AI Explanations Whitepaper

About This Whitepaper

This whitepaper is a technical reference accompanying Google Cloud's AI Explanations product. It is targeted at model developers & data scientists responsible for designing and delivering ML models. Our goal is for them to leverage AI Explanations to simplify model development as well as explain the model's behavior to key stakeholders.

Product managers, business leaders and end users may also find parts of this whitepaper relevant, particularly around use cases enabled by AI Explanations and, crucially, considerations around its proper usage as well as its current limitations. Specifically, we refer these readers to the "*Usage Examples*" as well as the "*Attribution Limitations and Usage Considerations*" sections.

Overview

Evolution of Machine Learning

In the quest for more accurate AI, the availability of compute resources coupled with increasing dataset sizes have fueled a trend towards more complex non-linear models.

Indeed, we've seen a progression from handcrafted rules and heuristics, to linear models and decision trees, ensembles and deep models to, most recently, meta-learning or models that create other models.
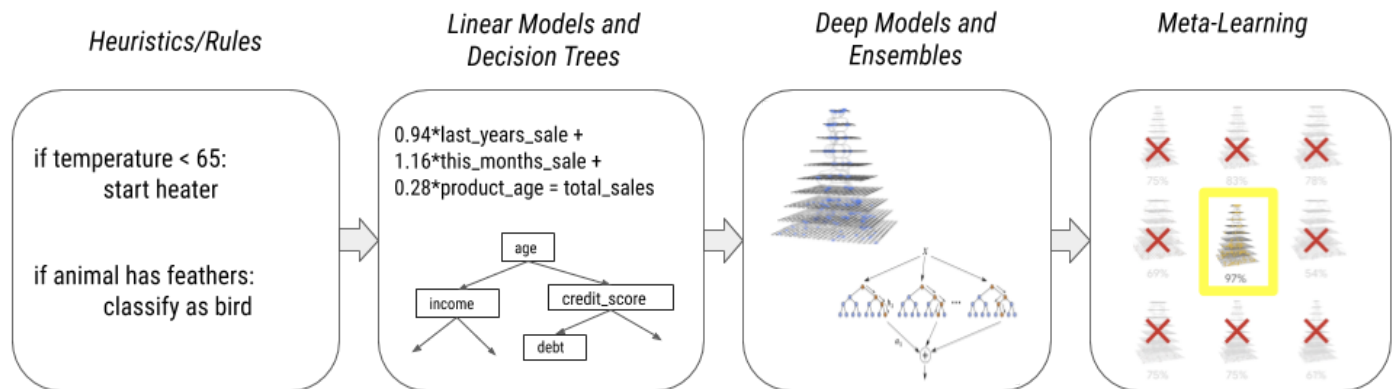
These more accurate models have resulted in a paradigm shift along multiple dimensions:

- **Expressiveness** enables fitting a wide range of functions in an increasing number of domains like forecasting, ranking, autonomous driving, particle physics, drug discovery, etc.
- **Versatility** unlocks data modalities (image, audio, speech, text, tabular, time series, etc.) and enables joint/multi-modal applications.
- **Adaptability** to small data regimes through transfer and multi-task learning.
- **Efficiency** through custom optimized hardware like GPUs and TPUs enabling practitioners to train bigger models faster and cheaper.

On the flip side, these more complex models have become increasingly opaque. This, coupled with the fact that these models are still fundamentally built around correlation and association, have resulted in several challenges:

- **Spurious correlations** can be learned from the data, often hampering the model's ability to generalize and leading to poor real world results.
- **Loss of debuggability and transparency** leading to low trust as well as the inability to fix or improve the models and/or outcomes. Furthermore, this lack of transparency impedes adoption of these models, especially in regulated industries e.g. Banking & Finance or Healthcare.
- **Proxy objectives** resulting in large differences between how models perform offline, often on matching proxy metrics, compared to how they perform when deployed in the applications.
- **Loss of control** due to model practitioners' reduced ability to locally adjust model behavior in problematic instances.
- **Undesirable data amplification** reflecting biases that don't agree with our societal norms and principles.

