Chapter 1. An Overview of Explainability

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 2nd chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *rfernando@oreilly.com*.

Explainability has been a part of machine learning since the inception of AI. The very first AIs, rule-based chain systems, were specifically constructed to provide a clear understanding of what led to a prediction. The field continued to pursue explainability as a key part of models, partly due to a focus on general AI but also to justify that the research was sane and on the right track, for many decades until the complexity of model architectures outpaced our ability to explain what was happening. After the invention of ML neurons and neural nets in the 1980s,[1] research into explainability waned as researchers focused on surviving the first AI winter by turning to techniques that were “explainable” because they relied solely on statistical techniques that were well-proven in other fields. Explainability in its modern form (and what we largely focus on in this book) was revived, now as a distinct field of research, in the mid 2010s in response to the persistent question of “this model works really well… but how?”

In just a few years, the field has gone from obscurity to one of intense interest and investigation. Remarkably, many powerful explainability techniques have been invented, or repurposed from other fields, in the short time since. However, the rapid transition from theory to practice, and the increasing need for explainability from users who interact with ML, such as users and business stakeholders, has led to growing confusion about the capability and extent of different methods. Many fundamental terms of explainability are routinely used to represent different, even contradictory, ideas, and it is easy for explanations to be misunderstood due to practitioners rushing to provide assurance that ML is working as expected. Even the terms explainability and interpretability are routinely swapped, despite having very different focuses. For example, while writing this book, we were asked by a knowledgeable industry organization to describe explainable and interpretable capabilities of a system, but the definitions of explainability and interpretability were flipped in comparison to how the rest of industry defines the terms! Recognizing the confusion over explainability, the purpose of this chapter is to provide a background and common language for future chapters.

What Are Explanations?

When a model makes a prediction, Explainable AI methods generate an explanation that gives insight into the model's behavior as it arrived at that prediction. When we seek explanations, we are trying to understand "*why* did X happen?" Figuring out this "Why" can help us build a better comprehension of what influences a model, how that influence occurs, and where the model performs (or fails). As part of building our own mental models, we often find a pure explanation to be unsatisfactory, so we are also interested in explanations which provide a *counterfactual*, or *foil*, to the original situation. Counterfactuals are scenarios which seek to provide an opposing, plausible, scenario of why X did not happen. If we are seeking to explain "why did it rain today?" we may also try to find the counterfactual explanation for "why did it not rain today [in a hypothetical world]?" While our primary explanation for why it rained might include temperature, barometric pressure, and humidity, it may be easier to explain that it did not rain because there were no clouds in the sky, implying that clouds are part of an explanation for why it does rain.

We also often seek explanations that are causal, or in the form of "X was predicted because of Y." These explanations are attractive because they give an immediate sense of what a counterfactual prediction would be: remove X and presumably the prediction will no longer be Y. It certainly sounds more definitive to say "it rains because there are clouds in the sky." However, this is not always true; rain can occur even with clear skies in some circumstances. Establishing causality with data-focused explanations is extremely difficult (even for time-series data), and no explainability techniques have been proposed that are both useful in practice and have a high level of guarantee in their analysis. Instead, if you want to establish causal relationships within your model or data, we recommend you explore the field of interpretable, causal models.

Explainability Consumers

Understanding and using the results of Explainable AI can look very different depending on who is receiving the explanation. As a practitioner, for example, your needs from an explanation are very different from those of a non-technical individual who may be receiving an explanation as part of an ML system in production that they may not even know exists!

Understanding the primary types of users, or personas, will be helpful as you learn about different techniques so you can assess which will best suit your audience's needs. In Chapter 7, we will go into more detail about how to build good experiences for these different audiences with explainability.

More broadly, we can think of anyone as a consumer of an explanation. The ML system is presenting additional information to help a human perceive what is unique about the circumstances of a prediction, comprehend how the ML system behaves, and, ultimately, be able to extrapolate to what could influence a future prediction.

Currently, a key limitation of Explainable AI is that most techniques are a one-way description; the ML system communicates an explanation to the human, with no ability to respond to any follow-on requests by the user. Put another way, many Explainable AI methods are talking *at* users rather than *conversing* with users. However, while these techniques are explaining an ML system, none would be considered to be machine learning algorithms. Although very sophisticated, these techniques are all “dumb” in the sense that we cannot interact with them in a two-way dialogue. A smart explainability technique could adapt to our queries, learning how to guide us towards the best explanation, or answer the question we didn’t know we were asking. For now, if we want to try and obtain more information about a prediction, the best we can do is to change the parameters of our explanation request, or try a different explainability technique. In Chapter 8, we outline how this will change in the future, but in the meantime, most explainability techniques represent a process closer to submitting a requisition form to an opaque bureaucracy to get information than going to your doctor and engaging in a conversation to understand why you have a headache. In our work with Explainable AI, we have found that it is useful to group explainability into three broad groups based on their needs: practitioners, observers, and end-users.

Practitioners: Data Scientists and ML Engineers

ML *practitioners* primarily use explainability as they are building and tuning a model. Their primary goal is to map an explanation to actionable steps which can be taken to improve the model’s performance, such as changing the model architecture, training data, or even the structure of the dataset itself before the model is deployed. The goal of this process is often to improve the training loss and validation set performance, but there are times where accuracy may be not the primary concern. For example, a data scientist may be concerned that the model has become influenced by data artifacts not present in the real-world data (for example, a doctor’s pen marks on an X-ray for a diagnosis model) or that the model is generating outcomes that are unfairly biased towards certain individuals.

However, a ML engineer may be asking “how can we improve the performance of our data pipeline?” Certain features may be costly to obtain, or computationally expensive to transform into a usable form. If we find that the model rarely uses those features in practice, then it is an easy decision to remove them to improve the system overall. Practitioners may also be interested in explainability once the system is deployed, but their interest still remains in understanding the underlying mechanisms and performance. Explainability has proven to be a robust and powerful tool for monitoring deployed models for drift and skew in their predictions, indicating when a model should be retrained.

And of course, you may simply be interested in an explanation because, like any practitioner, you’ve encountered a situation when the model made you squint and say “what the...?”

