CHAPTER 6

Advanced and Emerging Topics

with Sheeraz Ahmad

The focus of the book so far has been on well-established techniques, modalities, and use cases. However, Explainable AI (XAI) continues to be an active area of research, so that new techniques are continually being developed and existing techniques are improved and scrutinized further. Feature-based explanations such as Shapley values and Integrated Gradients introduced in the previous chapters can cover many use cases, especially as applied to text, tabular, and image data. However, there are several emerging techniques and topics that can be valuable in your explainability toolbox in specific situations.

In this chapter, we will discuss three broad, emerging topics. First, we will introduce alternative explanation techniques like attribution to inputs (as opposed to features) and making models explainable by design. Second, we will briefly cover how some of the previously introduced techniques can be more generally applied to data formats that are not text, tabular, or image, specifically focusing on time-series and multimodal data (text + image). Third, we will discuss how explainability techniques can be evaluated in a systematic way, as opposed to spot checks on a handful of data points.

Alternative Explainability Techniques

In this section, we will discuss two alternative explainability methods, namely, alternate input attribution, which is attribution to training data points or user-defined concepts, as well as explainability by design, which involves intervening in the modeling process to make it inherently more explainable.

Alternate Input Attribution

Although reasoning based on an example's features is a sensible way to approach explainability, predictions can also be attributed to other inputs such as other examples in the training data or examples in some supplementary data. Note that feature attributions are also indirectly affected by the training data since after all, training data is what the model uses to learn about a task, and feature attribution techniques require querying the trained model in various ways. However, this indirect effect (training data → model → feature attribution) is hard to trace, making it tricky for us to answer the question of which points in the training data a correct (or erroneous) prediction should be attributed to.

We will now discuss techniques that allow us to directly attribute credit (or blame) to individual data points in the training set, or to data in a supplementary set that can be curated by a domain expert. We'll discuss three broad types of alternate input attribution methods: example-based explanations, influence functions, and concept-based explanations. Example-based explanations provide insight into model behavior by surfacing elements of the training dataset that the model treats as similar (or different). Influence function-based explanations also utilize examples from the dataset but focus on those examples that significantly affect model behavior. Here, the influence of a training example is measured by how much the model parameters or predictions change if that example were to be removed from the training dataset. Lastly, concept-based explanations use the internal state of the model to compare how high-level, abstract concepts compare with input instances and model predictions. The advantage is that these concepts align better with human intuition than individual features might.

Example-based explanations

Here's what you need to know about example-based explanations:

- These provide insight into model behavior by surfacing approximate nearest neighbor-based explanations for model instances.
- They work equally well with different data modalities—image, text, and tabular.
- Example-based explanations are primarily considered model agnostic and provide explanations based on elements of the datasets, not model features.
- Counterfactual explanations are a form of example-based explanations.

Pros	Cons
• These explanations are useful for debugging models as well as closing the loop with stakeholders. • They are a very intuitive, human-understandable representation of model behavior. • Example-based explanations are helpful in building a mental model for understanding your ML model predictions and give additional insight into complex data distributions, surfacing hidden data issues or outliers. • It's quick to get started with generating example-based explanations using open source libraries like ScaNN.	• Example-based attribution does not offer guarantees in completeness—summing up the distances of each training data point from a given test point doesn't sum up to anything meaningful. • These explanations don't offer any insights into the causal relationship between the test point and the corresponding example-based explanations. • It can be difficult to scale up example-based explanations beyond ~1–10K examples locally; beyond that, you'd want to use a cloud service.

Example-based explanations are an intuitive way to communicate a model's reasoning. Akin to feature attribution where the objective is to assign partial responsibility of the prediction to individual features, the objective for example-based explanations is to do the same to individual points in the training data. For instance, the question of why the leftmost image in Figure 6-1 was classified as a husky, a breed of dog, can be answered in different ways. We can point to the contributing features, in this case pixels (using a technique like Integrated Gradients) or regions (using a technique like XRAI). Alternatively, we can point to other similar examples in the training data that were huskies, as potential explanations.

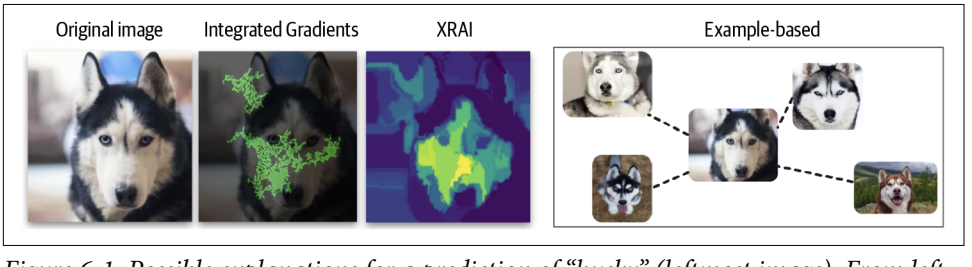

Figure 6-1. Possible explanations for a prediction of "husky" (leftmost image). From left to right: Integrated Gradients, XRAI, and example-based explanations.

Example-based explanations work by first transforming any given input into a meaningful representation using the learned internal state of the model, and then finding the nearest neighbors in the training data for a query point, based on a predefined notion of distance such as Euclidean or cosine distance. For deep learning models, such representations are often called embeddings and are expected to capture semantics of the data distribution for a well-trained model.

The term *embeddings* is used across various domains to mean similar things. In essence, it is a vector representation (i.e., an element of the latent space) of the data often learned by a model as a by-product of performing its intended task. For example, in case of text, we can think of it as capturing semantic properties of the word like polarity (positiveness), gender, level of formality, etc., each by one element of this embedding vector and from a range of 0 to 1. A model might have learned this vector as it tried to perform sentiment classification. Similarly for images, each embedding vector element might capture saturation, humanness, soft versus sharp contours, etc., which a model might have learned as it tried to perform pedestrian detection. Of course, these are made up of examples, and we can't be sure a specific aspect is clearly captured by an element of the embedding. In general, they seem to be useful and often appear to capture meaningful aspects of the data.

With the assumption that meaningful embeddings have been learned by the model, example-based explanations return results often aligning with user intuition. Such explanations can then be used for applications such as *misprediction analysis*—if the example-based explanations for a mispredicted test point have labels that agree with the predicted label and disagree with the true label, it might point to issues with data quality. Figure 6-2 illustrates one such case where the explanations for a misclassified bird are largely from the airplane class, and subsequent manual inspection reveals the uniqueness of this bird image and how it indeed looks similar to an airplane. This in turn points to a potential data sparsity for silhouetted birds.

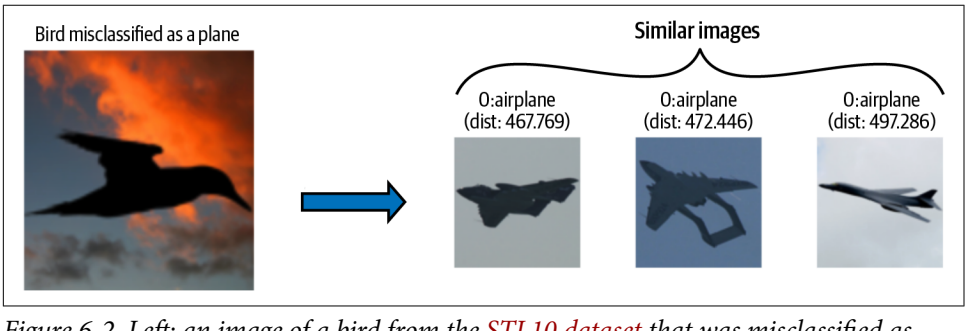

Figure 6-2. Left: an image of a bird from the STL10 dataset that was misclassified as a plane by a fine-tuned MobileNet model. Right: example-based explanations (using the penultimate layer output as embeddings and Euclidean distance) that in fact look similar to the query image but come from the airplane class.

We saw how example-based explanations can be useful in some specific applications, and to help you figure out whether they can be a good fit for your needs, let us look at their pros and cons.

One of the primary advantages of example-based explanations is that they provide very intuitive, human-understandable representation of model behavior. This makes this method particularly useful for debugging models as well as closing the loop with stakeholders. By surfacing related examples from the training dataset, example-based explanations are helpful in building a mental model for understanding your ML model predictions and give additional insight into complex data distributions that can be useful in surfacing hidden data issues or outliers. Example-based applications are uniquely useful for identifying issues with training data and pinpointing when more data might need to be collected. In addition, it's quick to get started[1] with generating example-based explanations thanks to the availability of open source libraries like ScaNN.

On the other hand, unlike many feature attribution techniques, this simple example-based attribution does not offer guarantees in completeness—if we compute distances of each training data point from a given test point, they don't sum up to anything meaningful. This means that it does not point to or quantify a specific property of an input or image. Instead, a data scientist has to fill the gap and determine what the example-based attribution reveals, such as in Figure 6-2. In explanations that have a completeness property, one can quantify how much something contributes to a decision without much need to fill in the interpretability gap.