These challenges highlight the need for explainability in order to keep the humans in the loop and empower them to develop and leverage AI responsibly.

Explainable AI

Systems built around AI will affect and, in many cases, redefine medical interventions, autonomous transportation, criminal justice, financial risk management and many other areas of society. However, considering the challenges highlighted in the previous section, the usefulness and fairness of these AI systems will be gated by our ability to understand, explain and control them.

The field of XAI (eXplainable AI) has seen a resurgence since the early days of expert systems a few decades ago. In the paper Towards A Rigorous Science of Interpretable Machine Learning,

Doshi-Velez and Kim define XAI as the *"ability to explain or to present in understandable terms to a human"* drawing upon the Merriam-Webster dictionary definition of "interpret" and adapting it to the interaction between humans and intelligent agents.

Research progress in XAI has been rapidly advancing, from input attribution (LIME, Anchors, LOCO, SHAP, DeepLift, Integrated Gradients, XRAI etc.), concept testing/extraction (TCAV, DeepR, Towards Automatic Concept-based Explanations), example influence/matching (MMD Critic, Representer Point Selection, Influence Functions, Attention-Based Prototypical Learning), distillation (Distilling the Knowledge in a Neural Network, Distilling a Neural Network Into a Soft Decision Tree). Furthermore, we continue to see novel approaches to inherently interpretable and controllable models like Deep Lattice Networks and Bayesian models.

In addition to needing to probe the internals of increasingly complex models, which in and of itself is a challenging computational problem, a successful XAI system must provide explanations to *people*, meaning that the field must draw on lessons from philosophy, cognitive psychology, Human−Computer interaction (HCI) and social sciences. The interested reader can refer to Miller's *Explanation in artificial intelligence: insights from the social sciences* and the series on *Explaining Explanations by Hoffman, Klein and Miller* (Part 1, Part 2, Part 3 and Part 4). Furthermore, the VIS and CHI conferences are becoming a great reference for work at the intersection of HCI and XAI (e.g. Designing Theory-Driven User-Centric Explainable AI, Gamut, AILA).

XAI itself is one element of building safe, dependable AI systems. As technical capabilities to understand models have improved, so too has our understanding of the pitfalls of AI. Initiatives like PAIR (People + AI Research) provide open-source tools and resources to guide ethical AI development.

Human Reasoning and Explanations

An explanation entails an interaction between a human and an intelligent agent[1]. We start by highlighting some relevant aspects of human reasoning in the context of explanations as they feed directly into design choices we've made in Cloud AI's explainability offering. We believe it's crucial to internalize these concepts as that will lead to better outcomes in successful applications of XAI.

This section is a summary of key concepts, drawing upon the vast body of work from HCI, philosophy, psychology and cognitive sciences (cited in the previous section).

---

[1] The intelligent agent can be an ML model and is assumed to be in this whitepaper.

Seeking Explanations

The primary function of explanations is to facilitate learning better mental models for how events come about (see Lombrozo's [Structure and Functions of Explanations](#)). People then use these mental models for prediction and control. It's natural then to expect people to ask for explanations when faced with abnormal or unexpected events, which essentially violate their existing mental model and are therefore surprising from their point of view. Furthermore, people seek to monitor for these unexpected events and attempt to proactively mitigate them if their occurrence is costly (e.g. an unexpected flu epidemic that sparks a large scale medical campaign).

Explanations are therefore used to find causes for particular events and generalize concepts. In the context of AI, we summarize the following explanations purposes:

- **Informing and supporting human decision making** when AI is used as a decision aid.
- **Improving transparency** by creating a shared understanding between the AI and the human.
- **Enabling debugging** when the system behaves unexpectedly.
- **Enabling auditing** in the context of regulatory requirements.
- **Verifying generalization ability** and **moderating trust** to appropriate levels.

Through these functions, a model user can make better decisions and a model builder can improve the model itself.

Facts and Foils

In seeking explanations, people don't just ask why event P happened; they ask why event P happened *instead of some other (sometimes implicit) event* Q. P is the *target event*, which did occur and is therefore referred to as *fact*. Q is the contrast event that did not occur, which we will refer to as *foil* (adopting [Lipton's terminology](#))[2].

