report(pipe):
    y_actuals = y_test['tags']
    y_preds = pipe.predict(X_test['post'])
    report = metrics.classification_report(y_actuals, y_preds)
```

```
print(report)
print("accuracy: {:.3f}".format(metrics.accuracy_score(y_actuas,
y_preds)))print_report(pipe)
```

	precision	recall	f1-score	support
.net	0.64	0.67	0.66	377
android	0.93	0.88	0.91	437
angularjs	1.00	0.97	0.98	433
asp.net	0.77	0.75	0.76	387
c	0.80	0.86	0.83	421
c#	0.56	0.61	0.58	387
c++	0.79	0.73	0.76	411
css	0.81	0.86	0.83	401
html	0.65	0.70	0.67	378
ios	0.67	0.62	0.64	397
iphone	0.67	0.71	0.69	412
java	0.82	0.79	0.80	377
javascript	0.80	0.79	0.79	391
jquery	0.85	0.85	0.85	402
mysql	0.86	0.78	0.82	414
objective-c	0.67	0.64	0.65	400
php	0.83	0.82	0.83	398
python	0.93	0.93	0.93	406
ruby-on-rails	0.97	0.93	0.95	363
sql	0.78	0.86	0.81	408
accuracy			0.79	8000
macro avg	0.79	0.79	0.79	8000
weighted avg	0.79	0.79	0.79	8000
accuracy: 0.789				

Figure 13-6. Output of the model

Figure 13-6 is a simple logistic regression model and it performs well. You can check out its weights using the following function.

```
for i, tag in enumerate(clf.classes_):
    coefficients = clf.coef_[i]
    weights = list(zip(vec.get_feature_names(),coefficients))
    print('Tag:',tag)
    print('Most Positive Coefficients:')
    print(sorted(weights,key=lambda x: -x[1])[:10])
    print('Most Negative Coefficients:')
    print(sorted(weights,key=lambda x: x[1])[:10])
    print("-----")-----
```

OUTPUT:

```
-----Tag: python
Most Positive Coefficients:
[('python', 6.314761719932758), ('def', 2.288467823831321), ('import',
1.4032539284357077), ('dict', 1.1915110448370732), ('ordered',
1.1558015932799253), ('print', 1.1219958415166653), ('tuples',
1.053837204818975), ('elif', 0.9642251085198578), ('typeerror',
0.9595246314353266), ('tuple', 0.881802590839166)]
Most Negative Coefficients:
[('java', -1.8496383139251245), ('php', -1.4335540858871623),
('javascript', -1.3374796382615586), ('net', -1.2542682749949605),
('printf', -1.2014123042575882), ('objective', -1.1635960146614717),
('void', -1.1433460304246827), ('var', -1.059642972412936), ('end',
-1.0498078813349798), ('public', -1.0134828865993966)]
-----
Tag: ruby-on-rails
Most Positive Coefficients:
[('rails', 6.364037640161158), ('ror', 1.804826792986176),
('activerecord', 1.6892552000017307), ('ruby', 1.41428459023012),
('erb', 1.3927336940889532), ('end', 1.3650227017877463), ('rb',
1.2280121863441906), ('gem', 1.1988196865523322), ('render',
1.1035255831838242), ('model', 1.0813278895692746)]
Most Negative Coefficients:
```

CHAPTER 13 EXPLAINING TEXT CLASSIFICATION MODELS

```
[('net', -1.5818801311532575), ('php', -1.3483618692617583), ('python',
-1.201167422237274), ('mysql', -1.187479885113293), ('objective',
-1.1727511956332588), ('sql', -1.1418573958542007), ('messageform',
-1.0551060751109618), ('asp', -1.0342831159678236), ('ios',
-1.0319120624686084), ('iphone', -0.9400116321217807)]
```

And that is all pretty good. The coefficients make sense, and we can improve the model using this information. But this was a lot of code. ELI5 makes this exercise simple. You use the following command (see the output in Figure 13-7).

```
import eli5
eli5.show_weights(clf, vec=vec, top=20)
```

y=jquery top features		y=mysql top features		y=objective-c top features		y=php top features		y=python top features		y=ruby-on-rails top features		y=sql top features	
Weight [*]	Feature	Weight [*]	Feature	Weight [*]	Feature	Weight [*]	Feature	Weight [*]	Feature	Weight [*]	Feature	Weight [*]	Feature
-3.977	jquery	-4.897	api	+4.001	objective	+2.558	php	+6.315	python	+6.354	rails	+2.649	sql
-2.060	ready	+1.316	million	+1.276	password	+1.605	git	+2.288	def	+1.205	ror	+1.284	database
+1.166	jqfiddle	+1.158	isregistered	+1.268	monich	... 23131 more positive 128892 more negative ...	+1.403	import	+1.689	activerecord	+1.175	databases
+1.109	fadain	+0.962	restore	+1.262	cocos	... 35365 more positive 12717 more negative ...	+1.192	dict	+1.414	ruby	... 39770 more positive 112343 more negative ...
+1.090	animate	+0.927	mdn	+1.241	zzplib	+1.271	iphone	+1.156	ordered	+1.393	erb	... 112343 more negative 1086 ok
+24776 more positive ...		+0.920	truncated	... 35365 more positive ...		+1.278	java	+1.122	print	+1.365	end	-1.099 javascript	bookings
+0.900	tablename	... 32999 more positive 176757 more negative ...		+1.282	objective	+1.054	tuples	+1.228	rb	-1.111 jquery	laval
+0.900	blockquote	+1.155	who	+1.283	page_title_home	... 34264 more positive 34264 more positive ...	+1.199	gen	+1.104	render	-1.125	miasselin
-1.029	employee	... 119114 more negative ...		+1.182	comment	+1.339	void	... 117849 more negative 117849 more negative ...	+1.081	model	-1.154	datas
-1.056	scope	-0.921	connections	+1.204	def	+1.340	flags	-0.976	programming	+1.045	has_many	-1.185	asp
-1.081	python	-0.933	python	+1.206	stringbyAppendingFormat	+1.366	logout	-1.012	sql	... 17639 more positive 17639 more positive ...	-1.191	million
-1.087	folder	-0.954	app	+1.223	jquery	+1.379	fields_by_id	-1.013	public	... 134474 more negative 134474 more negative ...	-1.275	iphone
-1.087	int	-0.992	web	+1.225	mysql	+1.398	javascript	-1.050	end	-1.142	objective	-1.278	chat
-1.151	mean	-1.001	product_table	+1.243	selectedString	+1.450	clicking	-1.143	var	-1.055	messageform	-1.312	python
-1.164	sql	+1.03	2010	+1.282	textfieldDidDedding	+1.568	rails	-1.149	void	-1.142	script	-1.396	rails
-1.164	javascript	+1.065	define	+1.402	ot	+1.600	uri	-1.154	objective	-1.142	script	-1.457	net
-1.333	getelementbyid	+1.113	conn	+1.437	css	+1.617	angularjs	-1.201	printf	-1.173	objective	-1.696	net
+1.502	raids	+1.124	net	+1.471	net	+1.705	footer	-1.254	net	-1.187	mysql	-1.716	chat
-1.650	angularjs	+1.161	name_table	+1.497	javascript	+1.711	uri	-1.337	javascript	-1.201	python	-2.360	<BIAS>
-1.930	ng	+1.642	android	+1.631	android	+2.092	python	-1.434	php	-1.348	php	-4.457	mysql
-2.222	angular	+1.889	rails	+1.749	swift	+2.463	jquery	-1.850	java	-1.582	net	-1.716	<BIAS>
-3.738	<BIAS>	-3.892	<BIAS>	-2.530	<BIAS>	-3.317	<BIAS>	-3.737	<BIAS>	-3.730	<BIAS>	-4.457	<BIAS>

Figure 13-7. Output of ELI5 for explaining the model

The weights value for Python is the same as the values we got from the function we wrote manually. And it is much prettier and wholesome to explore. But that is just the tip of the iceberg. ELI5 can also debug models. Now let's find out why a particular example is misclassified. We are using an example originally from the class Python but got misclassified as Java (see Figure 13-8).