Observers: Business Stakeholders & Regulators

Another group of explainability consumers is *observers*. These are individuals, committees, or organizations, who are not involved in the research, design, and engineering of the model, but also are not using the model in deployment. They often fall into two two categories: stakeholders, who are within the organization building the model, and regulators, who are outside the organization.

Stakeholders often prefer a non-technical explanation, instead seeking information that will allow them to build trust that the model is behaving as is expected, and in the situation it was designed for. Stakeholders often come to an explanation with a broader, business-focused question they are trying to answer- “do we need to invest in more training data?” or “how can I trust that this new model has learned to focus on the right things so we're not surprised later?"

Regulators are often from a public organization or industry body, but they may also come from another part of a company, (i.e., Model Risk Management) or an auditor from another company, such as an insurance company. Regulators seek to validate and verify that a model adheres to a specific set of criteria, and will continue to do so in the future. Unlike stakeholders, a regulator's explainability needs can range from quite technical to vague depending on the regulation. A common example of this conundrum is in the needs of many regulators to assess evidence that a model is not biased towards a specific category of individuals (e.g., race, gender, socioeconomic status) while also determining that model behaves fairly in practice, with no further definition given for what entails fairness. Since regulators may routinely audit a model, explanations that require less human effort to produce or understand, and can be generated efficiently and reliably, are often more useful.

End-Users: Domain Experts & Affected Users

Individuals or groups who use, or are impacted by, a model's predictions are known as end-users. *Domain experts* have a sophisticated understanding of the environment the model is operating in, but may have little to no expertise in machine learning, or even the features used by the model if they are derived, or new to the profession. Domain experts often use explanations as part of a decision support tool. For example, if an image model predicts manufacturing defects in parts on an assembly line, a quality control inspector may use an explanation highlighting where the model found the defect in the machined part to help make a decision about which part of the manufacturing process is broken.

Affected users often have little-to-no understanding of how the model works, data it uses, or what that data represents. We refer to these users as affected because they may not directly use the model, but the prediction results in a tangible impact on them. Examples of affected individuals include people receiving credit offers, where a model has predicted their ability to repay loans, or a community receiving increased funding for road maintenance. In either case, the affected users primarily want to understand and assess if the prediction was fair and that it was based on correct information. As a follow up, these users seek explanations which can give them the ability to understand how they could alter factors within their control to meaningfully change a future prediction. You may be unhappy that you were given a loan with a very, very high interest rate, but understand that it is fair because you have a poor history of repaying loans on time. After understanding the current situation, you might reasonably ask "how long of a history of on-time loan payments would I need to establish in order to get a lower interest rate?"

Types of Explanations

Modern day machine learning solutions are often complex systems incorporating many components from data processing and feature engineering pipelines, to model training and development to model serving, monitoring and updating. As a result, there are many factors that contribute to explaining why and how a machine learning system makes the predictions it does and explainability methods can be applied at each step of the ML development pipeline. In addition, the format of an explanation can depend on the modality of the model (e.g., whether it is a tabular, image, or text model). Explanations can also vary from being very precise and specific by being generated for a single prediction or based upon a set of predictions to give a broader insight into the model's overall behavior. In the following sections we'll give a high level description of the various types of explanations that can be used to better understand how ML solutions work.

Pre-modeling Explainability

Machine learning models rely on data and although many Explainable AI techniques rely on interacting with a model, the insights they create are often focused on the dataset and features. Thus, one of the most critical stages of developing model explanations begins before any modeling takes place and is purely data focused. Pre-modeling explainability is focused on understanding the data or any feature engineering that is used to train the machine learning model.

As an example, consider a machine learning model that takes current and past atmospheric information (like humidity, temperature, cloud cover, etc) and predicts the likelihood of rain. An example of an explainable prediction that is model dependent would be "The model predicted a 90% chance of rain because, among the data inputs, humidity is 80% and cloud cover is 100%". This type of explanation relies on feature attribution for that model prediction. On the other hand, pre-modeling explanations focus only on the properties of a dataset and are independent of any model, such as "The standard deviation of the chance of rain is +/- 18%, with an average of 38%." Inherently explainable models, such as linear and statistical models, may blur this distinction, with explanations such as "For each 10% increase in the cloud cover, there is a 5% increase in the chance of rain." Given the extensive availability of resources available on more "classical" statistical and linear modeling, we will only briefly discuss here commonly used pre-modeling explainability techniques.

Explanations that focus solely on the dataset are often referred to as Exploratory Data Analysis (EDA). EDA is a collection of statistical techniques and visualizations that are used to gain more insight into a dataset. There are many techniques for summarizing and visualizing datasets and there are quite a few useful tools that are commonly used such as Know Your Data, Pandas Profiling, and Facets. These tools allow you to quickly get a sense of the statistical properties of the features in your dataset such as the mean, standard deviation, range, and percentage of missing samples as well as the feature dimensionality and presence of any outliers. From the perspective of explainability, this knowledge of the data distribution and data quality is important for understanding model behavior, interpreting model predictions, and exposing any biases that might exist.

In addition to these summary univariate statistics, explanations in the form of EDA can also take the form of multivariate statistics that describe the relationship between two or more variables or features in your dataset. Multivariate analysis can also be used to compute statistics to show the interaction between features and the target. This type of correlation analysis is useful not just for helping to explain model behavior but can also be beneficial for improving model performance. If two features are highly correlated, this could indicate an opportunity to simplify the feature space which can improve interpretability of the machine learning model. Also, knowledge of these interdependencies is important when analyzing your model using other explainability techniques; e.g. see in Chapter 3 where we discuss the effect highly correlated features has on interpreting the results of techniques like partial dependence plots or other techniques for tabular datasets. There are a number of visualization tools that can assist in this type of correlation analysis such as pair plots, heatmaps, biplots, projection plots (T-SNE, MDS, etc.), and parallel coordinates plots.

Intrinsic vs. Post-Hoc Explainability

When and how do we receive an explanation for a prediction? Explanations that are part of the model's prediction itself are known as *intrinsic* explanations, while explanations that are performed after the model has been trained and rely on the prediction to create the explanation are called *post-hoc* explanations. Most of the techniques we discuss in this book are post-hoc explanations because they are more portable and can be decoupled from the model itself. By contrast, generating intrinsic explanations often requires modifications to the model itself or using an interpretable model. However, you may notice that some libraries are set up to provide an explanation with the prediction- this does not necessarily mean the explanation is intrinsic, as it may be that the service first generates the prediction, then the explanation, before returning both together.