For example, given the question "*why did Gary smash his plate?*", there is an entire class of potential foils represented by the negation *"Gary doesn't smash his plate"*. One reasonable foil is Gary leaving his plate in one piece, and a possible explanation is because he was upset. Another foil is smashing his glass instead, and a possible explanation is because he was at a Greek wedding rather than a Jewish one. Oftentimes, when foils are implicit, they can be inferred by language and tone.

---

[2] Some authors refer to Q as the *counterfactual*. This can be confusing in the context of determining causality where counterfactuals are relative to the cause C of event P (ie. they are hypothetical cases where cause C did not occur) whereas foils are relative to the outcome (ie, they are hypothetical alternate outcomes to event P).

In the context of XAI, foils are other (real or synthetic) examples in the dataset relative to some example being explained, and a good default for implicit foils can be a proxy "average-looking" example in the dataset. This justifies the common practice of using feature medians/modes as "baselines" for ablation studies[3] whereby features are "ablated" one by one by switching their values to the medians/modes and their respective contribution to the final prediction is recorded.

It's natural to think of explanations as contrastive, given that people mostly seek explanations for surprising events. In such cases, people expect to observe a particular event but then observe another - the surprising observed event is the fact and the expected one is the foil. This points to a unique opportunity in designing XAI systems to better satisfy an explainee's needs: allow the explainee the ability to specify the foil, thereby enabling them to point to the part of the model behavior they do not understand. This in turn helps *focus* the explanation on the *difference* between the observed fact and the expected foil, in turn making the explanation more relevant to the explainee[4]. An example from credit scoring applications is the "points lost" system where the foil is selected to be a person with "perfect credit", the explanations are then around deducting points due to suboptimal factors like having "short credit history" or "many open accounts with balances" etc.

Google Cloud AI Explanations

In order to support Cloud AI's mission of enabling useful and responsible [AI deployments](#), we are building a comprehensive XAI offering drawing upon years of experience producing and leveraging explainability techniques in mission-critical AI applications across Google.

We recognize that the explainability space is evolving quickly and is largely still a research frontier so we've taken the pragmatic approach of focusing our offering around tools and high quality implementations of methods that have largely proven their usefulness in deployed AI applications.

As the space evolves, we are committed to improving our existing offering as well as adding new methods and techniques to meet our users' needs.

Feature Attributions

Our current offering centers around instance-level feature attributions, which provide a signed per-feature attribution score proportional to the feature's contribution to the model's prediction. In

---

[3] The term has its roots in the field of experimental neuropsychology. Studies were conducted in the 1960s and 1970s where parts of animals' brains were removed to study their behavioural effects.

[4] Note that providing two complete explanations vs. a contrastive one does not take advantage of the contrastive structure of questions.

other words, this functionality will let you understand what led your model to a particular prediction. For example, if you have a model that predicts whether or not someone will be approved for a loan, the explanations service could tell you that for a specific person's application, their account balance and credit score signaled the model's prediction most.

This functionality is integrated into our managed offering through [AI Platform Prediction](#) as well as [AutoML Tables](#), enabling attributions on a wide range of TensorFlow models across various data modalities.

Shapley Values

Oftentimes, the features used in an ML model are not independent so the task of attributing the impact of each feature on the model's final prediction is non-trivial. We therefore leverage the well established [Shapley](#) concept from cooperative game theory proposed by Lloyd Shapley in 1953[5]. Following is a gentle introduction to the concept and a discussion of its adaption to AI models.

We'll explain Shapley values by using the example of a company with a set N of three employees $\{A, B, C\}$ (the "participants"), and a profit outcome $v(N)$. Let's say we want to distribute profit among the 3 employees based on how much they contributed to achieving that outcome. The Shapley value gives us a framework for doing so. It considers all the ways that adding each employee improved the profit compared to the *foil* of not having those employees. It then assigns value correspondingly.