```
y_preds = pipe.predict(sodata['post'])sodata['predicted_label'] = y_
predsmisclassified_examples = sodata[(sodata['tags']!=sodata['predicted_label'])&(sodata['tags']=='python')&(sodata['predicted_label']=='java')]eli5.
show_prediction(clf, misclassified_examples['post'].values[1], vec=vec)
```

```
y=java (probability 0.298, score -3.665) top features
Contribution* Feature
-0.525 Highlighted in text (sum)
-3.141 <BIAS>

add element from a list of dictionary to another <pre><code>dict_1 = [ { incident_id : sd000001372596 first_call : t } { incident_id : sd000001372594 first_call : f } { incident_id : sd000001372598
first_call : f } { incident_id : sd000001372599 first_call : t } { incident_id : sd000001372602 first_call : f } { incident_id : sd000001372601 first_call : f } { incident_id : sd000001372605 first_call : f } { incident_id : sd000001372606 first_call : f } { incident_id : sd000001372607 first_call : f } ] dict_2 = [ { incident_id : sd000001372605 date : 08-10-2016 00:54:13 } { incident_id : sd000001372606 date :
08-10-2016 00:57:20 } { incident_id : sd000001372607 date : 08-10-2016 01:00:25 } { incident_id : sd000001372598 date : 11-10-2016 10:57:34 } { incident_id : sd000001372602 date : 08-10-2016
10:44:34 } { incident_id : sd000001372601 date : 21-10-2016 22:30:49 } { incident_id : sd000001372594 date : 18-10-2016 14:53:34 } ] </code></pre> I have two list of dictionaries with different length and i want to add the ( dict_2 date ) to the dict_1 according the incident_id

y=python (probability 0.005, score -7.792) top features
Contribution* Feature
-3.737 <BIAS>
-4.054 Highlighted in text (sum)

add element from a list of dictionary to another <pre><code>dict_1 = [ { incident_id : sd000001372596 first_call : t } { incident_id : sd000001372594 first_call : f } { incident_id : sd000001372598
first_call : f } { incident_id : sd000001372599 first_call : t } { incident_id : sd000001372602 first_call : f } { incident_id : sd000001372601 first_call : f } { incident_id : sd000001372605 first_call : f } { incident_id : sd000001372606 first_call : f } { incident_id : sd000001372607 first_call : f } ] dict_2 = [ { incident_id : sd000001372605 date : 08-10-2016 00:54:13 } { incident_id : sd000001372606 date :
08-10-2016 00:57:20 } { incident_id : sd000001372607 date : 08-10-2016 01:00:25 } { incident_id : sd000001372598 date : 11-10-2016 10:57:34 } { incident_id : sd000001372602 date : 08-10-2016
10:44:34 } { incident_id : sd000001372601 date : 21-10-2016 22:30:49 } { incident_id : sd000001372594 date : 18-10-2016 14:53:34 } ] </code></pre> I have two list of dictionaries with different length and i want to add the ( dict_2 date ) to the dict_1 according the incident_id
```

Figure 13-8. Sample output

In Figure 13-8, the classifier predicts Java with a low probability. And you can examine a lot of things going on in the example to improve the model. The following are some examples.

- The classifier is taking a lot of digits into consideration (not good), which brings us to the conclusion of cleaning up the digits—or replacing DateTime objects with a DateTime token.
- While the dictionary has a negative weight for Java, the word dictionaries have positive weight. So maybe stemming could also help.
- Words like <pre><code> influence the classifier. These words should be removed while cleaning.
- Why is the word *date* influencing the results? Something to think about.

We can take a look at more examples to get more such ideas. Let's create a deep model for this simple task. This is all good, but what if our models don't provide weights for the individual features like LSTM? It is with these models that explainability can play a very important role.

To understand how to do this, we first create a TextCNN model on the data. Things get interesting from our point of view when we have a trained black-box model object. ELI5 provides eli5.lime.TextExplainer debug our prediction to check what was important in the document to make a prediction decision.

To use the TextExplainer instance, we pass a document to explain and a black-box classifier (a predict function that returns probabilities) to the fit() method. From the documentation, the predict function should look like the following.

predict (callable) — Black-box classification pipeline. predict should be a function that takes a list of strings (documents) and return a matrix of shape (n_samples, n_classes) with probability values - a row per document and a column per output label.

To use ELI5 we need to define our own function which takes as input a list of strings (documents) and return a matrix of shape (n_samples, n_classes). *You can see how we first preprocess and then predict.*

```
def predict_complex(docs):
    # preprocess the docs as required by our model
    val_X = tokenizer.texts_to_sequences(docs)
    val_X = pad_sequences(val_X, maxlen=maxlen)
    y_preds = model.predict([val_X], batch_size=1024, verbose=0)
    return y_preds
```

The following shows how you can use TextExplainer (see Figure 13-9). Using the same misclassified example as before in our simple classifier.

```
import eli5
from eli5.lime import TextExplainer
te = TextExplainer(random_state=2019)
te.fit(sodata['post'].values[0], predict_complex)
te.show_prediction(target_names=list(encoder.classes_))
```

y=python (probability 0.640, score 1.000) top features

Contribution	Feature
+1.270	Highlighted in text (sum)
-0.270	<BIAS>

add element from a list of dictionary to another <pre><code>dict_1 = [{ incident_id : sd000001372596 first_call : t } { incident_id : sd000001372594 first_call : f } { incident_id : sd000001372594 first_call : f } { incident_id : sd000001372599 first_call : t } { incident_id : sd000001372602 first_call : f } { incident_id : sd000001372601 first_call : f } { incident_id : sd000001372605 first_call : f } { incident_id : sd000001372608 first_call : f } { incident_id : sd000001372607 first_call : f }] dict_2 = [{ incident_id : sd000001372605 date : 08-10-2016 00:54:13 } { incident_id : sd000001372606 date : 08-10-2016 00:57:20 } { incident_id : sd000001372607 date : 08-10-2016 01:00:25 } { incident_id : sd000001372598 date : 11-10-2016 10:57:34 } { incident_id : sd000001372602 date : 08-10-2016 10:44:34 } { incident_id : sd000001372601 date : 21-10-2016 22:30:49 } { incident_id : sd000001372594 date : 18-10-2016 14:53:34 }] </code></pre> I have two list of dictionaries with different length and i want to add the (dict_2 date) to the dict_1 according the incident_id

Figure 13-9. Output using TextExplainer

You can see that the presence of keywords dict and list is influencing the decision of our classifier. In this example, TextExplainer generates a lot of texts similar to the document by removing some of the words and then trains a white-box classifier which predicts the output of the black-box classifier and not the true labels. The explanation we see is for this white-box classifier. Refer to the notebook for the code at www.kaggle.com/mlwhiz/interpreting-text-classification-models-with-eli5.

Summary

We did not discuss any theory in this chapter because most text classification models can be explained using techniques like LIME, SHAP, and ELI5. These methods were covered in previous chapters. With this chapter, we hope that you got some idea of how interpretability works in terms of textual data.