Within the group of post-hoc explainability techniques, another factor to group techniques is whether the method is *model-agnostic* or *model-specific*. A model-agnostic technique does not rely on the model's architecture, or some inherent property of the model, so it can be used universally across many different types of models, datasets, and scenarios. As you might imagine, it is also more useful to become more familiar with these techniques because you will have more opportunities to reuse them than a technique which only works on a specific type of model architecture.

How can a technique not know anything about the model, and yet still generate useful explanations? Most of these techniques rely on changing inputs to the model, correlating similar predictions, or running multiple simulations of a model. By comparison, a technique that relies on the model itself will leverage some internal aspect of the model's architecture to aid in generating the explanation. For example, an explanation for tree-based models may look at the weights of specific nodes within the tree to extract the most influential decision points within the tree to explain a prediction.

Does this mean that *opaque*, or black-box, models[2] always use model-agnostic techniques, while explainability for transparent, or interpretable, models is always model-specific? Not necessarily, there are many unique explanation techniques for Deep Neural Networks (DNNs) which are considered opaque models, and a linear regression model could be explained using Shapley values, a model-agnostic technique.

Local, Cohort, and Global Explanations

Explanations themselves can cover a wide variety of topics, but one of the most fundamental is whether the explanation is *local, cohort*, or *global* with respect to the range of predictions the explanation covers. A local explanation seeks to provide context and understanding for a single prediction. A global explanation provides information about the overall behavior of the model across all predictions. Cohort explanations lie in between, providing an understanding for a subset of predictions made by the model.

Local explanations may be similar for comparable inferences, but this is not an absolute and depends on the technique, model, and dataset. Put another way, it is rarely safe to assume that an explanation for one set of inputs and inference is blindly applicable to a similar set of inputs and/or prediction.

Consider, for example, a decision tree model that predicts expected rainfall with a decision node that strictly checks whether the humidity is greater than 80%. Such a model can lead to very different predictions and explanations for two days when the humidity is 80% vs. 81%.

Global explanations can come in many forms, and because they are not directly representative of any individual prediction, they likely represent more of a survey, or summary statistics, about the model's behavior. How global explanations are generated is highly dependent on the technique, but most approaches rely either on generating explanations for all predictions (usually in the training or validation dataset) and then aggregating these explanations together, or on perturbing the inputs or weights of the model across a wide range of values.

Global explanations are typically useful for business stakeholders, regulators, or for serving as a guide to compare the deployed model's behavior against its original performance.

> **WARNING**
>
> Global explanations are not a set of rules describing the boundaries of a model's behavior, but instead only what has been observed based in the original training data. Once a model is deployed and is exposed to novel data not seen during training, it is possible that during inference local explanations can differ from or even directly contradict global explanations.

Explainable methods for cohorts are usually the same techniques as what is used to calculate global explanations, just applied to a smaller set of data. One powerful aspect of cohort explanations is that one cohort can be compared against another cohort to provide insights about the global behavior of the model that may not be apparent if we just sought a global explanation. These comparisons are useful enough in their own right to serve as the underpinnings for another pillar of Responsible AI: testing and evaluating the Fairness of a model. Fairness techniques seek to evaluate how a model performs for one cohort of predictions compared to another, with the goal of ensuring that the model generates similar predictions based on relevant factors rather than discriminating on characteristics that are deemed to be unimportant, or may not even be desirable to use in the first place. A classic example is how an AI that seeks to determine whether an individual should be approved for a loan may be trained on historical data in the US, which contains prevalent discrimination against people of different races. Scrutinizing the ML for Fairness would tell us whether the model learned this racial discrimination, or whether it is basing its decisions solely on the relevant financial background of the individual such as their income and history of on-time loan payments. Although Explainability and Fairness share many of the same underpinnings, how to apply and understand Fairness, as well as the techniques themselves, are sufficiently distinct from Explainability that we do not cover them in this book.

Attributions, Counterfactual, and Example-based

Explanations can come in many forms depending on the type of information they use to convey an understanding of the model. When asked, many of us think of *attribution*-based explanations that are focused on highlighting relevant properties of the system, e.g., "It is raining because there are clouds in the sky." In this case, the clouds are an attribute of the sky, a feature in our weather model. Proponents and opponents are part of *counter-factual* explanations, which humans often find more satisfying than pure attribution-based explanations. Earlier when we said it was raining because of clouds, we could have also explained this by providing a counter-factual explanation: “It does not rain when it is sunny.” In this type of explanation, “clouds” would be the *proponent* of our counterfactual explanation, while sunny is the *opponent* to our explanation. Counterfactuals are often portrayed as negations (e.g., “it does **not** rain when it is sunny”) but this is not a requirement of counterfactual explanations. We could have just as easily found a counterfactual to a weather prediction for cold weather with the counterfactual “It is hot when it is sunny.” Finding, and understanding the causes behind proponents and opponents can be difficult depending on the modality of the dataset. An opponent value that is negative in a structured dataset is much easier to comprehend than why a texture in an image is considered an opponent by the model.

In Explainable AI, *example-based,* or *similarity,* explanations are those which focus on providing a different scenario that is analogous to the prediction being explained. For example, in our earlier explanation, we could have also said “It is raining because the weather is mostly like the conditions when it rains in Rome.” To ease the burden of understanding why similar predictions are relevant, example-based explanation techniques typically include secondary information to help highlight the similarities and differences in the dataset and/or how the model acted between the two predictions.

Themes Throughout Explainability

Explainability is a broad and multi-disciplinary area of machine learning that brings together ideas from various fields from game theory to social sciences. As a result there is a large and continually growing number of explainability methods and techniques that have been introduced. In this section, we’ll give an overview of some common themes that have been introduced and further developed the field.

Feature Attributions

Feature attribution methods are common throughout Explainable AI. What does it mean to attribute a prediction to an individual feature in your dataset? Formally, a feature attribution represents the influence of that feature (and its value for a local explanation) on the prediction. Feature attributions can be absolute, for example if a predicted temperature is 24° C, a feature could be attributed 8° C of that predicted value, or even a negative value like -12° C. Feature attributions can also be relative, representing a percentage of influence compared to other features used by the model.