Another issue to keep in mind is scalability: it can be difficult to scale up example-based explanations beyond ~1,000–10,000 examples using OSS solutions and local processing power, and you will likely need to use a cloud service if your example set is larger than that. Furthermore, example-based explanations don't offer any insights into the causal relationship between the test point and the corresponding example-based explanations—how would the prediction of the test point change if one of the explanation points was removed from the dataset? Intuitively, if the dataset is rich enough, removing one of the explanation points for any test point would have negligible effect on the model prediction. This goes somewhat against the idea from feature attribution where high attribution generally implies high importance to the prediction. Relatedly, a nonsimilar example can still be useful since it can impart more discriminative power to the model.

One approach that has been proposed to address this issue and better consider the perspective of individual data points is influence function-based explanations. We discuss this technique in the next section.

1 Have a look at this ScaNN demo using the GloVe dataset.

Influence function-based explanations

Here's what you need to know about influence function-based explanations:

- Influence functions describe model behavior through the lens of the training data and influential examples. They are used to understand the relative importance of different points in your training data.
- An influential training example is an example such that removing it from the training data would cause considerable changes to the model parameters or predictions. Influence function-based explanations measure how model predictions would change if that example was removed.
- Computing the influence of specific training data points requires computing second-order derivatives. The influence can be positive or negative, which indicates whether a particular data point helped or hurt the prediction.

Pros	Cons
• This is a flexible technique that can be applied to debug models, detect dataset errors, and even create visually indistinguishable adversarial examples. • The explanations they offer align better with intuition—how important is a training data point to the model's prediction. • They work well for small or moderately sized models.	• They don't scale well to large models or large datasets. • Influence functions lack a systematic way to account for correlated data points. The influence of a given data point might appear low if there's another data point that is strongly correlated with it. • These explanations require the model to be twice differentiable.

Using influence functions[2] (IFs) to explain model predictions was proposed in a 2017 paper by Koh et al. and was inspired by a classical technique in the field of robust statistics from 1974.[3] In the context of explainability, IFs provide an alternate way to assign attribution to training data points by asking the question of how would the loss on a specific test point x_{test} change if a given training data point was removed from the training set. That is, we want to measure:

$$L\left(x_{test}; \theta_{X \setminus x}\right) - L(x_{test}; \theta_X)$$

where X is the full dataset, $X \setminus x$ represents the dataset with a single example x removed, and θ_X denotes the model parameters learned using X (similarly for $\theta_{X \setminus x}$).

2 Pang Wei Koh and Percy Liang, "Understanding Black-Box Predictions via Influence Functions," International Conference on Machine Learning, PMLR, 2017.

3 F. R. Hampel, "The Influence Curve and Its Role in Robust Estimation," *Journal of the American Statistical Association* 69, no. 346 (June 1974): 383–93.

While this may appear to be a sensible approach, a naive implementation would be prohibitively costly, requiring us to retrain the model with different subsets of training data. This practical hurdle should sound familiar—an exact implementation of Shapley values runs into a similar issue requiring rebuilding and retraining the model with different subsets of input features.

To address this scalability problem, approximations are needed. We start by considering how the model parameters would change if we upweighted a single training example $z = (x,y)$ by a tiny, tiny bit called ϵ. Of course, upweighting by a tiny, tiny amount is the same as removing that example completely. If we let $L(z_i,\theta)$ denote the loss function at the point z_i with the parameters θ then the new model parameters, with this example z upweighted by ϵ, would be:

$$\widehat{\theta}_{z,\epsilon} = \arg\min_{\theta} \frac{1}{n}\sum_{i=1}^{n} L(z_i,\theta) - \epsilon\, L(z,\theta)$$

The idea of influence functions is to approximate what the change in the loss for the parameters learned on the full dataset θ_X versus the loss for the parameters learned on the full dataset minus one test example $\theta_{X \setminus x}$. So, the influence on the loss of upweighting and element x at a test point x_{test} can be approximated (by chain rule, omitting the derivation) by:

$$\text{Influence}(x,x_{test}) = \left.\frac{dL\left(x_{test},\widehat{\theta}_{x_{test},\epsilon}\right)}{d\epsilon}\right|_{\epsilon = 0} \approx \frac{1}{n}\nabla_{\theta}L(x_{test};\theta_X)^{T} H_{\theta_X}^{-1}\nabla_{\theta}L(x;\theta_X)$$

where ∇ and H are the first- and second-order derivatives, respectively.

The general idea is that θ_X is a stationary point of the loss function, and thus a solution for $\nabla_{\theta}L(X;\theta) = 0$. This equation can be approximated by a first-order Taylor series expansion giving rise to the Hessian term. Since matrix inversion is still computationally expensive, the authors of the original IF paper proposed further approximations to make the evaluation more efficient. In the subsequent years, somewhat related approaches like representer points[4] and TracIn[5] were published that generalize and expand on the ideas pioneered by IF.

The influence of specific training data points can be positive or negative that indicates whether a particular data point helped or hurt the prediction. Figure 6-3 shows the

4 Chih-Kuan Yeh et al., "Representer Point Selection for Explaining Deep Neural Networks," *Advances in Neural Information Processing Systems* 31 (2018).

5 Garima Pruthi et al., "Estimating Training Data Influence by Tracing Gradient Descent," *Advances in Neural Information Processing Systems* 33 (2020): 19920–30.

helpful data points for two different image classification models, a support-vector machine (SVM) and Inception network. Lower-level features such as contrast and texture seem to have higher influence for SVM, whereas higher-level features such as shape and patterns seem to have higher influence for Inception net. Even an image from a different class (dog) can help the classification for another class (fish) by effectively helping a model learn to differentiate them better.

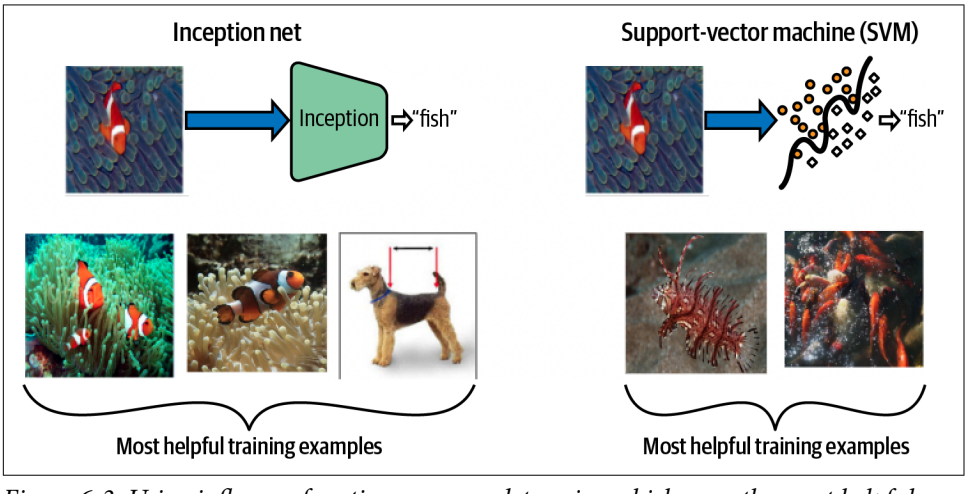

Figure 6-3. Using influence functions, we can determine which were the most helpful training images for each model. For the SVM model, fish close to the test image were mostly helpful, while dogs were harmful for the SVM model. However, for the Inception network, one of the most helpful training images was also a dog, presumably because it helped to determine what a fish doesn't look like as well.

Influence functions have their own set of pros and cons. The explanations they offer align better with intuition—how important is a training data point to the model's prediction. The approximations in the original paper also make it computationally feasible to generate influence functions for moderately sized models and datasets. However, the algorithm doesn't scale well for large models and datasets since it requires computing second-order derivatives and matrix-vector products in high dimensions. Influence functions also lack a systematic way to account for correlated data points, so that the influence of a given data point might appear low if there's another data point that is strongly correlated with it. Even with these caveats, influence functions are a useful tool to understand the relative importance of different points in your training data.

Concept-based explanations

Explanations can also be based on concepts[6]—abstractions that are at a higher level compared to individual features. Here's what you need to know about concept-based explanations, and in particular TCAV:

- One of the commonly used techniques is TCAV (Testing with Concept Activation Vectors). TCAV is a global explainability method.
- Concept activation vectors (CAVs) are a way for machines to represent examples (e.g., images) using the internal layers of a neural net's embedding space.
- TCAV uses CAVs and directional derivatives to quantify how much a user-defined concept is important for the model's prediction.
- The TCAV score is a proportion and always lies between 0 and 1. Values closer to 1 indicate that more of the images of the label class are positively influenced by a concept vector, whereas values closer to 0 indicate less of an influence.

Pros	Cons
• The TCAV score is a proportion, so it's easily interpretable. • Users can determine or explore any concept they define or care about (e.g., gender, textures, patterns); they are not limited to concepts considered during training. • TCAV works without any retraining or modifications to the ML model. • TCAV can interpret entire classes or sets of examples (not just individual data inputs) with a single quantitative measure.	• TCAV can be difficult or expensive to curate a collection of enough examples that illustrate a concept. • TCAV may not perform as well for shallow networks that don't have the capacity to separate internal states as well. • This has been used primarily for images, less so for text or tabular datasets.

For the sake of concreteness in this section, we will focus on concept-based explanations for images. Such explanations can align better with human intuition, which is not attuned to pixel-level perception, as shown in several papers where changes to the pixel values that were imperceptible to humans could be highly significant to a machine learning model. This article by Goodfellow et al.[7] provides an example. Figure 6-4 shows these concept-based explanations for an example image where predictions are attributed to conceptual categories as opposed to Integrated Gradients and XRAI that assign attributions to pixels and regions, respectively.