Now let's say we have the following profit outcomes for different combinations of employees N:

- $v(\{\}) = 0$
- $v(\{A\}) = 10$
- $v(\{B\}) = 20$
- $v(\{C\}) = 30$
- $v(\{A, B\}) = 60$
- $v(\{B, C\}) = 70$
- $v(\{A, C\}) = 90$
- $v(\{A, B, C\}) = 100$

So, when the company had no employees, its profit was 0, this will serve as the *foil*. With just A or B as employees, its profit was 10 and 20, respectively. You can see that the profit generally grows as we add more employees. For example, with just employee A, the company had a profit of 10.

---

[5] Shapley, Lloyd S. "A Value for n-person Games". Contributions to the Theory of Games 2.28 (1953): 307-317

Adding B raised the profit to 60. Does that mean B alone contributed 50 to the profit? Perhaps not, as adding B when C was the only employee only increased the profit by 40 (70 - 30).

To account for this, the Shapley value considers all the permutations of employees joining the company. We first compute all unique ways of growing the company from 0 to 3 employees:

1. $\{\} \rightarrow \{A\} \rightarrow \{A, B\} \rightarrow \{A, B, C\}$
2. $\{\} \rightarrow \{A\} \rightarrow \{A, C\} \rightarrow \{A, B, C\}$
3. $\{\} \rightarrow \{B\} \rightarrow \{A, B\} \rightarrow \{A, B, C\}$
4. $\{\} \rightarrow \{B\} \rightarrow \{B, C\} \rightarrow \{A, B, C\}$
5. $\{\} \rightarrow \{C\} \rightarrow \{B, C\} \rightarrow \{A, B, C\}$
6. $\{\} \rightarrow \{C\} \rightarrow \{A, C\} \rightarrow \{A, B, C\}$

Each time we add an employee, we compute a credit for that employee based on how much the profit grows after they were added. The path (1) gives us these credits to each employee:

- A: $v(\{A\}) - v(\{\}) = 10 - 0 = 10$
- B: $v(\{A, B\}) - v(\{A\}) = 60 - 10 = 50$
- C: $v(\{A, B, C\}) - v(\{A, B\}) = 100 - 60 = 40$

Finally, we need to compute the credits for each possible path, then average the results together.

1. $\{\} \rightarrow \{A\} \rightarrow \{A, B\} \rightarrow \{A, B, C\} \parallel A = 10, B = 50, C = 40$
2. $\{\} \rightarrow \{A\} \rightarrow \{A, C\} \rightarrow \{A, B, C\} \parallel A = 10, B = 10, C = 80$
3. $\{\} \rightarrow \{B\} \rightarrow \{A, B\} \rightarrow \{A, B, C\} \parallel A = 40, B = 20, C = 40$
4. $\{\} \rightarrow \{B\} \rightarrow \{B, C\} \rightarrow \{A, B, C\} \parallel A = 30, B = 20, C = 50$
5. $\{\} \rightarrow \{C\} \rightarrow \{B, C\} \rightarrow \{A, B, C\} \parallel A = 30, B = 40, C = 30$
6. $\{\} \rightarrow \{C\} \rightarrow \{A, C\} \rightarrow \{A, B, C\} \parallel A = 60, B = 10, C = 30$

Averaging out these values, we obtain the Shapley values: $A_{avg} = 30$, $B_{avg} = 25$, $C_{avg} = 45$

The Shapley value is the unique method that satisfies the following desirable axioms, which motivates its use and are the main reason we chose it to power our feature attributions offering:

1. **Completeness** (also known as efficiency): Shapley values sum up to the difference between the target outcome and the outcome of the *foil*. 30 + 25 + 45 = 100 - 0 (outcome of {})
2. **Symmetry**: For two participants i, j and S, where S is any subset of participants not including i, j, when {S, i} = {S, j} → i and j should have the same attribution value.
3. **Dummy**: For one participant i and S, where S is any subset of participants not including i, when {S, i} = {S} → i should get zero attribution.

4. **Additivity**: For two outcome functions, the Shapley values of the sum of the two outcome functions should be the sum of the Shapley values of the two outcome functions.

Adapting this concept to AI models, the outcome $v(X)$ is the model's output prediction and the participants $X = [x_1, x_2, x_3]$ are the input features. We are therefore leveraging the Shapley framework to attribute the model's prediction onto its input features[6].