CHAPTER 14

The Role of Data in Interpretability

Now, let's try to understand what lies beyond the methods and modeling techniques. This chapter attempts to find answers to the following questions.

- How does data complexity affect model explainability?
- If different tools generate the explanations, are they the same or different?

In the last few chapters, we studied the methods built on a framework of feature importance. We evaluated methods like CIU, DALEX, ELI5, SHAP, and Skater. Each of these methods works on some common theme to generate global importance measures for ranking features of the model.

Let's elaborate on the preceding two questions. The tools mentioned earlier explain black-box models. Can it be deduced that they similarly generate the global feature importance of the same data sets, or is there a mismatch between the explanations generated for the same data set? On similar lines, are the generation of similar explanations related to specific properties of the data set?

This chapter answers the earlier two questions and whether data set complexity matters for interpretability. This is done by choosing tabular data sets, clustering the data sets based on various predefined properties, building models and explainability methods on these data sets, and then calculating the correlation between pairs of different methods to analyze the results.

The data sets¹ were australian, phishing websites, spec, satellite, anal-catdata lawsuit, banknote authentication, blood transfusion service center, churn, climate model

¹The data sets are open sourced and used mostly on the OpenML platform. Please search by data set name on OpenML platform (www.openml.org) for more information.

simulation crashes, credit-g, delta ailerons, diabetes, eeg-eye-state, haberman, heart-statlog, ilpd, ionosphere, jEdit-4.0-4.2, kc1, kc2, kc3, kr-vs-kp, mc1, monks-problems-1, monks-problems-2, monks-problems-3, mozilla4, mw1, ozone-level-8hr, pc1, pc2, pc3, pc4, phoneme, prnn crabs, qsar-biodeg, sonar, spambase, steel-plates-fault, and tic-tac-toe e wdbc.

The following preprocessing steps were done to prepare the data sets for modeling purposes.

1. Convert categorical features to ordinal values based on frequency.
2. Convert boolean features to numerical (0 and 1).
3. Use min-max normalization to values between 0 and 1.

Analyzing different data sets enables you to group them based on their similarities and differences, even when the data sets are of different themes. For example, one group can be created based on class entropy values and then segregate the data sets that have more or less information, thus enabling the data set complexity to impact relationships for future model explanations. This chapter analyzes 15 different properties of the data sets and then uses the k-means clustering algorithm to identify the groups of data sets.

The 15 properties were number of features, number of instances, dimensionality, percentage of binary features, standard deviation nominal of attribute distinct values, mean nominal attribute distinct values, number of binary features, percentage of symbolic features, percentage of numeric features, class entropy, autocorrelation, number of numeric features, number of symbolic features, majority class percentage, and the minority class percentage.

Let's analyze the results based on random forest and gradient boosting as both are tree-based ensembles and offer low model explainability.

A standard train-test split was performed for each model: 70% of the data was used for training and the remaining 30% was tested. Then on each model, the metrics such as accuracy, precision, and recall were generated to define the metrics of the model created.

For each of the 41 data sets, two models were created: one based on the random forest algorithm and the other on gradient boosting, generating a total of 82 models. Global explainability feature importance ranks were produced through the six different feature importance methods for each model, resulting in a total of 492 different types of ranks. The 41 data sets were divided into three different groups based on k-means clustering.

For analyzing the rank pairs of global feature importance, Spearman rank correlation was used. This measures the correlation between rank pairs, considering the idea of ranks in which different values may appear (in this case, they are data set attributes).

In this step, two comparison matrixes of rank correlation pairs are generated for each data set. The results were computed in groups (one for each random forest and gradient boosting) and clusters (three different data set clusters).

Each cluster has 21, 17, and 3 data sets, respectively. To understand the impact of a data set, we need to study the behavior of these clusters. This chapter does not discuss the last cluster because it has very few data sets (only three).

Out of the total 15 properties, we discuss the four properties that best describe the behavior of the analyzed clusters.

The four properties of the data sets shown were selected from 15 properties, for being the ones that best differentiate the two analyzed clusters. However, we don't mean to say that the other 11 properties do not explain the clusters. The only difference is that the remaining 11 do it to a lesser extent.

- **Cluster 0:** This cluster characterizes data sets with a higher number of features, lower-class entropies (less information), higher autocorrelations, and higher imbalances.
- **Cluster 1:** This cluster has data sets with fewer features, higher class entropies (more information), lower autocorrelations, and lower imbalances.

The results are displayed in Figure 14-1 and Figure 14-2 (also see <https://arxiv.org/abs/2107.02661>).

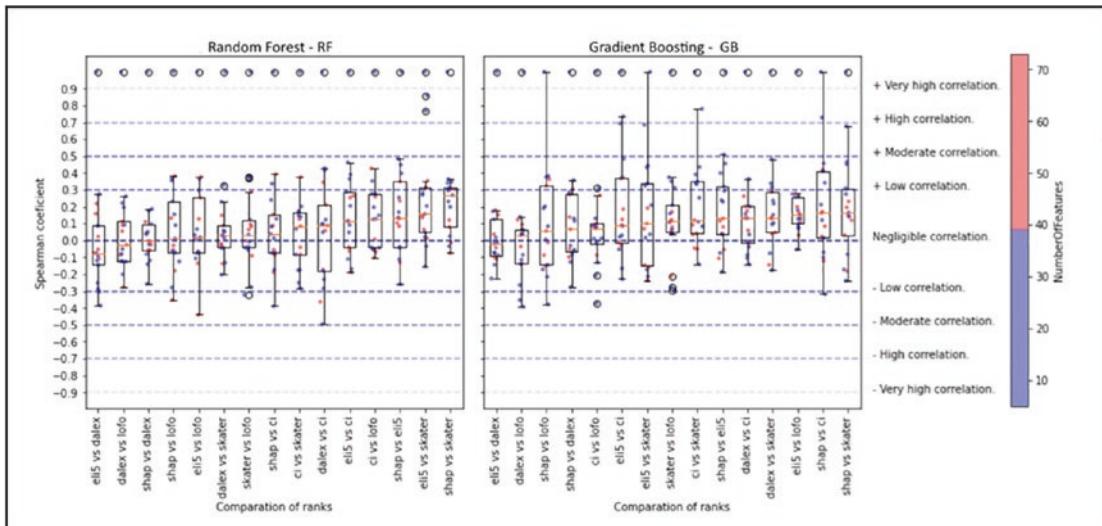
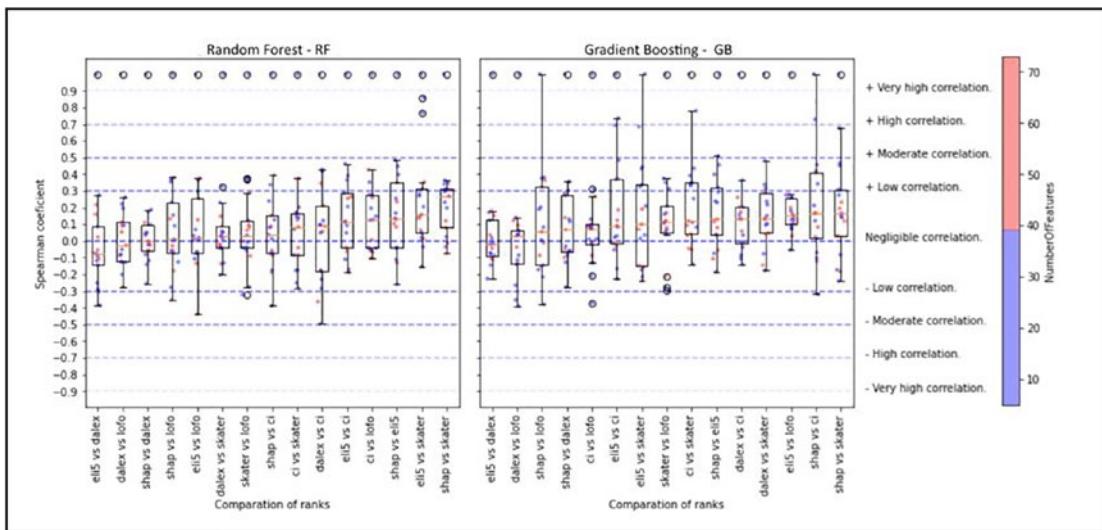
**Figure 14-1.** Boxplot graphs for cluster 0**Figure 14-2.** Boxplot graphs for cluster 1

Figure 14-1 is a boxplot graph that summarizes all rank pair comparisons (x axis) and their respective Spearman correlations (y axis) calculated for the data sets of cluster 0. Random forest is on the left, and gradient boosting is on the right. The dashed blue lines show the different levels of correlations. The points identify the positions of the correlation values of each comparison. The color of the points refer to the attribute

quantities of each data set. Note the low variance of most boxplots, along with levels of negligent or low correlations found.