6 Been Kim et al., "Interpretability Beyond Feature Attribution: Quantitative Testing with Concept Activation Vectors (tcav)," International Conference on Machine Learning, PMLR, 2018.

7 Ian J. Goodfellow et al., "Explaining and Harnessing Adversarial Examples," arXiv, 2014.

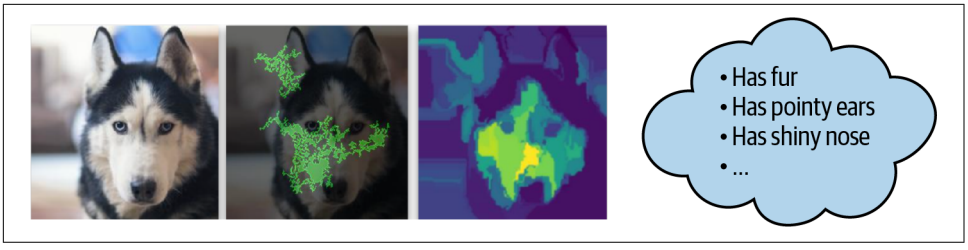

Figure 6-4. Possible explanations for a prediction of "husky" (leftmost image). From left to right: Integrated Gradients, XRAI, and concept-based explanations.

In order to generate concept-based explanations, we need to address two questions. First, how can concepts be provided to the model? And second, how can we measure how such abstractions are learned by the model? We'll discuss how these two questions are addressed with TCAV as an example. For the first question, to implement TCAV, you, the practitioner, provide a set of examples that represent a certain concept or find an independent dataset with that concept already labeled.

The TCAV technique is flexible enough that it can be applied to any user-defined concept (e.g., gender, patterns, textures, or job titles); you just need to curate a collection of examples that exhibit that concept if it doesn't already exist. We recommend roughly 50–200 images per concept and target class for the TCAV algorithm to really be able to pick up on the idea of the concept. That being said, depending on your use case and the complexity of the concept, you can likely get away with only 10–20 pictures, but 200 is generally pretty safe.

For example, in the case of images and animal classification, to measure your model's utilization of the concept of "stripes," as for recognizing zebras, you would provide a set of supplementary images representing the "stripe" concept, and a set of random images representing nonstripes (to facilitate discriminative learning). Or, if you're interested in assessing your model against various fairness metrics, you might collect a database of images representing gender or protected classes to determine how much your model has come to depend on those concepts.

Luckily, for basic concepts there is an easy alternative to creating your own database of images: the Broden (Broadly and Densely Labeled) dataset,[8] which contains images with labels of both low-level concepts such as colors and patterns as well as high-level concepts such as objects. The dataset contains over 1,000 visual concepts ranging across different abstraction levels including scenes, objects, materials, textures, colors, and patterns. This can then be augmented with user-defined concepts that might not be well represented in the existing 1,000 concepts. Figure 6-5 shows some example concepts from the Broden dataset.

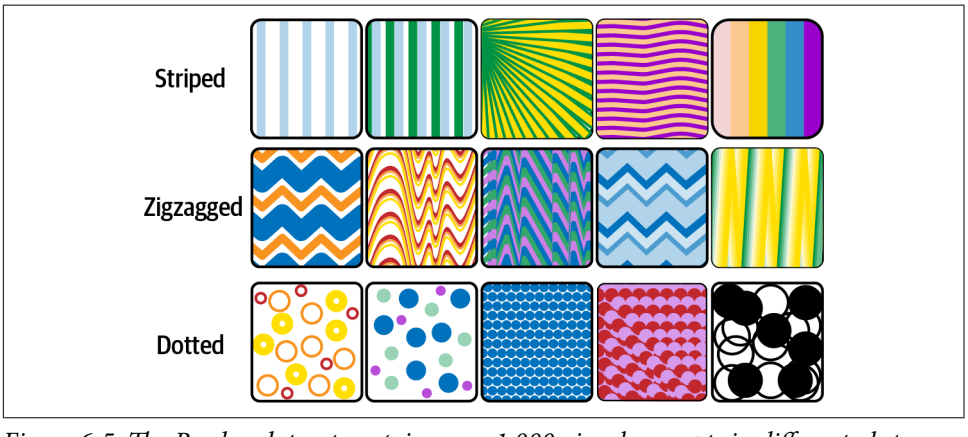

Figure 6-5. The Broden dataset contains over 1,000 visual concepts in different abstraction levels including materials, textures, and colors. Here are some examples of the images that represent the concepts "striped," "zigzagged," and "dotted."

The second question we needed to address is how such abstractions can be learned so that they can be used during attribution. As a potential solution, consider an intermediate embedding layer for a deep learning model, or any other meaningful representation for the broader class of models. We will use the term embedding in the broader sense of a meaningful representation as opposed to the stricter sense of a layer's output. A linear classifier that separates the images representing the stripe concept from the ones in the nonstripe concept is a hyperplane describing the decision boundary between the two. The concept activation vector is defined as the normal to this hyperplane, and in general the images representing the stripe concept will have a larger (signed) projection on this vector compared to the ones representing the nonstripe concept. Figure 6-6 illustrates this process of concept learning.

8 David Bau et al., "Network Dissection: Quantifying Interpretability of Deep Visual Representations," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017.

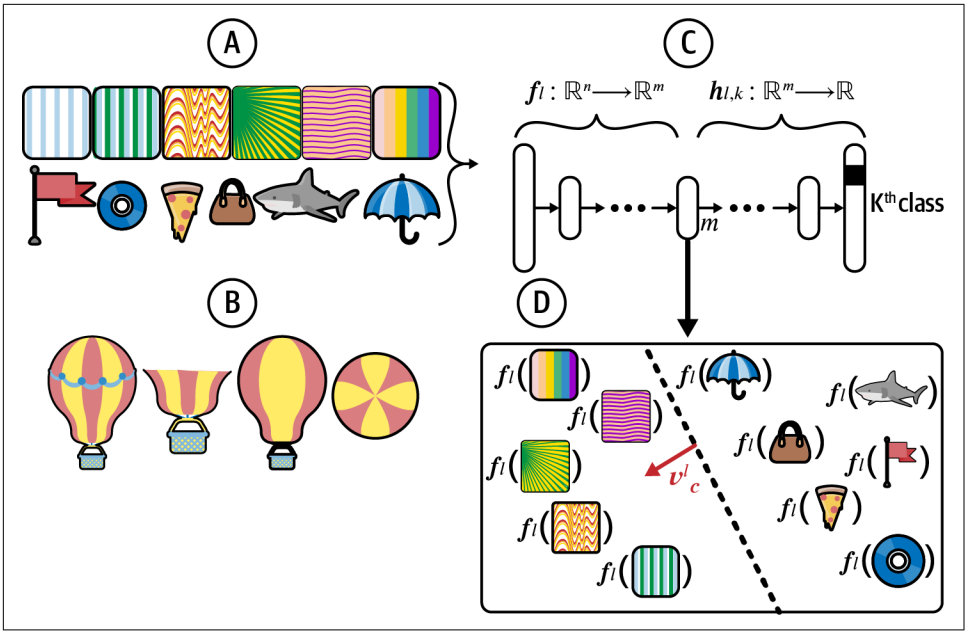

Figure 6-6. Process for learning concept vectors: (a) user-defined set of examples for a concept (e.g., "striped"), and random examples for nonconcept (e.g., "nonstriped"); (b) labeled training data for the specific class (e.g., "hot air balloons"), (c) and a trained network; (d) concept vectors are learned by training a linear classifier to distinguish between the embeddings produced by a concept's examples versus a nonconcept's examples at a given layer. The concept plane is the decision boundary, and the concept activation vector is the vector orthogonal to the decision boundary.

With these prerequisites in place, we can generate global, aggregated explanations that summarize what fraction of images were on the stripe side of the hyperplane. More formally, if we denote by f_l the l-th layer of a neural network and let $h_{l,k}$ logit for the model's predicted class k. That is, $h_{l,k}$ is the part of the neural network that maps the activations of the l-th layer to the k-th class prediction. The directional derivative of that function in the direction of a unit vector CAV v for a concept C is then given by:

$$D_{C,k,l}(x) = \nabla h_{l,k}(f_l(x)) \cdot v_C$$

where x is a specific image input. The dot product measures how aligned two vectors are. If the vectors are orthogonal, then the dot product is zero. The dot product is maximal when the vectors are parallel. So, $D_{C,k,l}$ measures how aligned the gradient and the concept vector are, in essence quantitatively measuring the sensitivity of the model predictions for any layer l and for any concept vector v_C. The TCAV score is

then an aggregation of that sensitivity measure across all images. Letting X_k denote the class of all images with the label k, we want to count the fraction of k-class inputs whose l-layer activation vector was positively influenced by the concept C. Thus, the TCAV score is computed via:

$$\text{TCAV}_{C,k,l} = \frac{|x \in X_k : D_{C,k,l}(x) > 0)|}{|X_k|}$$

Since, by definition, the TCAV score is a fraction, the value will always lie between 0 and 1. When TCAV is closer to 1, more of the images of the k-label class are positively influenced by the concept vector associated with the concept C. When the TCAV score is closer to 0, that concept vector doesn't have as much of an influence.