> **NOTE**
>
> In this book, we often describe features as influencing a model, while the specific amount of influence is the attribution. In practice, “feature influence” and “feature attribution” are often used interchangeably.

If the idea of a feature having influence that is relative seems strange to you, then you’re in good company. Understanding what feature attributions convey as explanations, and what they don’t, is rife with confusion and it is often difficult for end-users to build an understanding of feature attributions’ true mechanism. For a more intuitive feel of how feature attributions work, let’s walk through an imaginary scenario of an orchestra playing music. Within this orchestra, there are a variety of musicians (which we will treat as features) that contribute to the overall performance of how well the orchestra plays a musical composition. For Explainable AI, let’s imagine that the orchestra is going to participate in ExplainableVision, a competition where a judge (the ML model) tries to predict how well an orchestra performs music by listening to the music played by each individual musician in the orchestra. In our analogy, we can use feature attributions to understand how each musician influenced the overall rating of the orchestra given by the judge.

Each feature attribution represents how a musician sways the judge towards the most accurate prediction of the musical talent of the orchestra. Some musicians may be useful to the judge in determining an accurate score of the overall orchestra’s performance, so we would assign them a high value for their feature attribution. Perhaps it is because they are close to the average talent of the orchestra, or the amount of time they spend playing in any given performance is very high. However, other musicians may not be as helpful and cause the judges to give an inaccurate score, in which case we would give them a small feature attribution. This low score could be for a number of reasons. For example, it could be because those musicians are just bad at playing music, which is distracting to the judge. Or perhaps, they’re fine musicians but they play out of harmony with the rest of the orchestra. Or maybe they’re fantastic musicians and cause the judges to give a very positive assessment of the orchestra’s talent, even though the rest of the orchestra is really bad. In any of these cases, the feature attribution of that musician should be given a smaller value because their contribution negatively affects the judge’s ability to provide an accurate score of the orchestra’s overall skill. For either high or low feature attribution scores, the assessment to determine the attribution of a single musician to the judge’s overall prediction of the orchestra’s talent is still “how much did that musician influence the judge’s rating?”

Choosing an appropriate feature attribution technique is not just a matter of finding the latest or most accurate state-of-the-art method. Techniques can vary wildly, or even be completely opposed in the attribution they give to different features. Figure 2-1 shows an example of this by comparing seven different feature attribution techniques for the same dataset and model:

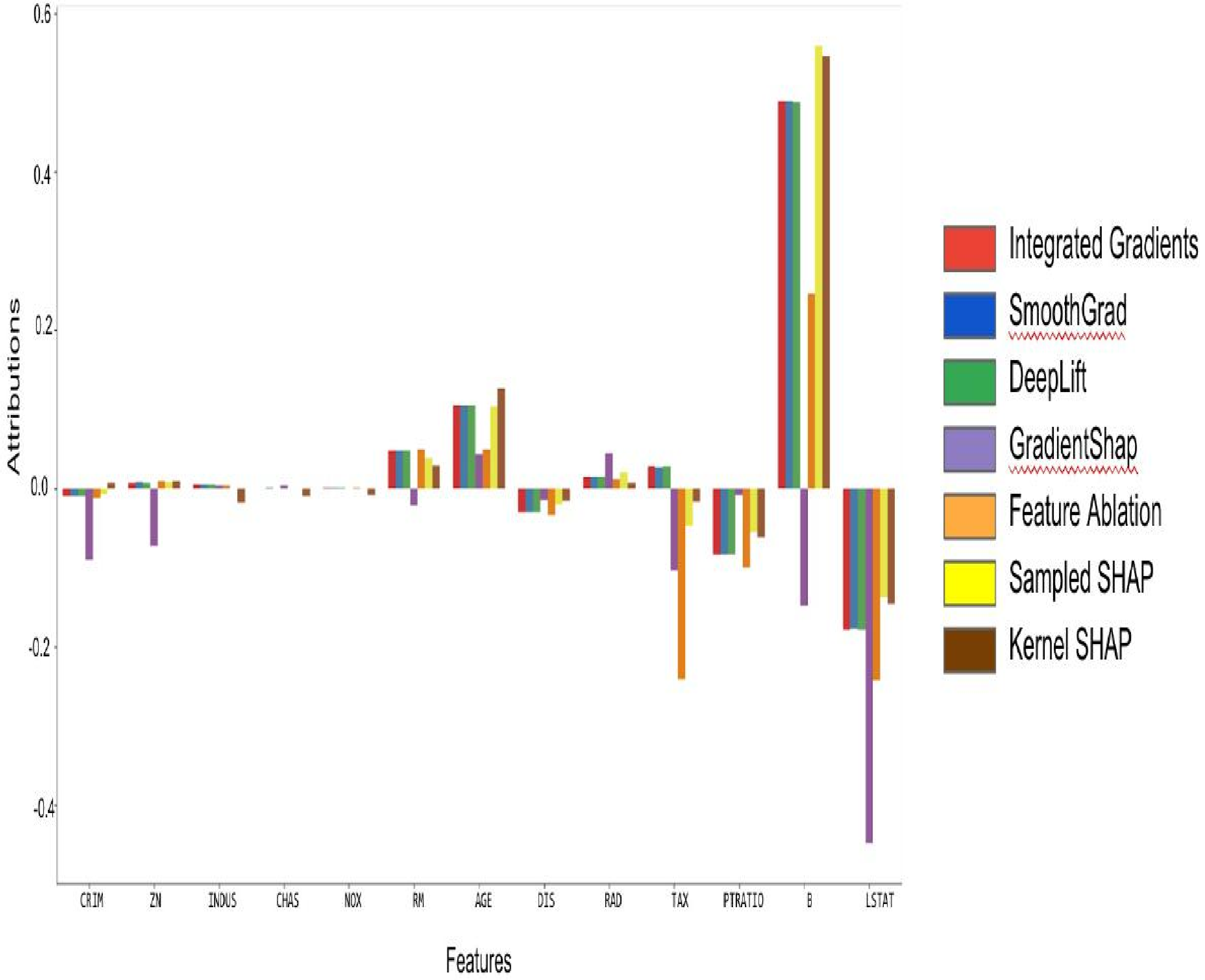

Figure 1-1. Feature attributions across seven different techniques for a PyTorch model trained on the Boston Housing dataset.