A notable consideration when applying Shapley is that there is no clearly defined way of *removing* a feature for a given instance's prediction, which, as shown above, is needed in the calculation process of the Shapley value. As it turns out, this is another manifestation of the *foil* requirement and a good segue into the next section.

Baselines and Counterfactuals

In the previous example, the *foil* was the employees being absent, represented by the empty set {} with an outcome of 0. The choice of an empty set isn't magical. In fact, it reflects the core question we asked "how much can we expect the profit to grow *when adding each of the three employees?*" We could have chosen another *foil* to reflect a different question. For example, if we wanted to ask "how much credit should B & C get respectively for profits earned *assuming A was already employed*", then our *foil* would become {A} with a base outcome of 10 instead of 0. As you can see, we naturally get different results depending on the *foil*, which in turn informs the question we're asking.

In the context of the AI model, recall that we had three input features $X = [x_1, x_2, x_3]$ representing employees A, B and C respectively. We represent the foil with a special token β, so:
$$\{\} \rightarrow [\beta, \beta, \beta]$$
$$\{B,\ C\} \rightarrow [\beta, x_2, x_3]$$

We can further generalize this to a per-feature token yielding what we refer to as the *baseline* instance:
$$= [\beta_1, \beta_2,\ \beta_3]$$

Putting it all together, the complete question we're asking to explain for an AI model is "*why did the model make this prediction <u>instead of that other prediction?</u>*". The first part of the question is informed by the instance in question X and the second part is informed by the baseline . It then

---

[6] There are many ways to apply the Shapley values that differ in how they reference the model, the training data, and the explanation context. This gives rise to a multiplicity of Shapley values used in explaining a model's prediction, which is somewhat unfortunate considering Shapley's uniqueness property. A thorough discussion of this topic can be found in [The many Shapley values for model explanation](#) by Mukund Sundararajan and Amir Najmi. Our methods fall into the Baseline Shapley category and support multiple simultaneous baselines across various input data modalities.

becomes natural to introduce the concept of a baseline score, which is simply the model's prediction for the baseline instance .

So when we compute Shapley values for an AI model, we need to choose a baseline that reflects the desired *foil*. This means picking some example from the domain of the model that's appropriate to compare against. Furthermore, during the computation process, we generate multiple trials that go from the baseline to the desired target instance, ie. from $\rightarrow X$. Such trials include instances like $[\beta_1, x_2,\ x_3]$ and $[x_1, x_2,\ \beta_3]$ which we refer to as *counterfactuals*. We rely on the model to predict outcomes for these counterfactuals, which are then used to compute the Shapley values using the process described above.

In summary, the choice of *baseline*, from which *counterfactuals* are derived, can heavily influence the attributions themselves. This raises the question of how to choose a good baseline.

Uninformative Baselines

An uninformative baseline is one where there's no additional information being presented. It typically maps to the most likely foil, which in turn is often implicit in the way the question is formulated. Going back to "*why did Gary smash his plate?*", the uninformative baseline here is then simply "*instead of not smashing the plate*".

In the context of AI models, an uninformative baseline instance looks like an "average instance" from the training dataset or an instance with low information resulting in a high entropy model prediction.

Examples of uninformative baselines include black/white/random images for vision inputs, stopwords (like "the", "a", "and", etc.) and separator tokens for text inputs, medians for numerical inputs, and mode for categorical inputs.

Informative Baselines

Instead of using uninformative baselines, it can be useful to select baselines with high information content to contrast with a given instance and highlight the most important feature attributions that resulted in the observed outcome difference.

For example, one might ask a reasonable question like "*why did my credit score drop 50 points since last month?*" This implies an informative baseline of the account state a month ago and a desire to understand the feature differences that caused this drop.

Another way informative baselines are helpful is in allowing us to make certain features invariant if they are not actionable. For example, users are unable to change their demographic factors in heart disease assessment, but they can change how often they exercise, drink etc.

Attribution Methods

Computing the exact Shapley value is infeasible from a computational standpoint as it scales exponentially[7] to the number of input features.