Note that it's also possible to provide local (individual test point) concept-based explanations, but with the caveat that their quality will depend highly on the user-curated concepts, whereas global aggregation can smooth some of that variability that would occur with a smaller sample size.

There are two main issues with the formulation of concept-based explanations we just discussed. First, for less standard concepts, like those that are not found in the Broden dataset or other similarly curated dataset, the onus of providing meaningful concepts and sufficient examples is on you, the practitioner, and is an error-prone requirement. Second, it's unclear precisely how many concepts are required to sufficiently explain a prediction. In other words, while for some domains only 10 concepts might be needed to explain every prediction, for another it might be 1,000 concepts. To address this second issue, a follow-up work to TCAV[9] also proposes generating supplementary image[10] concepts as patches from the original training data, and learning enough concepts such that a model using only the concept scores (like the distance from the concept hyperplane) can achieve an accuracy as high as the full raw images. However, issues such as redundant concepts, effect of embedding choice, and so on still remain, and concept-based explanation continues to be an active area of research.

Explainability by Design

Another way to approach explainability is by incorporating transparency principles into the model from the get-go. One way this can be done is by choosing models that are inherently explainable, such as linear models or tree models. Using the terminology from Chapter 2, such models are often referred to as interpretable models.

9 Chih-Kuan Yeh et al., "On Completeness-Aware Concept-Based Explanations in Deep Neural Networks," *Advances in Neural Information Processing Systems* 33 (2020): 20554-565.

10 The idea generalizes to other modalities and has been applied to text with reasonable success.

Consider the loan approval models in Figure 6-7, and let's look at an applicant with an income of $80K, a credit score of 750, and a height of 5 ft. 10 in. Note that, here we're purposefully including height as an irrelevant feature and expect that the models will learn to not rely on it. For both the models, the decision is to approve the loans, and there is a clear explanation for how the different features do or do not contribute to the decision: high income and high credit score make the loan approval more likely, whereas height does not affect the decision at all. We can make these observations without the need of a post hoc explanation technique like Shapley values. Of course, more sophisticated models can potentially offer better performance. What follows are alternatives for making models more explainable by design beyond adhering to simple model architectures.

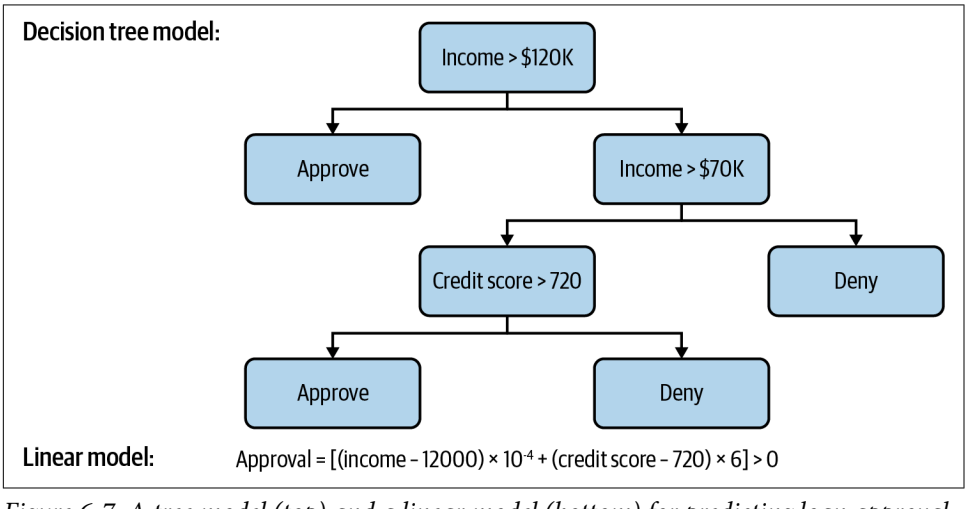

Figure 6-7. A tree model (top) and a linear model (bottom) for predicting loan approval decisions. Both provide inherently transparent decisions, and how the decision will change with respect to the features "income" and "credit score."

Other ways explainability can be made a part of the model building is via adding some explainability constraints during model training or by approximating complex, opaque models by simpler, more transparent ones. Since inherently explainable models form a very limited class of models, we'll explore the more general methods, namely explainability via constraints and distillation, in the following sections.

Explainability via constraints

A natural extension to inherently interpretable models are models that retain some of the desired properties of the interpretable models while offering more flexibility. Monotonicity and linearity are two such properties that we will describe in more detail.

Monotonicity of prediction with certain input features can lead to models that are more interpretable and trustworthy. For example, everything else being the same, a college applicant with higher grades compared to another applicant should have a higher likelihood of getting accepted—which can't be guaranteed by models in general, leading to opaque decisions. Akin to linear models, where a positive coefficient implies nondecreasing and a negative coefficient a nonincreasing relationship, more sophisticated models can be built with monotonicity constraints injected during the design and training phase. In our work on deep lattice networks,[11] we propose model components within the framework of deep neural networks that can retain monotonicity across the layers. Intuitively, monotonicity can be propagated through the layers if the intervening nonlinear layers are monotonic (like ReLU), and the linear transformations have positive weights. Figure 6-8 illustrates this intuition.

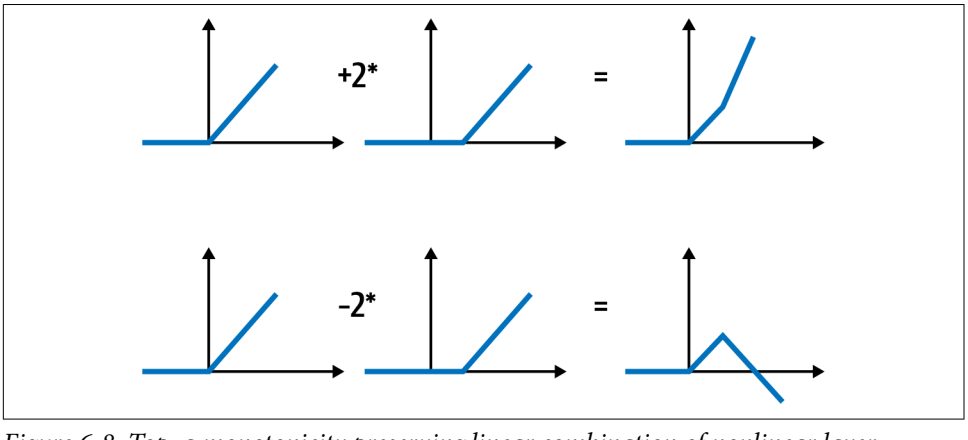

Figure 6-8. Top: a monotonicity preserving linear combination of nonlinear layer outputs. Bottom: without any constraints on the parameters, the monotonicity is not preserved.

Another way to introduce interpretability into the models is by using linear models but in a transformed space that can capture nonlinear interactions. Note that this approach would involve a postprocessing step to use an inverse transform on the transformed features to obtain an interpretation on the original features. Techniques like this have been prevalent and are often realized using kernels[12] where an input is first transformed to a different space and then a linear model is built using these transformed features. Figure 6-9 shows how points that might be linearly

11 Seungil You et al., "Deep Lattice Networks and Partial Monotonic Functions," *Advances in Neural Information Processing Systems* 30 (2017).

12 Bernhard Schölkopf, "The Kernel Trick for Distances," *Advances in Neural Information Processing Systems* 13 (2000).

nonseparable in their original space can become linearly separable in the transformed space, thus allowing you to build powerful yet simple models.

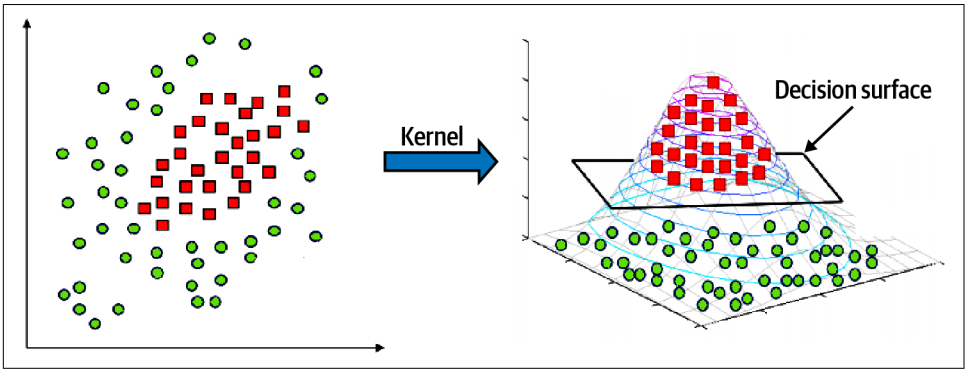

Figure 6-9. Left: the data is nonseparable, requiring more sophisticated yet opaque models for good performance. Right: kernel transformation can make the data linearly separable so an interpretable linear model can be used instead.