The boxplot graphs in Figure 14-2 summarize all rank pair comparisons (x axis) and their respective Spearman correlations (y axis) calculated for the cluster 1 data sets. Random forest is on the left, and gradient boosting is on the right. The dashed blue lines show the different levels of correlations. The points identify the positions of the correlation values of each comparison. The color of the points refers to the attribute quantities of each data set. Note the high variances of most boxplots, along with the high levels of correlations found.

Figures 14-1 and 14-2 show that the execution for data sets belonging to cluster 0 have low or negligent correlations as per several tests. This shows that the six interpretability methods to determine the comparison generate different explainability ranks for most of the data sets in the groups, irrespective of the random forest or the gradient boosting algorithm used. The results show that data sets with greater complexity generate different explanations. This justifies the low correlations found in the results.

The results answer the first hypothesis. Considering the current tools aimed at explaining black-box machine learning models, it can be inferred that they generate global rankings of same, similar, or different explainabilities?

Different correlations show that the properties of the data sets directly influence the explainability of the models. Thus, the tools can generate explainabilities with higher correlations or lower correlations.

Despite having similar results between random forest and gradient boosting models, there are differences in correlations. There is a larger variance in some comparisons of the gradient boosting algorithm results; for example, between SHAP and Lofo, ELI5 vs. CIU, and ELI5 vs. Skater. This proves that the complexity of the algorithm used in the model (in this case, bagging and boosting) influences the generation of explainability.

Importantly, the outliers existing in the figures of boxplots show the possibility of existing other data set clusters among those analyzed. The colors used to distinguish the number of attributes from different data sets in each boxplot show that the results obtained in the correlations are not a function of the number of attributes that a data set has.

The comparison of the results shows the answer to the second hypothesis. Following the same idea as in the previous question, are the generations of equal, similar, or different explainabilities related to specific properties of a data set?

Yes, since the results of the rank correlations for the two data set clusters presented in the benchmark output were different, which shows that properties such as class entropy, autocorrelation, majority class percentage, and the number of attributes are properties that interfere with explainability ranks.

- If a model (algorithm and data set) solves a classification or regression problem for a data set with low complexity, there must be few ranks (or even only one rank) of explainabilities referring to this model, allowing you to infer that the different explainability metrics present higher correlations with each other.
- If a model (algorithm and data set) solves a classification or regression problem for a data set with high complexity, there must be many explainability ranks for this model, leading the ranks of the different explainability metrics to show lower correlations with each other.

Summary

This chapter deduced that data set complexity does matter for generating explanations on classification models, and the choice of model also greatly affects the overall feature importance ranks. In this chapter, we wanted to warn our readers not to consider explainability tools as magic wands that can be waved to generate accurate and easily understandable models. Users of interpretability techniques should adopt the explainability practice as a ritual in their model building exercises and then follow the best practices of model development. Model development and making it human understandable requires a lot of effort than applying an explainability trick on top of the model functions.

CHAPTER 15

The Eight Pitfalls of Explainability Methods

Model-agnostic model explanation techniques such as permutation importance plots, permutation feature importance, and Shapley values provide many insights into model explanations but often lead to wrong conclusions if they are not used correctly.

This chapter highlights many general pitfalls of the model interpretations, such as using interpretation techniques in the wrong context or using models that do not generalize well. Sometimes people ignore feature dependencies, interactions, and uncertainty estimates in a high-dimensional data set. This chapter focuses on the common pitfalls that can arrive when using explainability techniques.

Since many explainability techniques work on the same principle of manipulating the data and discovering the model, they also share a lot of drawbacks. Most of the models usually have non-linear effects and higher-order interactions. The interpretation method can be applied regardless of the model's performance on test data, but we can only draw solid conclusions about the data when it generalizes well. In some cases, a simpler model is sufficient to carry out the desired tasks, and hence a complex model and model explainability technique may add unnecessary complexity. Drawbacks like these are easily ignored, but they exist quite frequently in many machine learning exercises.

Assuming One-Fits-All Interpretability

There are a lot of explainability methods that practitioners can choose to answer the questions around why the model took a certain decision. Too many options make it difficult for practitioners to choose a method for their requirements. If a single explainability method can fit across multiple use cases is a very dangerous assumption. For example, there are many global feature importance methods like permutation

feature importance and SHAP. The output provided by both can be entirely different, and their usage objective can also be different. For example, if the model builder wants insight into the relevance of the feature (e.g., a data scientist selecting model features) regarding the model's generalization error, a loss-based method such as permutation feature importance should be used. When the model builder wants to expose the features the model relies on for prediction, then Shap feature importance (e.g., a business analyst analyzing the model to present to stakeholders) should be used. Hence, model builders must investigate their choices of explainability methods to align with their goals and result in meaningful analysis.

Bad Model Generalization

When a model is underfit or overfit, it may result in misleading feature importance scores because the output does not match the underlying model well enough.

Most explainability methods are designed to interpret the model instead of creating inferences about the data-generating process. When a model approximates the data-generating process well enough, its interpretation reveals insights into the underlying process. Training data should never assess the performance of the models because then the model can overfit the training data, which leads to optimistic or inflated estimates. Out-of-sample validation based on sampling procedures should be used since many of these methods are readily available and well-studied and practiced. In computational model selection and hyperparameter tuning, nested sampling is preferred.

Unnecessary Use of Complex Models

One common pitfall while using explainability techniques is to use an opaque complex model when a simple self-interpretable model is sufficient. Although model-agnostic explainability techniques shed a lot of light on a complex black-box model's behavior and mechanism, sometimes interpretable models offer a high degree of transparency.

Although model-agnostic methods can shed light on the behavior of complex ML models, inherently interpretable models still offer a higher degree of transparency, and considering them increases the chance of discovering the true data-generating function. It is commonly believed that complex ML models always outperform more interpretable

models in terms of accuracy and should thus be preferred. However, there are several examples where interpretable models have proven to be serious competitors.

We recommend starting with simple, interpretable models such as linear regression models and decision trees. GAM or generalized additive models can gradually transition between simple linear models and more complex machine learning models. GAMs have the desirable property to additively model smooth, non-linear effects and provide PDPs out-of-the-box, but without the potential pitfall of masking interactions. The additive model structure of a GAM is specified before fitting the model so that only the pre-specified feature or interaction effects are estimated. Interactions between features can be added manually or algorithmically (e.g., via a forward greedy search).