In this example, the Integrated Gradients (IG), SmoothGrad, and DeepLift techniques have close agreement in their feature attribution values, despite the fact that DeepLift is conceptually a different approach from IG and SmoothGrad. Conversely, the GradientSHAP technique strongly diverges from its sibling techniques, Sampled SHAP and Kernel SHAP, despite all three using the same underlying theory. Finally, we can see how Feature Ablation, which is not similar to any of the other methods, sometimes has very similar attributions to the other techniques (i.e., for RM and PTRATIO), but has attributed influence to B and TAX unlike any other method.

Feature attributions may still seem abstract, which is okay because they rely on the modality of the dataset. For example, in Chapter 3, Explainability for Tabular Data, we will discuss feature attributions as numerical values because the inputs are scalar. By comparison, in Chapter 4, Explainability for Computer Vision, feature attributions are the contribution by individual pixels in the image to the prediction. In Chapter 5, which is focused on NLP, the attributions are typically tokenized words.

Shapley values

One commonly used method to determine feature attribution is Shapley values. Shapley values use game theory to determine a feature's influence on a prediction. Unlike a technique like feature permutation (see Chapter 3 where we discuss Permutation Feature Importance), which relies on changing the values of features to estimate their impact, Shapley values are purely observational, instead inferring feature attributions through testing combinations with different groups of features.

A Shapley value is calculated using cooperative game theory; Lloyd Shapley published the technique in 1951, and it contributed to his 2012 Nobel prize in Economics. It is useful to understand the core idea behind Shapley Values in order to decide if they're worth it for your use case, and be able to use them effectively with your ML model. Shapley Values rely on examining how each feature influences the predicted value of a model by generating many predictions based on a partial set of the features used by the model and comparing the results of the predicted values.

We can describe how Shapley values are computed in two ways, coalitions and paths. With coalitions, Shapley are represented by grouping the features of a dataset into multiple, overlapping subsets, which are the coalitions. For each coalition, a prediction value is generated using only the features in that coalition and compared against the prediction for a coalition that includes one additional feature. Calculating Shapley values can also be framed as a dynamic programming problem, where a series of steps, or paths, are generated. Each step in a path represents incrementally including another feature from the dataset to generate the prediction. We generate many different paths to represent the different permutations of the order in which features could be included.

More concretely, imagine we had a weather dataset, and we wanted to predict the amount of rain (in inches). In this example, we'll refer to this prediction function as `P()` which takes any subset of features, e.g., `{feature_1, feature_2, …}` as its input. For the purposes of explaining how Shapley Values are calculated, it is not important to understand how`P()` determined the predicted values.[3] The ability to explain an opaque model, while not knowing how it works internally, is one of the advantages of many Explainability techniques.

In the simplest version of our weather dataset, we just have two features, `temperature` and `cloud_cover`. To calculate the Shapley Values for each of the individual features `temperature` and `cloud_cover`, we first compute four individual predictions for different combinations of features in the dataset. For now, don't worry about how we arrived at these predicted values using only a subset of features—we'll discuss that later in this chapter, as well as in Chapter 3, in the section on Baselines.

1. `P({}) = 0` // Initially no features, also known as a null baseline
2. `P(temperature) = 2 inches`
3. `P( cloud_cover ) = 5 inches`
4. `P( {temperature, cloud_cover }) = 6 inches`

Our prediction that uses all of the features in the dataset is `P({temperature, cloud_cover})`, which gives us an estimated 6 (inches) of rain. To determine the Shapley Value individually for `temperature` we first remove our `P(cloud_cover)` prediction (5) from the overall `P({temperature, cloud_cover})` (6), leading to a contribution of 1. However, this is only part of the Shapley value, to compute the entire path we also need to move backwards from P(temperature) to P({}), leading to a contribution of 2. We then average the contributions from each step on the path to arrive at a Shapley value of 1.5 for `temperature`.[4] Using the same approach, we can calculate the Shapley value for `cloud_cover` is 4.5 (by averaging 6 - 2 and 5-0).

This reveals a useful property of Shapley Values, *Efficiency*, meaning that the entire Prediction is equal to the sum of the Shapley values of individual features. In our example above, our combined contributions (Shapley values) of 1.5 (`temperature`) and 4.5 (`cloud_cover`) summed to 6 (our original prediction which included all of the features).

In this example, we have two paths, one to calculate the contribution of temperature and another to calculate the contribution of cloud_cover:

1. `P({}) → P({temperature}) → P({temperature, cloud_cover})`
2. `P({}) → P({cloud_cover}) → P({temperature, cloud_cover})`

What happens if we have more than two features? To accomplish this, we begin computing what are known as Shapley paths, or unordered, incremental groupings of features (also called coalitions). A path represents a way to go from no features in our prediction to the full set of features we used in our model. Each step in the path represents an additional feature in our coalition.

Let's expand our weather dataset to include a `humidity` feature, for an overall prediction that is `P({temperature, cloud_cover, humidity})`.

Our Shapley paths are now:

1. `P({}) → P({temperature}) → P({temperature, cloud_cover}) → P({cloud_cover, temperature, humidity})`
2. `P({}) → P({temperature}) → P({humidity, temperature}) → P({cloud_cover, temperature, humidity})`
3. `P({}) → P({cloud_cover}) → P({temperature, cloud_cover}) → P({cloud_cover, temperature, humidity})`
4. `P({}) → P({cloud_cover}) → P({cloud_cover, humidity}) → P({cloud_cover, temperature, humidity})`
5. `P({}) → P({humidity}) → P({humidity, temperature}) → P({cloud_cover, temperature, humidity})`
6. `P({}) → P({humidity}) → P({cloud_cover, humidity}) → P({cloud_cover, temperature, humidity})`

You may have noticed that we have repeated parts in our paths, such as `P({humidity, temperature})`. The ordering of the features does not matter for how Shapley values are organized into paths or coalitions. This is an important property, and the one most misunderstood, as we'll discuss in further detail later regarding a common misconception with Shapley values and causality.