Therefore, the main challenge toward enforcing linearity is to come up with kernels or transformations that can make sense to an end user. For example, one such transformation for images might be encoding distance to prototypes. Let's consider a transformation to three prototype-nearness scores (`horse-score`, `crossing-score`, `car-score`); if an image is transformed to (`0.9`, `0.5`, `0.001`), a linear model can predict it to be a zebra, which makes for a transparent and interpretable prediction. Alvarez-Melis and Jaakkola[13] offer a formal generalization to this idea, showing how interpretable models can be built with certain constraints on transformations and composability (more general than linear combination).

Explainability via distillation

Earlier in this chapter, we discussed how simpler models like linear or tree-based models are inherently more explainable. However, they may fare poorly against the more sophisticated models in terms of performance. Model distillation is a framework that attempts to combine the performance of the complex models with the transparency of the simpler models. The central idea involves approximating the performance of a complex, opaque model called the "teacher" with that of a simpler, transparent model called the "student." Along with transparency, this approach has the added bonus of making the models smaller and consequently, faster. There are two main issues to consider here: what is the benefit of this two-tier approach

13 David Alvarez Melis and Tommi Jaakkola, "Towards Robust Interpretability with Self-Explaining Neural Networks," *Advances in Neural Information Processing Systems* 31 (2018), *https://oreil.ly/XxLTs*.

(complex to simple distillation) compared to directly training the simple model, and what are the general ways to build these student models?

Instead of first building a complex model (teacher), and then distilling it down to a simple model (student), a practitioner might consider directly building a simple model if transparency is the goal. While this idea is reasonable on the surface, it misses out on the rich information that the teacher model can provide to the student model. Consider a classification problem—a simple model trained using just the original training data can be thought of as an approximation to the teacher where the goal is only to match the hard labels (cat or dog) since both have access to the same information. However, with the distillation approach, soft labels (`probability(cat)`, `probability(dog)`) can be created using the teacher and provide much more fine-grained information to the student since now the data includes richer differentiation between instances of different dogs (analogously cats).

To elaborate, the general way to build student models is by augmenting the training data using the original, complex teacher model. This can be done via soft labels and predicted hard labels for classification and predicted target for regression. The student can then be built to both optimize performance on the original training set and to optimize how close it is to the teacher. Figure 6-10 shows this general framework where the original data as well as the augmented data is used to train a distilled student model. You can find more details in this foundational paper by Hinton et al.[14] and follow-up work by Frosst et al.[15]

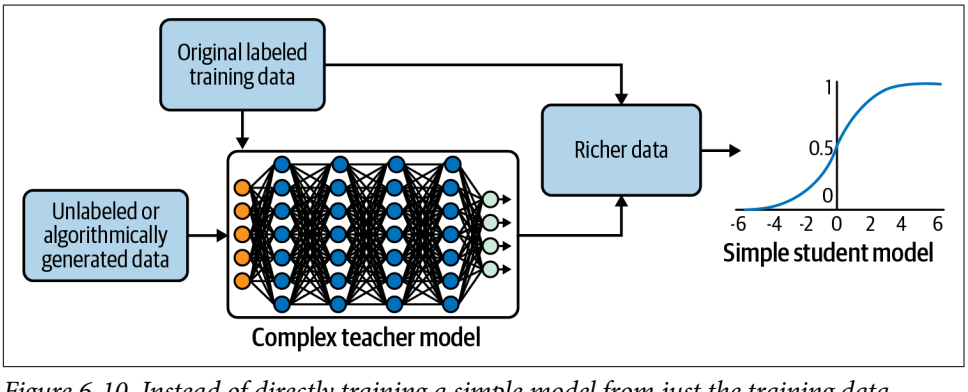

Figure 6-10. Instead of directly training a simple model from just the training data, distillation techniques can use both the training data and the augmented data from a teacher model to train a better-performing student model.

14 Geoffrey Hinton et al., "Distilling the Knowledge in a Neural Network," arXiv, 2015.

15 Nicholas Frosst et al., "Distilling a Neural Network into a Soft Decision Tree," arXiv, 2017.

Other Modalities

While you are more likely to work with datasets containing images, text, and tabular data, the explainability techniques discussed in this book are general and can be applied to other data modalities with a few extra steps. In previous chapters, we went into the details for applications of different explainability techniques to the three common modalities. In this section, we'll go over two case studies that highlight how the different techniques can be applied to other modalities.

Time-Series Data

Time-series forecasting problems are some of the most common ML use cases in the business world. Many real-world datasets have time component features, and accurate time-series forecasting can be incredibly beneficial in decision-making. However, working with time-series data is approached quite differently to how you might approach a typical supervised regression problem. With any ML task, creating features is one of the most important and time-consuming components; with time-series models, it is even more challenging because with time-series data, each feature potentially affects the prediction over different time horizons. A common approach to creating time-series is to use a sliding window where the features for a single training example are taken over a certain number of time steps and used to predict variables of interest for (potentially multiple) future time steps, as shown in Figure 6-11. In addition, these time-dependent features are often combined with static covariates such as holidays, store location, store promotions, or product information. This time dependence on features poses a unique challenge for explainability.

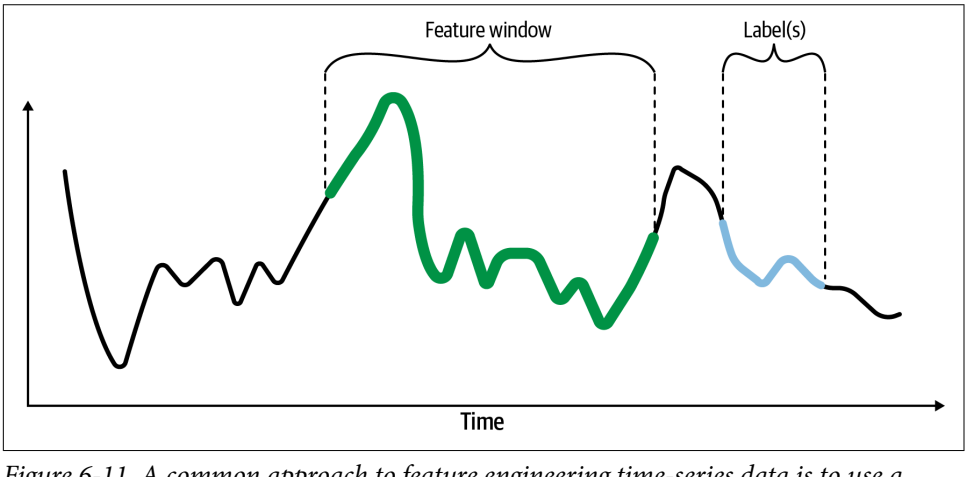

Figure 6-11. A common approach to feature engineering time-series data is to use a sliding window to create features and labels. The feature window would create the features for a single training example and its corresponding label, or labels in the case of multihorizon forecasting.

For example, suppose we wanted to build an ML model to predict customer traffic at a restaurant on any given day. In this case, the number of customers could be used as a time-series feature to capture temporal trends; but also, local ads, the weather, or special holidays are relevant features whose values depend on the time they occurred. In fact, ads and weather on a given day as well as for at least a few days in the past can affect how many people go to the restaurant. Even the future can affect traffic since a gloomy weekend forecast can motivate diners to go out on a Thursday night.

Depending on the model, techniques like Shapley values can still be used because the user can create a baseline time-series and compute how the prediction changes as features are swapped from baseline values to the given value. However, generating feature attributions can be difficult to understand since instead of n, we now have $n \cdot t$ Shapley values, where n is the number of features and t is the time horizon (past + future).

As a concrete example, let us assume we are aiming to predict the customer traffic to the restaurant on a Friday night using only the weather and advertisements and looking at the data for three days into the past, ignoring any information about the future. Now, we have six features that can affect the prediction: weather on Tuesday, Wednesday, and Thursday, as well as whether or not the restaurant was advertised on a local website on the same days. Figure 6-12 shows these features as `[(False, Cloudy), (True, Rainy), (True, Sunny)]`. In this case, the baseline might be `[(False, Cloudy), (False, Cloudy), (False, Cloudy)]`.

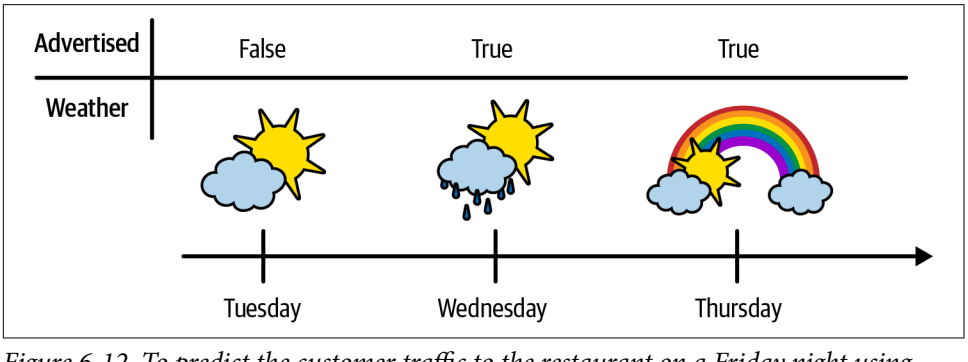

Figure 6-12. To predict the customer traffic to the restaurant on a Friday night using only the weather and advertisements and looking at the data for three days would result in six features to consider when implementing Shapley values.