A complex model should only be preferred if the gain in accuracy is significant and relevant. However, it is highly advisable to start with simple models and slowly climb up the ladder to try more complex models.

Ignoring Feature Dependence

Before applying interpretation methods, practitioners should check for dependencies between features in the data. Let's look at various issues which can be caused by dependency on the features.

Interpretation with Extrapolation

Whenever ML models rely on feature interactions (i.e., features are dependent), methods such as permutation feature importance, partial dependence plots, and shapely values that extrapolate in the training data often result in misleading interpretations. These extrapolations, or perturbations, produce artificial data points used for model predictions and are further aggregated to produce global explanations. Feature values are perturbed generally by randomly subsampled values.

Hence, whenever you apply model interpretation methods, you should always check for dependencies in the features via descriptive statistics or measures of dependence. If you include dependent features in the model, you should also provide additional information regarding the strength of dependence and the type of relationship.

Confusing Linear Correlation with General Dependence

Sometimes when the correlation coefficient between features is zero, there still can exits dependence between features, and when put in the model, it can cause misleading model interpretations. The opposite of “independence between features implies that Pearson correlation coefficient will be zero” is generally false. The coefficient generally tracks linear relationships and is often influenced by outliers.

When we have a lesser number of features, we can use scatter plots. However, for high-dimensional plots, we can use other measures of dependence. If dependence is monotonic, then Spearman’s rank correlation can be used. We should use separate measures for categorical or mixed features.

Misunderstanding Conditional Interpretation

Conditional variants of interpretation techniques avoid extrapolation but require a different interpretation. Interpretation methods that perturb features independently of others extrapolate under dependent features but provide insight into the model’s mechanism. Conditional variants of interpretation techniques avoid extrapolation but require a different interpretation. Therefore, these methods are said to be true to the model but not true to the data. For feature effect methods such as the partial dependence plots, the plot can be interpreted as the isolated, average effect the feature has on the prediction. For the permutation feature importance, the importance can be interpreted as the drop in performance when the feature’s information is destroyed” (by perturbing it). When features are highly dependent and conditional effects, and importance scores are used, the practitioner must be aware of the distinct interpretation.

Misleading Interpretations Due to Feature Interactions

Sometimes feature interaction cause a lot of confusion when explanations are generated by using various methods. Let’s investigate the various issues that feature interactions can cause.

Misleading Feature Effects Due to Aggregation

Global interpretation methods, such as partial dependence plots or accumulated local effects plots, visualize the average effect on a model's prediction. However, they can produce misleading interpretations when features interact.

For the partial dependence plot, we recommend additionally considering the corresponding individual conditional expectations curves. ICE curves directly show the heterogeneity between individual predictions, while partial dependence plots and accumulated local effects average interaction effects. You do not lose much information by aggregating these individual conditional expectations curves to a global marginal effect curve such as the partial dependence plots. However, when the features interact, the marginal effect curves of different observations might not show similar effects on the target.

Failing to Separate Main from Interaction Effects

It is very important to separate the interaction effects from the main effects for a model interpretation method. Permutation feature importance, for example, includes both the importance of the features and the interaction with other features. Methods like SHAP and LIME provide additive explanations, but there is no separation of main and interaction effects. Functional ANOVA is probably the most popular approach to decompose the joint distribution into main and interaction effects. Using the same idea, the H-statistic quantifies the interaction strength between two features or between one feature and all others by decomposing the two-dimensional PDP into univariate components. In non-interacting features, the two-dimensional partial dependence function equals the sum of the two underlying univariate partial dependence functions.

Ignoring Model and Approximation Uncertainty

Many interpretation methods only provide a mean estimate but do not quantify uncertainty. Both the model training and the computation of interpretation are subject to uncertainty. The model is trained on (random) data, and therefore should be regarded as a random variable. Interpretation methods are often defined in terms of expectations over the data (PFI, PDP, Shapley values, ...) but are approximated using Monte Carlo integration. Ignoring these two sources of uncertainty can result in the interpretation of

noise and non-robust results. This effect could cancel out once averaged over multiple model fits. A single PDP can be misleading because it does not show the variance due to PDP estimation (second plot) and model fitting. If we are not interested in learning about a specific model but rather about the relationship between feature X1 and the target, we should consider the model variance.

By repeatedly computing PDP and PFI with a given model but with different permutations or bootstrap samples, the estimate's uncertainty can be quantified, for example, in the form of confidence intervals. For PFI, frameworks for confidence intervals and hypothesis tests exist, but they assume a fixed model. If the practitioner wants to condition the analysis on the modeling process and capture the variance of the process instead of conditioning on a fixed model, PDP and PFI should be computed on multiple model fits.

Failure to Scale to High-Dimensional Settings

Sometimes high-dimensional settings are ignored while generating or using explanations. Let's investigate the issues that this can cause.

Human-Intelligibility of High-Dimensional IML Output

Applying explainability methods naively to high-dimensional data sets (e.g., visualizing feature effects or computing importance scores on feature level) leads to an overwhelming and high-dimensional explainability output, which impedes human analysis. Especially interpretation methods based on visualizations make it difficult for practitioners in high-dimensional settings to focus on the most important insights.

A natural approach is to reduce the dimensionality before applying any IML methods. Whether this facilitates understanding depends on the possible semantic interpretability of the resulting reduced feature space as features can either be selected or dimensionality can be reduced by linear or non-linear transformations. If users would like to interpret in the original feature space, many feature selection techniques can be used, resulting in much sparser and easier to interpret models. Wrapper selection approaches are model-agnostic and algorithms like greedy forward selection or subset selection procedures, which start from an empty model and iteratively add relevant (subsets of) features if needed, even allowing to measure the relevance of features for predictive performance. An alternative is to directly use models that implicitly perform

feature selection, such as LASSO or component-wise boosting, as they can produce sparse models with fewer features.

Computational Effort

Some interpretation methods do not scale linearly with the number of features. For example, for the computation of exact Shapley values, the number of possible coalitions, or for a (full) functional ANOVA decomposition, the number of components (main effects plus all interactions) scales with $O(2^p)$.

For the functional ANOVA, a common solution is to keep the analysis to the main effects and selected two-way interactions (similar for PDP and ALE). Interesting two-way interactions can be selected by another method, such as the H-statistic. However, the selection of two-way interactions requires additional computational effort. Interaction strength usually decreases quickly with increasing interaction size, and you should only consider d -way interactions when all their $(d-1)$ -way interactions were significant. For Shapley-based methods, an efficient approximation exists based on random sampling and evaluating feature orderings until the estimates converge. The estimates' variance reduces in $O(1/m)$, where m is the number of evaluated orderings.

Unjustified Causal Interpretation

Technical data science experts are often interested in causal insights into the underlying data-generating mechanisms, which model interpretability methods do not generally provide. Common causal questions include the identification of causes and effects, predicting the effects of interventions, and answering counterfactual questions. For example, a medical researcher might want to identify risk factors or predict average and individual treatment effects. A researcher can therefore be tempted to interpret the result of explainability methods from a causal perspective. A causal interpretation of predictive models is often not possible. Standard supervised ML models are not designed to model causal relationships but to merely take advantage of associations. A model may rely on the causes and effects of the target variable and variables that reconstruct unobserved influences on Y (e.g., causes of effects). Consequently, the question of whether a variable is relevant to a predictive model (indicated e.g., by $PFI > 0$) does not directly indicate whether a variable is a cause, an effect, or does not stand in any causal relation to the target variable.