Given the prescribed interface of users either directly providing a model or training data to AutoML which results in a model, we only consider post-hoc approaches for the current version. We offer the following methods for TensorFlow models which conform to our requirement of supplying a baseline[8] and retain the axiomatic guarantees while providing a more computationally feasible alternative:

- **Integrated Gradients**: Enables computing the Aumann-Shapley value for differentiable models as discussed in [Axiomatic Attribution for Deep Networks](#) by Mukund Sundararajan et al.
- **Sampled Shapley**: Provides an approximation through sampling to the discrete Shapley value. Estimation error bounds are discussed in [Bounding the Estimation Error of Sampling-based Shapley Value Approximation](#) by Sasan Maleki et al.
- **XRAI**: Specialized for image inputs, combines Integrated Gradients with oversegmentation and region selection to determine attribution density at the level of regions instead of pixels as discussed in [XRAI: Better Attributions Through Regions](#) by Andrei Kapishnikov et al.

Integrated Gradients is recommended for neural networks and differentiable models in general. It offers computational advantages especially for large input feature spaces (e.g. images with thousands of input pixels). XRAI is recommended for image models where it's desirable to localize attributions at the region vs. pixel level. Sampled Shapley is recommended for tabular and non-differentiable models, which is the case in AutoML Tables models consisting of meta-ensembles of trees and neural networks. This method scales to the number of input features and is therefore slower for models with many input features.

We believe the above attribution methods offer good coverage of model support across multiple data modalities so we focused on high quality implementations of them and are continuing to invest in other methods which can offer computational or theoretical advantages for certain use cases.

The interested reader can refer to alternative methods proposed in [A Unified Approach to Interpreting Model Predictions](#) by Scott Lundberg et al. which proposes SHAP (SHapely Additive exPlanations) approximations relying on conditional expectation sampling from the model's

---

[7] Technically, combinatorially; a Shapely computation is O(n!).

[8] We enable passing multiple simultaneous baselines and return the mean attributions.

dataset, please refer to the paper [The many Shapley values for model explanation](#) by Mukund Sundararajan et al. for a discussion of this approach as compared to the baseline one.

Furthermore, Integrated Gradients is one method in the family of other gradients-based methods like [Grad-CAM](#) by Ramprasaath R. Selvaraju et al., [SmoothGrad](#) by Daniel Smilkov et al., [LRP](#) by Alexander Binder et al. and [DeepLIFT](#) by Avanti Shrikumar et al., the latter two methods rely on discrete gradients whereas the other methods rely on applying the model's gradient operator.

Quantifying the Approximation Error

Attribution methods rely on efficient approximations of the Shapley value. Therefore, it's important to quantify the error and enable users to improve the approximation at the cost of increased computation, by modifying certain parameters of the attribution method. The AI Explanations service returns this approximation error to users to ensure robustness. We describe our measure of approximation error for the attribution techniques we provide.

Integrated Gradients/XRAI Approximation Error

Integrated Gradients, as the name suggests, integrates gradients of a scalar output with respect to the input along an infinite set of examples between the input we would like to explain, and an auxiliary input called the baseline. Computing this integral in a closed form is possible for certain types of networks, but is still computationally expensive. Instead, we approximate the integral using a summation.[9] The user can increase the number of steps in the integral approximation to reduce error. The exact version of Integrated Gradients has the property that the feature attributions sum to the prediction difference between the input score and the baseline score. We measure the approximation error as the relative deviation between this difference and the sum of the approximate feature importances[10]. More formally:

$$Approximation\ Error(x,\ x') \ = \ \frac{|F(x) - F(x') - \sum\limits_{i}^{N} IG_i(x, x')|}{|F(x) - F(x')|}$$

where F is the function being explained, IG is Integrated Gradients, x is the input and x' is the baseline.