Using Shapley values as is will give us fine-grained attributions on how each feature contributed on each day. While this information can be useful, it can also quickly get overwhelming and unwieldy for datasets with many features and models with long time horizons. To make sure that the explanations are more easily understandable, some postprocessing can be done. The simplest postprocessing would be aggregating the explanations by features or time.

For example, suppose our model predicts that 65 diners will come to the restaurant on Friday. Aggregating by features accumulates the Shapley values for each feature over the entire time horizon, leading to an overall attribution for an individual feature—weather contributed to 40 diners going to the restaurant whereas ads contributed to 25 diners going to the restaurant. Aggregating by time accumulates the Shapley values for each time point over all the features, leading to an overall attribution for an individual time point—Tuesday's feature information contributed to 5 diners going to the restaurant, Wednesday's feature information contributed to 30 diners, and Thursday's feature information contributed to 30 diners. Both schemes account for the 65 diners that were predicted by the model for Friday, but instead of six attributions (i.e., $3 \cdot 2$), we are down to five (i.e., 3 from aggregating by time + 2 from aggregating by features), as shown in Figure 6-13. If we were considering 20 features over 30 days, the difference would be far more drastic: 600 attributions versus 50.

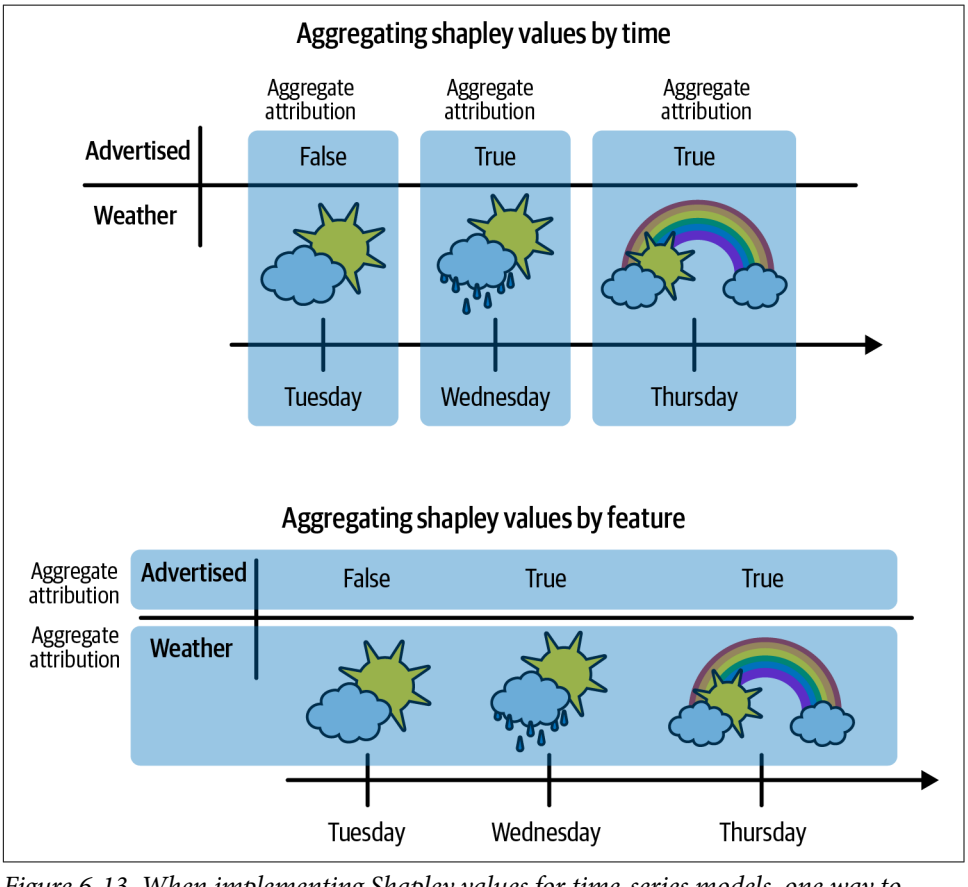

Figure 6-13. When implementing Shapley values for time-series models, one way to make explanations more easily understandable is through postprocessing, e.g., aggregating the explanations by features or by time.

This restaurant example focused on feature attribution, but similar extensions can be made, for example, for influence function-based explanations—how similar is this Thursday to other days in the past, or how would the prediction change if a specific day's data was removed from the training set.

Another approach to explaining time-series data is through attention-based models (discussed in Chapter 5). Also, AI labs out of MIT and the Harvard NLP group have released exBERT,[16] an interactive visualization tool for exploring transformer models. However, these attention-based models are not as well equipped to handle different types of inputs such as both temporal and static features, not just language or speech. To address this challenge, a recent 2021 paper introduced Temporal Fusion Transformers[17] (TFTs). TFTs are an attention-based DNN model designed for multihorizon forecasting and developed specifically to allow for more direct interpretability.

Multimodal Data

There are scenarios where the input can come from many different modalities such as medical diagnosis that can use images (X-ray), text (doctor's notes), tabular (patient information), and time-series (echocardiogram) information. Given the model-agnostic nature of several of the techniques we have discussed in the book, even models built over multimodal data can be explained with the same tools.

Let's consider a relatively simple problem of visual question answering, where given an image and a question, the model is required to predict an answer. Let's also assume we intend to use Integrated Gradients (IG), introduced in Chapter 4, as the explainability technique. We have already seen how IG can be used with images; to use it for text, we first need to transform the text from a discrete (words encoded by their IDs) to a continuous domain (words encoded by their embeddings) because IG requires gradient operations that are ill-defined for discrete domains. Once we have IG attributions for word embedding elements, they can be summed up to get the attribution for the word. The attributions for image pixels do not require any postprocessing. Figure 6-14 shows an example of this type of attribution applied to a visual question answering model with attribution at pixel and word level.[18] This can be a powerful tool for model debugging too—a model that has high attribution for uninformative words like *the* would require further scrutiny.

16 Benjamin Hoover et al., "exBERT: A Visual Analysis Tool to Explore Learned Representations in Transformers Models," arXiv, 2019.

17 Bryan Lim et al., "Temporal Fusion Transformers for Interpretable Multi-Horizon Time Series Forecasting," *International Journal of Forecasting* 37.4 (2021): 1748–64.

18 The code for this example can be found in the Captum tutorial on VQA models: *https://oreil.ly/7nE2e*.

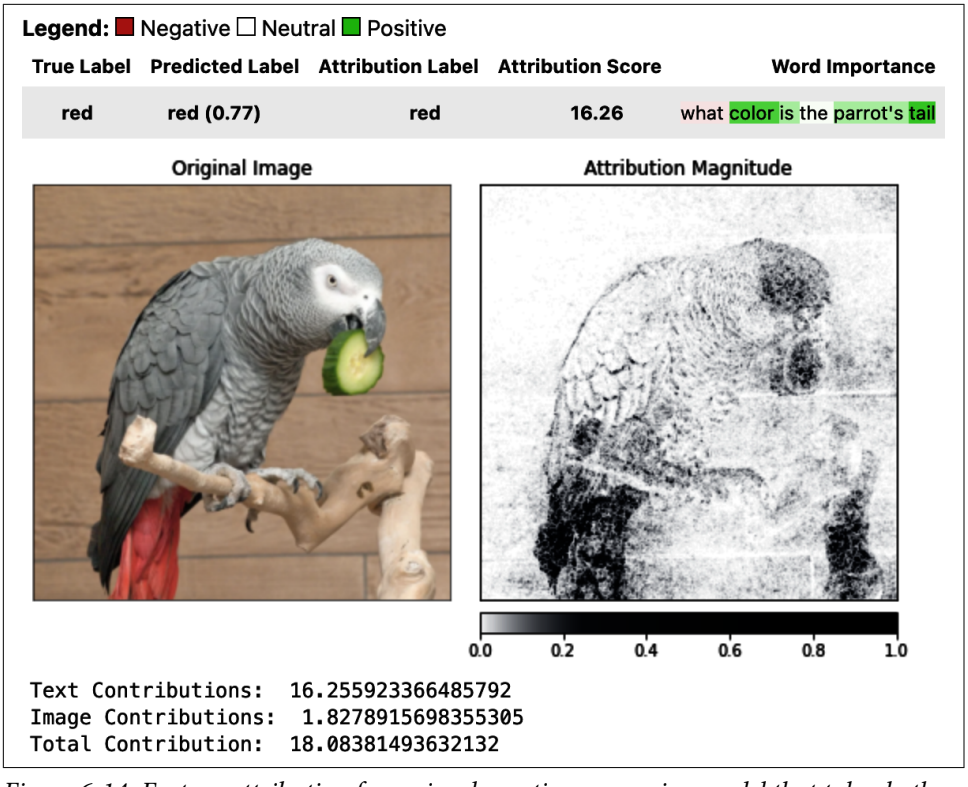

Figure 6-14. Feature attribution for a visual question answering model that takes both an image and a question (text) as inputs. The highlights show which words and which pixels contributed the most to the predicted label. Feature attributions are relative, and aggregation can be done to get the modality level attribution that yields a contribution of 16.26 from the question and 1.83 from the image. (Print readers can see the color image at https://oreil.ly/xai-6-14.)