Summary

This chapter reviewed numerous pitfalls of interpretation techniques, such as bad model generalization, dependent features, interactions between features, or causal interpretations. We hope to encourage a more cautious approach when interpreting ML models in practice to point practitioners to already (partially) available solutions. The stakes are high: ML algorithms are increasingly used for socially relevant decisions, and model interpretations play an important role in decisions. Therefore, we believe that you can benefit from concrete guidance on properties, dangers, and problems of IML techniques. We need to strive toward a recommended, well-understood set of tools, which require much more careful research.

Conclusion

Organizations are people-driven. To make them data-driven, we need to adopt explainability.

—Anirban Nandi

Model explainability and interpretability will soon become the core of every AI and ML organization. All businesses, large or small, that use machine learning must understand the models' explainability. With models becoming more and more complex, it will always be a requirement to build algorithms that can easily explain the inner workings of the models and explain to non-technical stakeholders why a certain decision was taken.

We leave you with some of the best practices to follow for adopting interpretability. We hope that you can use all the information we provided in your day-to-day tasks and find it easier to understand your models.

The following are some tips from Google's AI blog (<https://blog.google/technology/ai/>) on how to deal with interpretability.

Engage Interpretability with a Good Plan

You can bring in the concept of interpretability before, during, and after designing and training your model. You must figure out the right and suitable approach

User Experience and Interpretability Go Hand in Hand

It is very important to involve users in every interpretability pipeline that you build for your models. Use the time with users of the model to iterate and test assumptions about the model. Whenever possible, it is a good practice to enable users to do their own analysis—that is, empower users to understand how inputs affect model output.

Wherever Possible, Design the Model to Be Interpretable

It never hurts to use a simple interpretable model. At least, it could be a starting point for any model-building exercise. Use the smallest set of inputs necessary for your performance goals to clarify what factors are affecting the model, or use the simplest model that meets your performance goals.

Choose Metrics to Reflect the End Goal and the End Task

The metrics you consider must address the benefits and risks of your specific context. For example, a disaster system would need to have a high recall, even if that means the occasional false signal.

Understand the Trained Model

Many techniques are being developed to gain insights into the model (e.g., sensitivity to inputs). Analyze the model's sensitivity to different inputs for different subsets of examples. Understanding the trained model sometimes helps decipher complex relationships.

Communicate Explanations to Model Users

Provide understandable and appropriate explanations for the user; for example, technical details may be appropriate for industry practitioners and academia, while general users may find UI prompts, user-friendly summary descriptions, or visualizations more useful.

References

- “Human Factors in Model Interpretability: Industry Practices, Challenges, and Needs” by Sungsoo Ray Hong, Jessica Hullman, and Enrico Bertini. 2004.
- “To Explain or to Predict?” by Galit Shmueli.
- “Explainability Fact Sheets: A Framework for Systematic Assessment of Explainable Approaches” by Kacper Sokol and Peter Flach.
- “General Pitfalls of Model-Agnostic Interpretation Methods for Machine Learning Models?” by Christoph Molnar, Gunnar Konig, Julia Herbinger, Timo Freiesleben, Susanne Dandl, Christian A. Scholbeck, Giuseppe Casalicchio, Moritz Grosse-Wentrup, and Bernd Bisch.
- “Explaining by Removing: A Unified Framework for Model Explanation” by Ian C. Covert, Scott Lundberg, and Su-In Lee.
- “GLocalX: From Local to Global Explanations of Black Box AI Models” by Mattia Setzua, Riccardo Guidotti, Anna Monrealea, Franco Turinia, Dino Pedreschia, Fosca Giannottib, and Largo B. Pontecorvo.
- “Benchmarking and Survey of Explanation Methods for Black Box Models” by Francesco Bodria, Fosca Giannotti, Riccardo Guidotti, Francesca Naretto, Dino Pedreschi, and Salvatore Rinzivillo. February 2021.
- “Does Dataset Complexity Matters for Model Explainers?” by Jose Ribeiro, Raissa Silva, and Ronnie Alves. July 2021.
- “Machine Learning Interpretability: A Survey on Methods and Metrics” by Diogo V. Carvalho, Eduardo M. Pereira, and Jaime S. Cardoso.

REFERENCES

- “MAGIX: Model Agnostic Globally Interpretable Explanations” by Nikaash Puri, Piyush Gupta, Pratiksha Agarwal, Sukriti Verma, and Balaji Krishnamurthy. <https://ema.drwhy.ai/introduction.html>.
- “Interpretable machine learning. A Guide for Making Black Box Models Explainable” by Christoph Molnar. 2019. <https://christophm.github.io/interpretable-ml-book/>.

Index

A

Accuracy measurement
 benefits, 10
 black-box classifiers, 12
 interpretable models, 13
 meaning, 9
 model performance, 9
 pictorial description, 11
 predictive algorithms, 12
 real-world scenarios, 12
 time-consuming course, 13

Anchors
 advantages, 243
 algorithm's results/explanations, 240
 black-box classification, 238
 classification, 245
 components, 242
 definition, 240
 disadvantages, 244
 flow chart, 243
 hyperparameters, 244
 libraries, 245
 mathematical description, 241–243
 model-agnostic, 239
 perturbation, 239
 perturb instances, 245
 prediction, 246
 representation, 239
 result explanation, 240

Attention-based (AB) methods, 112, 113

B

Bank loan processing application
 binary classification, 18
 classification model, 17
 coefficients, 19
 decision tree, 19
 linear and logistic models, 19
 logistic regression, 18
 neural network representation, 21
 overview, 17
 random forest algorithm, 20
 variables, 18

Bayesian approach, 94

Bayesian information criterion (BIC), 222

Black-box model, 42, 228, 240, 298, 306, 312, 315

C

Ceteris paribus (CP) profiles, 186, 192

Class activation maps (CAM)
 CAMLogger callback, 283
 compute_heatmap method, 284
 draw conclusions, 284
 global average pooling, 280
 Grad-CAM (*see* Gradient-weighted class activation maps (Grad-CAM))
 model prediction (softmax/sigmoid), 283, 284

INDEX

Class activation maps (CAM) (*cont.*)

- modification, 281–283
 - network architecture ideal, 281
- Concept attribution
- AB method, 113
 - feature-based explanations, 112
 - saliency heat-map matrix, 114
 - SH, 112
 - text data, 112

Convolutional neural network (CNN), 3

Counterfactual explanations

- banking software, 250
- comparison, 258–260
- continuous outcome, 251
- definition, 249
- DiCE, 257, 260–268
- generating process
 - components, 253
 - disadvantages, 255
 - feature-wise distances, 253
 - Gower distance, 256
 - MAD, 253
 - numerical and categorical features, 256
 - objectives, 255
 - optimization algorithm, 254
 - producing model, 254
 - quadratic distance, 253
 - square distances, 254
 - trial/error values, 252
- meaning, 251

model-agnostic/specific

- model, 250