Returning to our weather dataset, how would we calculate the Shapley Value for our `humidity` feature? First we need to know the predicted value for our different combinations of features:

1. P({}) = 3
2. P(temperature) = 2
3. P(cloud_cover) = 4
4. P(humidity) = 5
5. P({temperature, cloud_cover}) = 6
6. P({temperature, humidity}) = 8
7. P({cloud_cover, humidity}) = 10
8. P({cloud_cover, temperature, humidity}) = 15

In our earlier example, we did not need to calculate the indirect contribution of any feature. Now with multiple intermediate steps in our path, we need to determine how much each feature contributed along the path to the final predicted value. For the first path we expand each step in the path to include the contribution made by the new feature added to the coalition. To calculate the attributions, we take the difference of the predicted value before and after the step occurs:

Step 1 (Base Case):

```
P({}) → P({temperature})
Intermediate_Attribution_temperature =  P({temperature}) - P({}) = 2 - 3 = -1
```

Step 2:

```
P({temperature}) → P({temperature, cloud_cover})
Intermediate_Attribution_cloud_cover = P({temperature, cloud_cover}) - P({temperature}) = 6 - 2 = 4
```

Step 3:

```
 P({temperature, cloud_cover}) → P({cloud_cover, temperature, humidity})
Intermediate_Attribution_humidity = P({cloud_cover, temperature, humidity}) - P({temperature, cloud_cover})
= 15 - 6 = 9
```

Why are these intermediate attributions? We have more than one Shapley path, so we must continue to compute the partial attributions across all paths. To obtain the final feature attribution, we take the average of all of the intermediate attributions for that feature.

WHAT ABOUT CLASSIFICATION MODELS?

Although it is not immediately obvious, using Shapley values with classification models is perfectly okay, and not as hard as it seems. This is because almost all modern classification models represent classes not as a set of labels, but in some vector form (for example, either through embeddings, or one-hot encoding). In this case, our mental representation of how Shapley values push an inference towards or away from the final prediction value may not be as accurate. Rather than thinking of the Shapley value as pushing towards one class and away from another, envision a Shapley value as strengthening or weakening the predicted score for that individual class. For a multi-class classification model, the Shapley values are doing this strengthening and weakening for each class. A typical way to calculate Shapley values for multi-class classification is to independently calculate the Shapley values for each possible class, rather than just the predicted class.

Sampled Shapley technique

In a world with infinite time, you would calculate intermediate attributions for every possible path, representing every possible coalition of features. However, this quickly becomes computationally infeasible–for a dataset with ten features, we would need to get predictions for 2^{10} combinations[5] (or 1,023 additional predictions!). So almost every feature attribution technique you encounter that relies on Shapley values will use an optimized method known as sampled Shapley.

The idea behind sampled Shapley is that you can approximate these paths by either skipping steps in the path for some features where there does not appear to be a large contribution, sampling random coalitions and averaging the results using Monte Carlo methods, or following gradients. The tradeoff to using sampling is that we now have an *approximation* of the true attribution values, with an associated *approximation error*. Due to this approximation error, our Shapley values may not sum up to the predicted result. There is no universal way to calculate the approximation error, or “good” range from the approximation error, but generally you will want to try tuning the number of sampled paths to get the best tradeoff between computation performance and error; a higher number of sampled paths will decrease the error but increase runtime, and vice versa.

Baselines

In our examples of Shapley paths, we repeatedly referred to model prediction values using only a subset of the features our model was built for (e.g., `P{temperature, cloud_cover}` when our model took `temperature`, `cloud_cover`, and also `humidity` as inputs). There are several ways to compute these “partial” predictions, but for the purposes of this book we will focus on Shapley techniques which use Baselines.[6]

The main concept behind baselines is that if one can find a neutral, or *uninformative*, value for a feature, that value will not influence the prediction and therefore not contribute to the Shapley value. We can then use these uninformative values as placeholders in our model input when calculating the Shapley value for different feature coalitions.

In our rainfall example, our partial prediction of `P({cloud_cover = 0.8, humidity = 0.9}) = 10` may actually be fed to the model with a baseline value for temperature of `temperature = 22` (Celsius), or `P({cloud_cover = 0.8, humidity = 0.9, temperature = 22}) = 10`

Baselines can vary, there may be an uninformative baseline which is best for different groups of predictions, or a carefully crafted baseline which is uninformative across your entire training dataset. We discuss the best way to craft baselines in chapters 3-6, which are focused on explanations for different modalities.

WHICH WAY TO CALCULATE SHAPLEY VALUES?

Calculating Shapley Values as originally intended (by entirely removing features and observing the result) is rarely feasible in ML because most datasets are historical- it is not feasible to “rerun” or recreate the environment of the dataset without a certain feature, or combination of features, and observe what a new predicted value would be. Due to this, a variety of techniques have been invented over time to overcome these limitations by addressing how to treat the removed features and combinations in intermediate steps of path. Many involve creating a synthetic value for the removed feature(s) that allows the model to be used as-is with minimal contribution from the removed features, rather than trying to alter the model architecture or final predicted value, and often assuming that the ordering of features in the intermediate path steps does not matter. For example, the open source SHAP library has four different implementations for efficiently calculating Shapley values based on the model architecture. For a deeper discussion of the different Shapley value techniques, see the very well written The Many Shapley Values for Model Explanation by Sundararajan and Najmi.

You will often see Shapley values referenced as a method for explanations, or forming the basis behind others. As they have seen many decades of use across a variety of fields, and been very well studied in academia, they represent one of the most proven techniques for explainability. However, as we saw in discussing sampled Shapley, true Shapley values are computationally infeasible for most datasets, so there are tradeoffs to this technique due to the lack of precision. Likewise, although superficially it is easier to understand feature attributions based on Shapley values, it can be quite difficult to explain the game theory concepts to stakeholders to build trust in use of these techniques.

Gradient-based techniques

Gradient based approaches towards explainability are some of the most powerful and commonly used techniques in the field. Deep learning models are differentiable by construction and the derivative of a function contains important information about the local behavior of the function. Furthermore, since gradients are needed to fuel the gradient descent process that most machine learning models are trained upon, the tools of autodifferentiation for computing gradients are robust and well established in computing libraries.