Evaluation of Explainability Techniques

For most part, this book has focused on well-established explainability techniques that have been widely used across different applications. At this stage, you're probably wondering how these techniques compare with each other or if one technique is better than the others somehow. Unfortunately, there is no free lunch and different techniques may do better or worse depending on your dataset, your use case, and how you plan to use the resulting explanations. Just as there is no one-size-fits-all machine learning model, there is no single explainability technique that surpasses all the rest. Instead, we encourage you to view and collect these techniques as tools in a well-stocked toolkit that exists to help you analyze your model and can be utilized through your entire ML workflow (see Chapter 8).

Although you may have a bunch of explainability techniques at your disposal, the question still remains as to how you can evaluate them and which one is best to use for your use case. Evaluating a predictive machine learning model is much more straightforward; there are well-known metrics you can use like accuracy, precision/recall, mean-square error, intersection over union, and so on. However, there is a lack of consensus for how to evaluate the quality of explanations. Often, researchers have relied on showing a handful of examples to convince others of the usefulness of their proposed explanation technique, but over time more attention has been devoted to developing systematic evaluation methods. In this section, we will go over a few different approaches to evaluating explainability techniques. Even though this discussion is not exhaustive, we aim to provide you with a starting point and to create awareness for the topic.

A Theoretical Approach

With the lack of clear evaluation metrics for assessing explainability techniques, one approach is to take a "first principles" perspective, meaning with no preconceived assumptions, and define a collection of axioms that any practitioner would expect an explainability technique to have.

Axioms of Mathematics

In mathematics, axioms are statements that cannot be proven true or false. They are accepted to be self-evidently true and serve as the starting point from which the rest of the abstract theory can be developed. For example, the axiom of equality states that a number is always equal to itself. Or perhaps, in more layman's terms, "It is what it is." This seems too obvious to not be true, and it's what any reasonable person would expect. However, this statement can't formally be proven true or false, so it is accepted as true. Axioms like this form the foundation of mathematical theory, from which the proofs of all other theorems, propositions, lemmas, and conjectures follow.

An axiomatic approach defines a collection of well-understood and accepted properties (i.e., axioms) that any explainability technique *should* have. This provides an intuitive benchmark with which to judge new or existing techniques. In their 2017 paper[19] introducing the technique of Integrated Gradients, Sundararajan et al. also introduce a collection of fundamental axioms for qualifying feature attributions. We'll briefly discuss those axioms here because they provide a nice sanity check for what

19 Mukund Sundararajan et al., "Axiomatic Attribution for Deep Networks," International Conference on Machine Learning, PMLR, 2017.

we expect an explainability method should possess and, when possible, we'll compare these axioms against some of the techniques we've seen so far in the book. However, keep in mind that these axioms are just one way of framing evaluation, and just because one method doesn't satisfy one of the axioms doesn't mean it should be abandoned completely. It's very likely that it could still be a useful technique for you and for your use case.

To frame the following axioms, we'll borrow the notation from the 2017 paper. Let F represent the model's (e.g., a deep neural network) prediction for a certain class label and the $x = (x_1,x_2,\cdots,x_n)$ represent an input tensor to F. We measure attributions in the context of a baseline, so let x' denote the baseline. For example, suppose we have some input image, as in Figure 6-15, and our model predicts the class label "sulfur-crested cockatoo" with a confidence of 0.886.

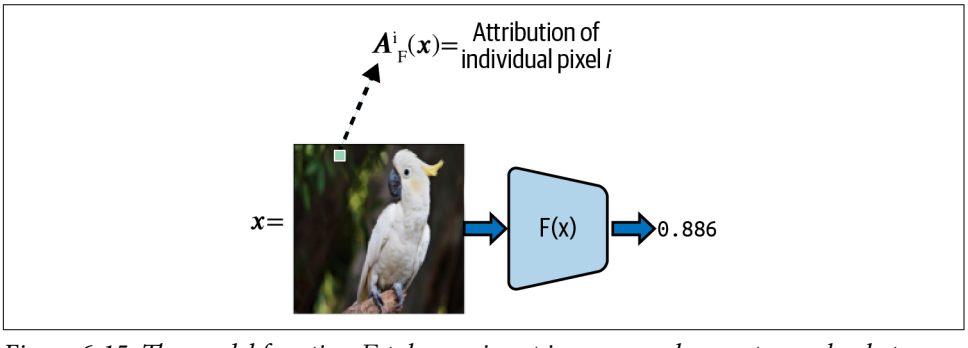

Figure 6-15. The model function F takes an input image x and maps to a value between 0 and 1, which represents the model's class prediction for that example.

The model function F maps the input image x to a probability score in (0,1). If we take the baseline to be a black image, then $F(x') = 0$. The features of the model are the pixels, the individual x_i's of the image, so we take $A^i_F(x)$ to represent the feature attribution of pixel x_i of the model F at the input x with respect to the baseline x'.

Axiom of completeness

The axiom of completeness is perhaps the most straightforward. It states that for any input to the model, the total attribution must be equal to the sum of all of the feature attributions of the input. Mathematically, this means:

$$F(x) - F(x') = \sum_i A^i_F(x)$$

You can think of completeness as a sanity check that the attribution method is comprehensive in assigning attributions to features. That is to say, the attribution of a given input is "completely" accounted for across all model features. It seems like

a reasonable request, and many of the explainability methods we've discussed satisfy this criteria, such as Integrated Gradients, Shapley values, and Layer-Wise Relevance Propagation. However perturbation techniques, such as LIME, do not.

Axiom of sensitivity

As the name suggests, the axiom of sensitivity examines how sensitive or stable an explainability method should be to changing feature values. It states that if the baseline and input differ only by one feature, but the prediction is the same, then that feature should have zero attribution. And conversely, if the input and baseline differ by only one feature, and the prediction is different, then that feature must have nonzero attribution. For example, in Figure 6-16, the two images differ by a single pixel value, but the model prediction doesn't change. The (in)sensitivity axiom says that in this case the attribution for that feature (i.e., pixel) should be zero.

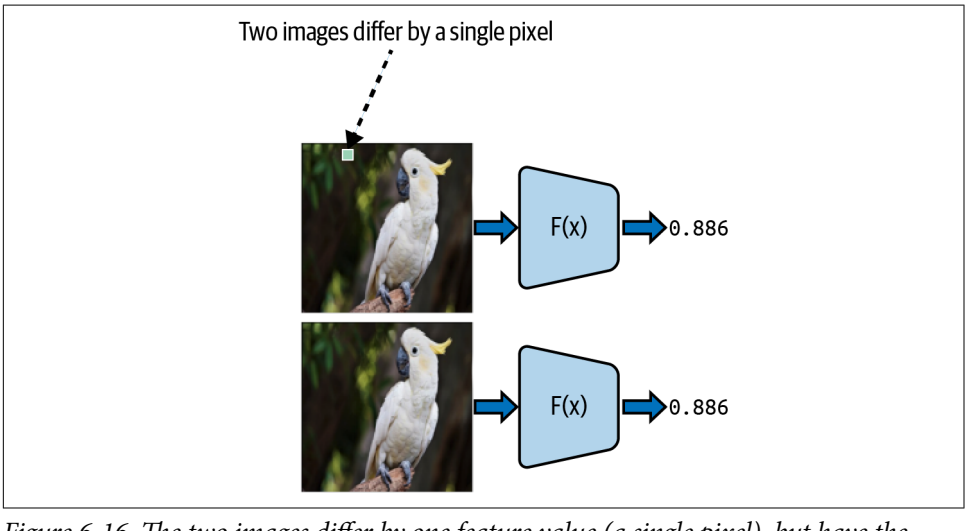

Figure 6-16. The two images differ by one feature value (a single pixel), but have the same prediction. The feature attribution for this pixel should be zero.

Mathematically, this axiom can be formulated as follows: if an input example x has only one nonzero feature and $F(x) \neq 0$ then the attribution for that feature is zero. Stated with respect to insensitivity, this axiom states that if the model function $F(x)$ does not depend on the value of a feature, as in Figure 6-16, then the attribution of that feature should be zero.

You can show that completeness implies sensitivity, so any technique that satisfies the axiom of completeness also satisfies the sensitivity axiom. Although perturbations methods of explainability do not satisfy completeness, they do satisfy sensitivity. However, methods like DeConvNets and Guided Backprop, which we discuss in Chapter 4, violate this sensitivity axiom.

Axiom of implementation invariance

The next axiom is related to the dependence on the machine learning model itself. The axiom of implementation invariance states that if two different models are functionally equivalent, meaning they compute the exact same function $F(x)$, then the attributions for all the features should be the same. In short, if the function F doesn't change between two models, then their attributions shouldn't change either. This axiom seems pretty straightforward to expect of an explainability technique. After all, why should the attributions change just because the implementation changes? So long as the final prediction is the same, that's all that should matter, right? Well, there are indeed methods that don't satisfy this criteria, such as DeepLIFT and Layer-Wise Relevance Propagation.

Axiom of linearity

The axiom of linearity says that if you can express your machine learning model as a linear combination of two other model functions, then the attributions should also be expressed in the same way. For example, if the model function F can be written as the sum of two model functions then you should expect that the attributions sum as well. Integrated Gradients and other path-based methods satisfy this axiom.