multi-objective, 258

prototypes, 257

quality criterion, 252

requirements, 251, 252

D

DALEX

- additive attributions, 189–191
- attributes, 195
- breakdown plots, 190
- CP profiles, 186, 192, 193
- functionalities, 198
- implementation, 188
- instance-level exploration
 - methods, 189
- iBD, 192, 193
- local diagnostics plots, 194
- machine learning models, 188
- model exploration, 186
- model-level explanations
 - model accuracy, 204
 - partial dependence plots, 208
 - parts, 205–207
 - performance, 204, 205
 - pictorial description, 206
 - profile, 207–209
 - variable importance plots, 206
- pipeline model, 196–198
- predictive models, 187

predict-level explanations

- breakdown plot, 201
- function, 198, 199
- predict_parts function, 199–202
- profiles, 202–204

Titanic data set, 194

Data sets

- abstract, 117

attribute information

- administrative variable, 121

- bounce/exit features, 119

- BounceRates variable, 124

- browser details, 127
- categorical attributes, 118
- chart, 130
- correlation matrix, 129
- ExitRates, 125
- features, 118
- informational variable, 122
- Jupyter notebook, 119
- libraries, 119
- month details, 126
- OperatingSystems, 127
- page values/bounce rates *vs.* revenue, 132
- pandas profiling overview, 121
- parameters, 134
- performance metrics, 134
- product-related/exit rates *vs.* Revenue, 133
- region details, 127
- response variable, 118, 130
- revenue variable, 129
- SpecialDay, 126
- TrafficType, 128
- traffic type *vs.* revenue, 131
- VisitorType, 128
- visitor type *vs.* revenue distribution, 131
- weekend details, 128
- weekend *vs.* revenue, 132
- characteristics, 117
- cutting-edge research, 98
- DALEX, 186–208
- different types, 98
- FACET, 170–180
- frameworks, 97
- ICE plot visualizes, 185
- information, 118
- LIME, 163–170
- partial dependence function, 180–185
- permutation, 137–141
- random forest method (*see* Random forest methods)
- SAGE model's, 157–163
- SHAP (*see* Sample handling and analysis plan (SHAP))
- sources, 118
- tabular data (*see* Tabular data sets)
- text data, 98
- Design opportunities, interpretability beneficial approach, 65, 66 communication/summarization model, 66 identification/integrate human expectations, 65 post-deployment phase, 67 scalable/integrable, 66 visualization tools, 67
- Determinantal point processes (DPP), 260, 261
- DiCE advantages, 262 binary_crossentropy, 265 continuous features, 265 counterfactual output, 264 data set, 264 d-dimensional, 260 disadvantages, 263 diversity/feasibility constraints, 261 explanation instance, 267 framework, 260 model loss, 266 optimization, 262 output process, 268 properties, 260

INDEX

- DiCE (*cont.*)
proximity, 261
query instance, 267
sparsity, 261
training and validation accuracy, 266
- E**
- Environment-driven approach, 4
- Explainable systems
functional requirement, 70, 71
applicable model, 72
breadth/scope, 72
caveats/assumptions, 73
compatible feature types, 73
computational complexity, 72
predictive system, 72
problem type, 71
supervised problems, 71
target, 71
- key dimensions, 69
- operational requirements (*see*
Operational requirements)
- operation requirements, 70
- safety requirements, 79
- usability, 76
- validation requirements, 80–82
- Explainability techniques
bad model generalization, 322
causal interpretation, 327
complex models, 322
feature dependencies, 323
aggregation, 325
computational effort, 327
conditional variants, 324
extrapolations/perturbations, 323
- high-dimensional settings, 326
human-intelligibility, 326
interaction effects, 325
interactions, 324
linear correlation/general
dependence, 324
- model and approximation, 325
- model-agnostic methods, 322
- one-fits-all interpretability, 321
- permutation, 321
- pitfalls model, 321
- Explanation methods
algorithmic feasibility, 48
correctness property, 48
dependability, 49
formulas/mathematical concepts, 45
human-friendly, 51
abnormal cases, 52
confirmation bias, 52
contrastiveness, 51
general/probable, 53
selectivity, 51
social interaction, 52
truth, 52
- loyalty, 49
- lucidness, 49
- mobility, 48
- objectives, 46
- originality property tests, 50
- properties, 45, 47
- Rashomon effect, 46
- reliability, 50
- representativeness, 50
- resoluteness, 49
- significance, 50
- transparency, 47

F

- Fault Assisted Circuits For Electronic Training (FACET)
 key components, 170
 machine learning workflow, 171, 172
 model inspection, 171
 redundancy, 177, 179, 180
 simulation algorithms, 171
 source code, 172–175
 synergy, 175–177
- Feature importance (FI), 99, 100

G

- General Data Protection Regulation (GDPR), 33
- Global average pooling (GAP), 280, 281, 287
- Global interpretable model, 37, 38, 236
- GLObal to loCAL eXplainer (GLocalX)
 black-box models, 221
 dendrogram, 222
 global/local explanation problem, 221
 hierarchical merges, 222
 local explanations, 221
 model-agnostic method, 229
 output process, 230–233
 quasi-polyhedra model, 225
 selecting pairs theories, 223–226
 source code, 226–230
- Gradient-only methods, 280
- Gradient-weighted class activation maps (Grad-CAM)
 activating process, 288
 class-discriminative localization map, 288
 class-discriminative saliency map, 290

- convolutional layers, 287, 290
 convolutional neural network, 288
 deep learning models, 288
 differences, 287
 flow chart, 287
 generalization, 287
 image presentation, 286
 input image, 289
 model prediction, 289
 neural network model, 287
 normalization, 292
 post hoc attention, 286
 ReLU, 291
 source code, 288
 target class, 292
 visualization, 286

H

- Human factor model
 anomalies/corner cases, 58
 building/validation stage, 57–59
 comparison method, 58
 components, 55
 data scientist, 60
 decision-making/knowledge discovery, 62
 deployment/maintenances, 59, 60
 gain confidence/trust, 62, 63
 goals, 61
 high-stakes situations, 60
 ideation/conceptualization stage, 57
 model development/improvement, 61
 non-essential/non-meaningful relationships, 61
 reviewers, 56
 root cause analysis, 59
 stakeholders/end users, 56

INDEX

Human factor model (*cont.*)

- technical builders, 56
- technical characterization, 63
 - collaboration, 63
 - context-dependent, 65
 - cooperative, 64
 - mental models, 65
 - process, 64
- tools/insights, 63

I, J, K

Image interpretability methods

- CAM (*see* Class activation maps (CAM))
- compute weights, 277
- deeper networks, 271
- image problem, 273
- neural networks, 271
- pixel attribution methods,
279, 280
- predict class, 276
- problem description, 272
- random perturbations, 274
- source code, 272
- superpixels, 272, 275
- weighted linear model, 277, 278

Individual conditional expectation (ICE),

- 74, 183, 185, 325

Intensive care units (ICUs), 58

Interpretability, 15

- accuracy, 25
- black-box model, 42
- categories, 35, 37, 41, 43
- churn model, 38
- classification, 43
- concepts, 25
- definition, 24

explanation methods (*see* Explanation methods)

human factor (*see* Human factor model)

humans explanation, 15–17

intrinsic/explanations, 36

post hoc model, 36

- black-box model, 40

- datapoints, 40

- explanation, 40

- global model, 37, 38

- interpret black-box

- models, 40

- local model, 38, 39

- model-agnostic methods, 39

- model internals, 40

predictions, 39

pre-model, 36

prime motivation

- accuracy, 28

- computational/societal bias, 27

- decision, 27