A gradient is a high dimensional analog of the derivative of a function in one variable. Just as the derivative measures the rate of change of a function at a point, the gradient of a real-valued function is a vector indicating the direction of the steepest ascent. Gradient descent relies on the gradient in the parameter space to find the parameters that minimize the loss. Similarly, measuring the gradient of a model function with respect to its inputs gives valuable information as to how the model predictions may change if the inputs change as well. This in turn, is very useful for explainability. The gradient contains exactly the information we need to say “if the input changed this much in this way, the model’s prediction would change as well.” Many of the explainability techniques we discuss in this book are based on evaluating how the value of predictions change as the values of features are changing as well.

Gradient based methods are particularly common for image based models (see Chapter 4) and they provide an intuitive picture of how these methods are typically applied. Given some example input image, to measure the gradient of how the model arrives at its prediction for that example, the image is varied along a path in feature space from some baseline to the values of the original input image.

But what exactly is a path in feature space? For images, we can think of a picture that is 32 x 32 pixels, like the images in the CIFAR-10 dataset, with 3 channels for RGB values as a vector in 3,072 dimensional space. Modifying the elements of that vector modifies the picture it represents. For gradient based techniques, you start with a baseline image (similar in spirit to the role of the baseline in computing Shapley values) and construct a path in that 3,072 dimensional space that connects the baseline with the vector representing the input image.

Gradient based techniques for images make a lot of sense because they take advantage of the intrinsic property of an image, namely that it represents an array of uniform features with a fixed, linear scale (e.g., from 0.0 to 1.0 or 0 to 255), This allows these techniques to rapidly evaluate multiple versions, or steps, of an image as they move along the gradient. The technique will then combine its observations from these steps to calculate an attribution value for each pixel[7], and in some cases, segment these attributions into regions. Gradient based techniques are also used for tabular (Chapter 3) and text (Chapter 5) models but the type of baseline you use for those cases will vary.

If this technique of determining attribution values sounds similar to the discussion of Shapley Values in this chapter, and Sampled Shapley, which is covered in-depth in Chapter 3, that’s because it is! Both integrated gradients and Shapley values are techniques that measure the individual influence of individual features in the model and they do that by measuring how the model’s prediction changes as new information of the input example is introduced. For Shapley values this is done combinatorially by adding features individually or in coalitions to determine which features or coalitions have the strongest influence in comparison to a baseline prediction. However, with images, this approach is a bit naive as it rarely makes sense to entirely remove “features” (or substitute with a baseline value) given that pixels location in the image is relevant. Furthermore, the pixel’s value relative to its neighbors also gives us important information, such as whether it constitutes the edge of an object. The idea of “dropping out” entire regions of an image to understand pixel influence has not yet been well explored, and given the computational complexity of exhaustively generating all (or random) regions, it is likely this type of technique will be more used in interpretable models in the future, rather than as a stand-alone, model-agnostic explainability technique.

Saliency maps and feature attributions

Saliency maps arise in various contexts in machine learning but are probably most familiar in computer vision. Saliency maps, broadly, refer to any technique that aims to determine particular regions or pixels of an image that are somehow more important than others. For example, saliency maps can be used to highlight the regions in an image to better understand where and how a human first focuses their attention by tracking eye movements. The MIT/Tuebingen Saliency dataset is a benchmark dataset for eye movement tasks. The saliency maps in this dataset indicate important regions in an image from tracking human eye movements and areas of fixation, see Figure 4-1.