Axiom of symmetry-preserving

The last axiom is the axiom of symmetry-preserving. We say that two features are symmetric if they can be interchanged and the model function F prediction doesn't change. The axiom states that if two features are symmetric, meaning they are interchangeable without changing the value of F, then their attributions should be the same as well. This is somewhat related to the sensitivity axiom, but they're slightly different. The axiom of sensitivity says that if you change one feature value and the total attribution of the input doesn't change, then that feature must have zero attribution. Symmetry-preserving instead pertains to the model function itself. It means that there are two input variables that are symmetric. So, for any image, swapping these two pixel values wouldn't change the predicted value of F. In this case, the attributions should be the same as well.

This axiomatic approach to evaluating explainability techniques provides a reasonable sanity check and a good starting point. However, just because a certain technique fails one of these axioms doesn't mean you should discount it completely. Depending on your use case, that technique could still be beneficial to you. They simply provide one lens with which to understand the potential drawbacks or caveats of a certain method. Since these axioms were introduced, there have been a number of other studies aimed at providing a more rigorous framework for evaluating XAI techniques. Next, we'll discuss some of these empirical approaches.

Empirical Approaches

In the previous section, we looked at some of the desirable properties of an explanation technique and stated them as axioms establishing a theoretical framework. Trying to prove that your explanation method satisfies these axioms can get mathematically cumbersome and might be overkill if the aim is to quickly establish whether the method is good enough for your use case. In this section, we look at empirical approaches that can be applied to most techniques out of the box and are founded on intuition about how a technique should behave in simple scenarios.

Basic sanity checks

Basic sanity checks are the first line of investigation for the subject of evaluation. A sanity check is a quick way to weed out poor methods. A good method must necessarily pass such a check, but passing the check is not sufficient proof of correctness or good performance, since the checks are not intended to be exhaustive. With this in mind, one of the simplest questions to ask is whether an explainability technique truly does reveal something about the underlying model behavior. If the explanations don't change much with different models, the quality of the explainability technique appears rightfully suspicious. Similarly, if the data labels were changed, breaking the relationship between the model inputs and their outputs, we should expect explanations to change as well.

A sanity check–based approach[20] has been used to evaluate saliency maps for commonly used explainability techniques, including many of the methods we've discussed in this book like Guided Backprop, Guided Grad-CAM, Grad-CAM, Integrated Gradients, and Gradient x Input. To test the sensitivity of these methods to the model weights, Adebayo et al. compare the output of each saliency method when applied to a trained model against the same saliency output for the model with randomized weights. To test for sensitivity to data, a data randomization test compares the outputs of these saliency methods for a model trained on the true dataset and for a model trained on a copy of the dataset where the labels have been randomly shuffled. As with the model parameter randomization test, you would expect that the saliency outputs would change dramatically when the labels are randomized.

Surprisingly, many of these saliency methods failed these basic sanity checks. For example, the saliency maps produced by Guided Backprop and Guided Grad-CAM were found to be not sensitive to these randomization tests. A follow-up paper[21] discovered some gaps in the evaluation scheme of Adebayo et al., suggesting caveats on

20 Julius Adebayo et al., "Sanity Checks for Saliency Maps," *Advances in Neural Information Processing Systems* 31, 2018.

21 Mukund Sundararajan and Ankur Taly, "A Note About: Local Explanation Methods for Deep Neural Networks Lack Sensitivity to Parameter Values," arXiv, 2018.

how sensitivity should be measured and how visualization of the results can induce bias. These sanity checks, when done properly, can still be of value and provide one means of evaluation for an explainability technique. As a practitioner, you should be cautious when interpreting the results of both the explainability techniques and the evaluation schemes.

Faithfulness check

The next and a more challenging way to evaluate explainability techniques is to consider faithfulness of explanations to the model; that is, whether high attribution (to features or training data points) actually implies importance. In other words, an explainability technique can be considered to be revealing a model's strong reliance on a feature or training data point, if the model's performance suffers strongly when the high attribution inputs (features or training data points) are removed. However, if a technique assigns a high attribution to an input (feature or a training data point), and removing or altering that input doesn't change the model's performance significantly, the technique's attribution is less reliable.

In the paper that introduces XRAI,[22] a masking-based evaluation scheme is defined that captures this idea: starting with a blurry image, most- to least-salient pixels (based on the attribution) are sequentially introduced and model performance is evaluated. For a good explainability technique, we expect sharp improvement in performance at the start since the high saliency pixels are indeed the ones that would be the most useful to the model.

As a concrete example, let's consider a 5×5 image of a cat for which the model predicts "cat" with confidence 0.9. This means there are 25 pixels that can be ranked by attribution. Starting from a blurry image, we add pixels one by one based on their attribution values and notice how the prediction changes. Let's assume that the blurry image has a confidence score of 0 for the image being a cat. For a faithful technique, we would expect a big jump in confidence (say, from 0 to 0.3) when the highest attribution pixel is added, a somewhat smaller jump (e.g., from 0.3 to 0.5) for the next one, an even smaller jump (0.5 to 0.6) for the one after, and so on. This would be an insertion-based check. We can also do a deletion-based check by removing high attribution pixels one by one, and for a good technique, we should see the prediction fall sharply at the start. If the technique was poor, we should see erratic jumps with both the insertion or the deletion checks.

22 Andrei Kapishnikov et al., "XRAI: Better Attributions Through Regions," Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019.

Synthetic datasets

Another approach to evaluation is through synthetic datasets where the ML practitioner knows what the salient inputs are a priori. An explainability technique that can demonstrate it picks out these salient inputs is then more trustworthy than the one that picks out nonsalient inputs. For feature-based explanations, Yang, Mengjiao, and Kim provide a case study[23] on how synthetic data can be used for evaluating explainability techniques. By taking an image patch from a specific class (say, dog) and pasting it in different backgrounds, this approach ensures that the background is never quite salient since an image can be a dog anywhere (dog on a beach, dog in a gym, dog in a ring, etc.) and a good explainability technique shouldn't attribute much relevance to the background. For example-based explanations, similar augmentations can be made by adding random data points to the training set and measuring how attribution gets assigned to these nonsalient data points.

Application specific

Using basic sanity checks, faithfulness checks and checks built using synthetic datasets can help you evaluate your explanation technique in many scenarios. However, as you might have noticed, most of these checks are built as safeguards against obvious mistakes and don't guarantee success for a technique that has passed them. Another prevalent theme of explainability evaluation is centered around real use cases and becomes most relevant for sensitive applications. If explanations are being used to facilitate cancer diagnosis, the evaluation should consider metrics like time saved, mistakes avoided, new mistakes made, etc. A technique can pass all the previous checks, but if it doesn't save doctors any time while diagnosing, it might not be a great fit for the application. Similarly, if the explanations are being used to convey understanding of a model's inner workings to an everyday user, the evaluation should consider how well the explanations align with the user's intuition (this is discussed in more depth in Chapter 7).

Evaluation of explainability techniques is an active and hotly debated topic with a consistent flux of new ideas. As these explainability techniques continue finding their way into critical or regulated applications, the question of how much trust can be placed in any technique will continue to be pertinent. Unlike areas in machine learning where the notion of ground truth is well-defined leading to clear metrics, quantifying the quality of evaluation techniques is further complicated by the fact that for real-world datasets, the ground truth is often ambiguous—is that an image of a bee because of the striped pattern or because of the presence of specific flowers? Both are valid explanations even if one corresponds more closely to human intuition. Until we have well-established evaluation methodologies, it is worthwhile to spend

23 Yang, Mengjiao, and Been Kim, "Benchmarking Attribution Methods with Relative Feature Importance," arXiv, 2019.

some time ensuring alignment between the technique's intended use case and the application at hand.

Summary

In this chapter, we went over some of the emerging topics in the field of explainability. Different ways of approaching the question of explainability continue to receive attention from both researchers and practitioners. With that in mind, we first discussed how attribution or saliency can be assigned to inputs other than just the data features. Specifically, we looked at attributing to training data via a notion of similarity and influence functions. We also revisited attribution to concepts that can be defined via an auxiliary input.

Next we looked at a class of explainability methods that intervene during the modeling process as opposed to the post hoc techniques that are the main area of focus for the book. You saw how constraints like linearity and monotonicity can lead to more transparent and trustworthy models. We also went over how complex models can be distilled to simpler approximate models, often by using richer data generated using the complex model. Such models are easier to understand and have been successfully used for applications like ensuring fairness.

To reinforce the point that several of the techniques discussed in this book are model and modality agnostic, we looked at a couple of examples of applying Shapley values and integrated gradient techniques to different modalities, with the former being mostly applicable out of the box for time-series and the latter applicable to mixed (image and text) modality with low-overhead pre- and postprocessing.

Lastly, we looked at the fairly nascent field of systematic evaluation of explainability techniques. Historically, hand-picked examples of a technique's good performance were taken to be sufficient evidence for the quality of a technique. However, lately, there has been a larger push toward using well-defined, independent evaluation frameworks, especially as explainability continues to be applied to sensitive domains. Ranging from basic sanity checks to faithfulness studies to methods employing synthetic data, these frameworks aim to ensure alignment between the user's expectations and the technique's performance. The goals and methodologies for evaluation are not clearly understood, and extra care is required when using XAI for critical applications. Since a significant amount of XAI usage is by decision-makers, research is also experiencing a surge in field studies involving interactions between the explainability tools and human users. We continue to explore this aspect in the next chapter.