Sampled Shapley Approximation Error

The Shapley value for a feature can be expressed as the average marginal contribution of that feature over all possible permutations which entails an O(n!) computation. As the method's name

---

[9] See proposition 1 in [Axiomatic Attribution for Deep Networks](#).
[10] Note that there may exist degenerate cases where a summation along the under-sampled gradients path leads to cancelations resulting in low approximation error. Empirically, we haven't found this to be an issue as long as the number of integral steps is reasonably high.

suggests, we sample permutations. The user can increase the number of permutations sampled to improve the approximation. We measure the approximation error by checking the convergence of the attributions. Specifically, if we run our computation over a set of sampled permutations K of size $|K|$, we identify the change in the approximate shapley values by leaving one permutation out. That is, we compute the value over without one permutation in K denoted as k_j, and separately over all K, and measure the relative absolute differences in the computed Shapley values. More formally:

$$Approximation\ Error(x,\ x') \ = \ \frac{\sum\limits_{j}^{|K|} \sum\limits_{i}^{N} |SS_i^K(x, x') - SS_i^{K \setminus k_j}(x, x')|}{|K| \sum\limits_{i}^{N} |SS_i^K(x, x')|}$$

where SS is Sampled Shapley, x is the input and x' is the baseline.

Aggregate Attributions

The unique properties of Shapley-based attributions enable consistent comparisons between instances and across models, as long as the same baseline is used. Furthermore, it's possible to aggregate these attributions across multiple instances in order to get a more global view of model behavior and increase statistical confidence in inferences made from looking at single instance attributions. This is because the attributions are linearized, summing to the prediction and baseline scores difference and are therefore decomposable[11].

For example, we can get a sense of overall influence of feature column[12] j in a given dataset or slice of N instances by computing $influence_j \ = \ \frac{1}{N} \sum\limits_{i=0}^{N} |attrib_{ij}|$. If the column is categorical, we can not only get the overall influence of the column as a whole but also of individual features within the column. In cases where some columns are missing, we can condition on their presence during the aggregation process. Another useful aggregation is across clusters of columns, e.g. if some columns share the same data source, we can compare attributions split by data source.

Furthermore, one can perform differential analysis to compare:

- Multiple separate models on the same dataset
- The same model on multiple slices within the dataset

---

[11] It's important to pay close attention to the scale at which the attributions are calculated. When there's a nonlinear final activation like sigmoid or softmax before the final model output, it's preferable to use the logits for aggregations.

[12] A feature column is a logical grouping of features e.g. a feature column "country" groups individual countries like "Lebanon", "Slovakia", "France", "Brazil", etc.

For example, say we want to compare average column influence between two groups of examples or dataset slices $delta\ influence_j = \left| \frac{1}{N_1} \sum_{i=0}^{N_1} |attrib_{ij}| - \frac{1}{N_2} \sum_{i=0}^{N_2} |attrib_{ij}| \right|$, in the case of comparing different models on the same dataset, we can pair the attributions on the same data points to calculate the per-instance deltas before taking absolute values. We can even perform statistical tests comparing, say, attributions from different models to understand whether/where the models have statistically meaningful differences.

Whenever we discuss aggregates and slice analysis, it's important to apply caution in interpreting the results by leveraging standard data analysis best practices and properly adjusting for confounders. A popular example of an issue that might arise is [Simpson's Paradox](.).

Comparison to Alternative Global Explanation Methods

Instead of starting from local attributions and aggregating up, It's possible to explain the global behavior of a model by inspecting the model's own structure or the structure of a similar model. Alternative global explanations methods have their pros and cons. For instance:

- Communicating the entire model, for instance all the coefficients of a linear model, or the entire tree structure of a decision tree. This approach ensures that the human understands the full picture which would explain model behavior on all inputs. However, it is only possible to communicate the entire model for simple models, with a small number of parameters and that often entails some sacrifice in accuracy.

- Extracting rules or constructing a simple model that approximates a complex model. This form of the output can be richer than feature attributions, it can for instance communicate important feature interactions. However, the issue with this approach is that the surrogate model is never completely faithful to the original model's workings which limits its use in applications like Financial Services where surrogate models are not considered an acceptable form of explanations for regulatory purposes.

Attribution Limitations and Usage Considerations

While Shapley-based attributions have proven particularly helpful in our experience applying them to real world applications[13] and are continuing to gain popularity in the Explainable AI community, they have their limitations and are meant to be used as a complementary tool, not a replacement to other available model analysis/evaluation techniques, and most crucially, the practioners' best judgement.