- develop tools and processes, 27

- displacement strategies, 28

- large-scale systems, 29

- reproduce operations, 28

- requirements, 26

- security risks, 29

research method

- academic publications, 32

- driven industries, 30

- entities, 30

- explanation technologies, 33

- Google trends, 32

- high-stake decisions, 29

- ML problem, 31

scope-related types, 37

transparency, 25

Interpretable/explainable ML,
 differences, 83
 decision tree, 84
 digging deeper, 85
 explainable ML
 accuracy, 88
 correct/positive label, 88
 critical issues, 87
 unfaithful model, 88
 features/components, 95
 interpretable ML
 efforts/construct, 90
 hidden pattern, 90
 key issues, 89
 profits *vs.* loses, 89
 recidivism tool, 89
 interpretable models working, 84
 model selection, 94
 predictive/explanatory modeling
 aspects, 91
 bias-variance, 92
 causation-association, 92
 conduct proper modeling, 92
 differences, 91
 explanatory, 93
 model creation, 91
 objective, 91
 retrospective, 92
 theory-data, 92
 predictive likelihood approach, 94
 statistical conclusions, 94
 statistics/parameters, 83
 validation methods, 93

L

LIME, 163–170
 black-box model, 165

categorical variables, 167
 decision function, 165
 evaluation, 168
 features/interpretable data
 representations, 164
 image interpretability methods, 272
 key requirement, 164
 libraries, 167
 lime output, 164
 local surrogate models, 166
 model interpretation, 163
 model-specific approaches, 163
 null values, 168
 PCA, 166
 random forest model, 168
 sampling code, 167
 single instance, 169, 170
 tabular data, 165
 text classification models
 data description, 299
 disadvantage, 302
 document selection, 300
 output process, 301, 302
 python class, 301
 source code, 299
 visualization, 299, 301
 working process, 298
 train and test data, 168
 Local model interpretability, 38, 39

M

Machine learning techniques
 accuracy measurement, 9
 algorithms, 4
 advantages, 6
 components, 4
 descriptions, 5

INDEX

Machine learning techniques (*cont.*)
 steps, 4
 template/list overview, 5, 6
bank loan processing (*see* Bank loan processing application)
definition, 1
evolution, 1–3
investigation
 algorithm selection, 7
 classification problems, 8
 experiment/report results, 8
 metrics/predictions, 7
 question identification, 7
 repeat process, 9
model accuracy, 10, 12, 13
model performance, 9
Median absolute deviation (MAD), 253
Model-Agnostic Global Interpretable Explanations (MAGIE)
 algorithm approach, 215
 algorithm execution, 219
 class coverage rule, 214, 215
 genetic algorithm, 212
 input data preprocesses, 216
 instance level conditions, 216
 learning rules, 216–218
 length rule, 214
 mutual information rule, 214
 optimization, 219
 postprocessing, 220
 postprocessing rules, 218
 precision/coverage, 217
 precision rule, 213, 214
 requirements, 217
 sorting rules, 218–220
 structuring step, 220
 symbols/definitions, 213
Modeling techniques, 315

boxplot graph, 318
boxplot graphs, 318
explainability ranks, 320
framework, 315
gradient boosting models, 319
properties, 315, 317
random forest, 316
Spearman rank correlation, 317
tree-based ensembles, 316
Multi-Objective Counterfactuals (MOC), 256, 258

N

Network in Network (NiN), 280
Neural Information Processing Systems (NIPS), 31

O

Occlusion/perturbation-based methods, 280
Operational requirements
 audience, 75
 causality/actionability, 76
 data/model transformations, 75
 domain, 75
 explanation family, 73
 function, 76
 mediums, 74
 system interaction, 74
 trust/performance, 76

P, Q

Partial dependence plot (PDP)
 causal interpretation, 183
 feature dependence, 181

feature distribution, 183
 features, 180
 heterogeneous effects, 183
 interactions, 182
 linear relationship, 180
 Monte Carlo method, 180
 numerical features, 182
 partial function, 181
 source code, 184

Permutation feature importance models
 advantages, 139
 characteristics, 137–141
 concepts, 138
 disadvantages, 139
 mean squared error, 138
 measurement, 138
 original model error, 138
 source code, 139, 141

Pixel attribution methods, 279, 280

Prototype selection (PS), 110

R

Random forest methods
 accuracy-based importance, 135
 characteristics, 134
 Gini impurity, 135–137

Reinforcement learning, 4, 238, 239, 244

Removal-based methods, 103–105

Rule-based methods
 advantage, 211
 anchors, 238–246
 GlocalX, 221–233
 MAGIE (*see* Model-Agnostic Global Interpretable Explanations (MAGIE))
 monotonic response functions, 211
 skope-rules, 233–238

S

Safety requirements
 information leakage, 79
 invariance, 80
 predictive model, 80

Sample handling and analysis
 plan (SHAP)
 accuracy, 153
 building model, 152, 153
 categorical variables, 151
 characteristics, 141
 class labels output, 151
 components, 142
 consistency, 148
 data frame shape, 152
 dependent and independent variables, 151, 152
 features, 142
 f-feature-models, 145
 force plot, 156, 157
 formula, 146, 147
 fundamental characteristics, 147
 KernelSHAP method, 149
 LIME weights, 149
 linear function, 147
 local accuracy, 148
 marginal value, 146
 missingness constraints, 148
 node representation, 142
 non-zero entries, 148
 PageValues variable, 154
 predictions, 143, 149
 red highlighted edges, 144
 revenue column, 151
 SAGE values, 161–163
 Shapely values, 142, 154
 source code, 154

INDEX

- Sample handling and analysis
 plan (SHAP) (*cont.*)
 summary plot, 155
 TreeSHAP, 150
 weighted marginal contribution, 144
- Sample Handling and Analysis
 Plan (SHAP)
- text classification
 ELI5 output, 310
 logistic regression, 307
 logistic regression model, 309, 310
 modeling information, 307
 output process, 305, 308, 311
 sentence highlighting (SH), 306–313
 source code, 303
 TextExplainer, 312
 training documents, 304
 visualization, 302
- Semi-supervised learning, 3, 4, 42, 71
- Shapley Additive Global
 ImportancE (SAGE)
 additive importance measures, 159
 characteristics, 157
 code implementation, 159
 conditional distribution, 158
 flowchart, 158
 intrinsic properties, 159
 marginal distribution, 159
 model function, 160
 model interpretation, 159
 performance, 160
 plot output, 161
 SHAP values, 161–163
 train and test data, 160
- Skope-rules
- decision rules, 233
different applications, 236
implementation, 235–238
methodology, 234, 235
- Supervised learning, 3, 100
- Support-vector machines (SVM), 7

T

- Tabular data sets
 additive measurement, 102, 103
 counterfactuals, 110, 111
 data-driven methods, 100
 FI, 99, 100
 features, 100
 frameworks, 99
 image data explanations, 111
 LIME/SHAP output, 108
 mathematical tools, 106
 model behaviors, 107
 predictive power, 100–102
 prototypes, 109, 110
 removal-based methods, 103–105
 rule-based methods, 109, 110
 saliency maps, 111
 subset function, 106
 summarizing feature influence, 107
 visual comparison, 111
- Text classification models
 data preprocessing, 296–298
 LIME classification, 298–302
 map documents, 295
 natural language processing, 295
 SHAP, 302–313
 stack overflow tags, 295
 system/environment, 297

U

- Unified framework, 104, 105
- Unsupervised learning, 3, 42, 71
- Usability requirements
 - actionability, 78
 - bidirectional communication, 77
 - completeness, 77
 - complexity, 78
 - contextfullness, 77

- interactivity, 77
- novelty, 78
- personalization, 79
- representativeness, 77
- soundness, 77

V, W, X, Y, Z

- Validation requirements, 80–82