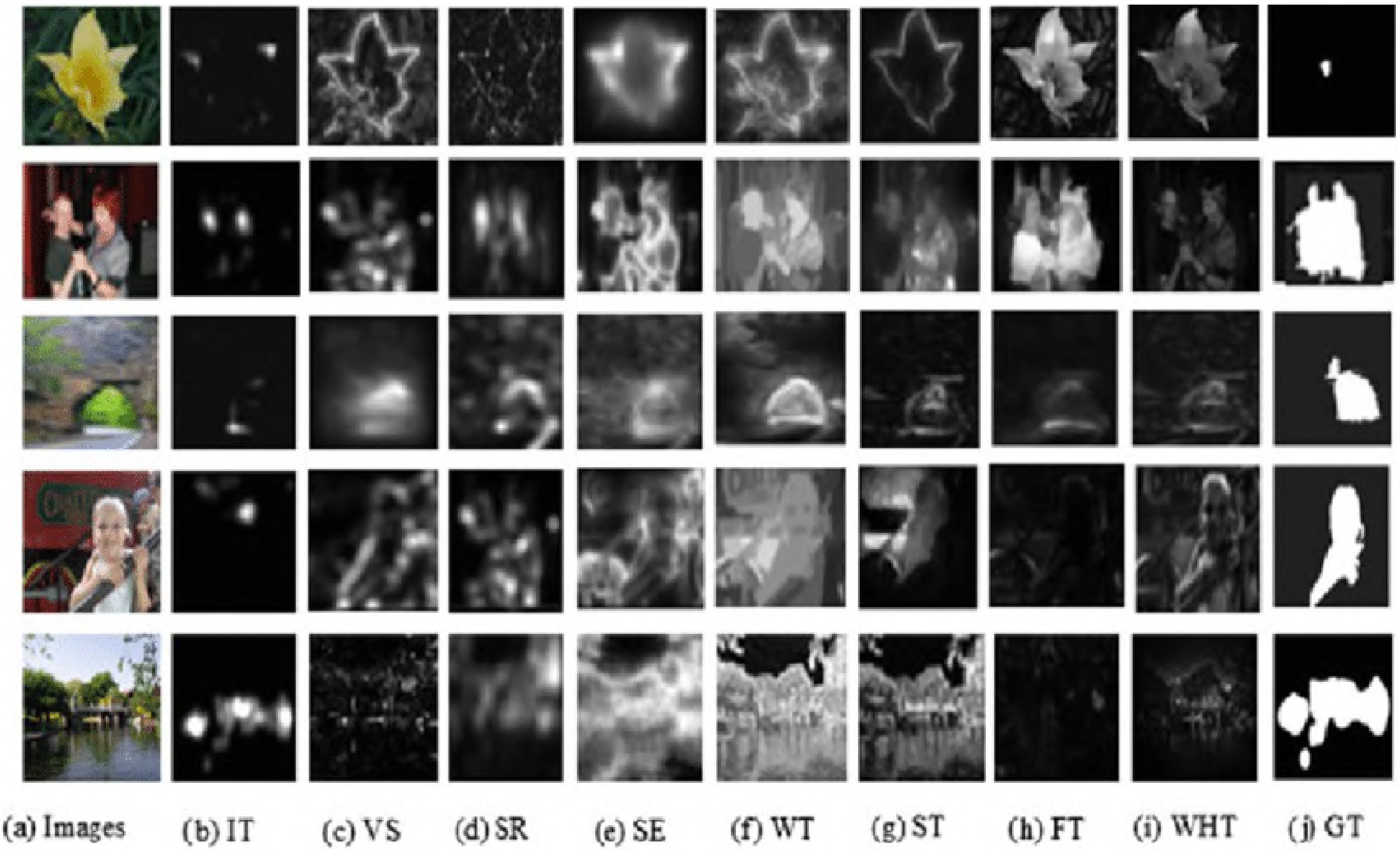

Figure 1-2. Examples from the MIT/Tuebingen Saliency dataset

In the context of explainability and feature attributions, saliency methods serve a similar purpose; that is, they emphasize the regions or pixels of an image that indicate where a trained model focuses its attention to arrive at a given prediction. For example, in Chapter 4 we'll see how the method of integrated gradients can be used to produce a mask or overlay highlighting the pixels that contributed most to the model's prediction. We'll see precisely how the mechanics of integrated gradients and other gradient based techniques lead to this kind of attribution mask as well as a host of other saliency-based explainability techniques that can be applied to image models.

Surrogate Models

Another form of explaining a model's behavior is to use a simplified version of the model, the *surrogate*, to give more explanatory power directly from observing the architecture of the model (also known as *model distillation*). In this sense, surrogate models represent a halfway point between post-hoc explainability and intrinsic interpretable models. While this may sound like the best of both worlds, the tradeoff is that a surrogate model almost always has worse performance than the original model, and usually cannot guarantee that all predictions are accurately explained, particularly in edge cases or areas of the dataset that were under-represented during training.

Surrogate models usually have a linear, decision tree, or rule-based architecture. Where possible, we try to highlight the ability to use a surrogate model, but the field of automated model distillation still resides more in research labs than industry. What this usually means is that you must build your own techniques, or adapt proof-of-concepts, to make your surrogate models.

Activation

Rather than explaining model behavior by which features influenced the model's prediction, activation methods provide insight into what parts of the model's architecture influenced the prediction. In a DNN, this may be the layers which were most pivotal in the model's classification, or individual neuron's contribution to the final output. Going even further, some techniques seek to explain through *concepts* learned by the model, driven by what was activated within the architecture for a given input.

Likewise, during training, an individual data point may be active in only certain circumstances, and strongly contribute to a certain set of labels, or range of predicted values. This is typically referred to as *training influence*, but is analogous to activations within the model architecture.

Activation methods are among the most recently proposed explainability techniques in machine learning and provide an intriguing approach to leveraging the internal state of a complex model to better understand the model behavior. However, these techniques haven't yet been widely adopted among practitioners in the community and so you will see less applications of them in this book although Concept Activation Vectors are discussed in Chapter 6.

Putting It All Together

While we may think of explanations as being primarily about their utility, meaning, and accuracy, understanding the ways in which explanations can vary is the first step in choosing the right tool for the job. By understanding what type of explanation will be most useful to your audience, you can go straight to using the best technique. Choosing an explanation method by simply trying many different techniques until an explanation looks good enough is closer to asking ten strangers you find on the street for investment advice. You may get a lot of interesting opinions, but it is unlikely most of them will be valid.

Start with asking yourself about who is receiving the explanation, what is it that needs to be explained, and what will happen after the explanation? For example, an end-user receiving an explanation about being denied a loan will not find a global explanation focused on understanding the model's architecture to be relevant or actionable. Better to use a technique that is local and focused on the features in order to center the explanation on factors in the user's control. You may even add a technique that provides a counter-factual explanation; for example, that the loan would likely have been approved if their credit score and finances were more similar to other consumers who had requested the same loan amount.

Business stakeholders, on the other hand, want to see the big picture. What types of features are globally influential in this model? Why should this more complex, opaque model be trusted over the previous linear model that's been in use for years? A global feature attribution technique that is post-hoc and model-agnostic could be used to compare how both models behave.

Summary

In this chapter we gave a high level overview of the main ideas you are likely to consider as a practitioner developing explainable ML solutions. We started by discussing what explanations and how an explanation may change depending on the audience (e.g. ML Engineer vs Business Stakeholders vs Users). Each of these groups have distinct needs and thus will interact with explanations in their own way.

We then discussed the different types of common explainability techniques, providing a simple taxonomy that we can use to frame the methods we will discuss in the later chapters of the book. Lastly we covered some of the recurring themes that arise throughout explainability, like the idea of feature attribution, gradient based techniques, saliency maps and more recent developments like surrogate models and activation maps.

In the following chapters, we will dive into explainability for different types of data, starting with tabular datasets in Chapter 3. As you will see in Chapter 3, all of the background and terminology in this chapter is immediately put into practice now that we have given you the knowledge to understand and distinguish between different types of techniques.

[1] Ironically, the use of neurons and neural nets was inspired by trying to explain how our brain's vision system was able to accomplish seemingly impossible tasks, like performing edge detection and pattern classification before an object was classified by our consciousness mind.

[2] Following Google developer guidelines, we avoid the use of the phrase "black box" in this book. See https://developers.google.com/style/word-list#black-box for more details.

[3] Also, P() is not a probability function, so you may encounter combinations of inputs that violate assumptions about probability function notation.

[4] For larger coalition sizes, the average is weighed by the number of coalitions of that size. See the formal definition of Shapley values for details.

[5] The actual number of predictions could be even higher if you did not optimize and save the results of predictions to be reused between different paths.

[6] Other options include Conditional Expectations, and RBShap. /cite{Sundararajan, Najmi, 2019} provide an excellent breakdown of these different approaches, including tradeoffs and real-world examples.

[7] As we'll explore further in the chapter, the exact way of calculating how to combine these steps is a key differentiating factor between techniques.