---

[13] Please refer to the "*Usage Examples*" section.

Human Analysis Pitfalls

The field of explainability revolves around empowering the human in the loop and attributions are ultimately intended to aid in human analysis.

An important consideration from cognitive psychology about human reasoning is that we don't always reason rationally and oftentimes employ heuristic reasoning to make decisions faster. Croskerry described the *dual process model* in diagnostic reasoning which was later popularized by Kahneman in his Thinking Fast and Slow bestseller. This model presents two systems of thinking:

- **System 1 thinking** is heuristic, fast, low effort as the employed heuristics are subconscious cognitive shortcuts. For example, recognizing someone's face is done heuristically by mentally clustering and comparing with familiar faces and, while this allows for quickly recognizing an acquaintance, leads to overmatching to celebrity faces "*you look like the lead actress from that TV series!*".
- **System 2 thinking** is analytical, slow, high effort as the reasoning process is more deliberate and rational. People need to be trained, in a domain-specific manner, to employ System 2 thinking properly. For example, doctors are trained to employ the hypothetico-deductive process to diagnose a patient.

Reasoning errors can be attributed to each of these two systems, as well as their interplay during the decision making process. For example System 1 errors stem from **heuristic biases** which oversimplify the situation and System 2 errors stem from **misattributed trust** which leads to overreliance on a tool with questionable reliability/quality. Furthermore, A System 2 thinking method like the hypothetico-deductive model can be corrupted by **confirmation bias**.

The considerations listed here are concrete materializations of these concepts as applied to reasoning about model explanations:

1. It is possible that the model behaves correctly, but not in the same way as a human might approach the same prediction task. The intention here isn't to point out distinctions between models and humans, but to acknowledge that there are often different ways to solve the same prediction problem. For instance, the data may consist of correlated features. Correlation constitutes redundancy, i.e., the model could perform correctly by relying on various combinations of features. If the specific combination that the model relies on differs from what the human expects, the human may feel the model is wrong when it is not. This isn't a unique issue to attributions of complex models, in fact simple linear models with L2 regularization is another manifestation of the same when input features are correlated.

2. Related to the first point, models are designed to discriminate between a given set of labels. Such discrimination is sometimes possible without referencing all the features that

a human would use to discriminate between the open-ended set of objects in the real world. A user may ignore this fact and expect the explanation to cover a wider set of features than what the model uses for discrimination. For example, this can happen when the attributions are localized to a subset of all regions indicating a lung cancer diagnosis in an x-ray, which were enough for the model to predict the disease class.

3. The human may feel that the model is wrong, when in fact it is right. Or, the human may feel the model is right if it behaves similar to the human (confirmation bias), even when its logic is in fact wrong as highlighted by the attribution method, leading the human to believe the attribution is wrong, when in fact it is right. People handle these discrepancies by having conversations. The What-If Tool exists precisely to enable such back-and-forth interactions between the human, the model and the attributions.

4. Proper presentation/visualization matters. The attributions are rarely communicated as raw numbers. It is important that the visualizations don't deform the raw numbers but show them faithfully. This is not trivial and we've documented some learned lessons in [Exploring Principled Visualizations for Deep Network Attributions](#) and followed best practices in designing our offering. At the end of the day, there is still a risk of misinterpretation.

5. Finally, as with any data analysis tool, it's important to account for statistical robustness issues that may arise from small datasets and/or high variance models.

Conceptual Limitations

Following are some limitations that stem from the concept/definition of attributions:

1. **Attributions depend on the data:**

   a. Attributions are projections of the underlying model, which in and of itself, is a partial reflection of the underlying data it's trained on. Attributions only surface the patterns a model discovered in the underlying data, and not any sort of prior relationship.

   b. When attributions indicate something is wrong, one must carefully decide whether to blame the model or the data. When diagnosing data issues, it's important to pinpoint whether the training set and the testing set are drawn in an independently and identically distributed (IID) manner, as skew can be a common source of data problems. The interested reader can refer to [Did the Model Understand the Question](#) by Pramod K. Mudrakarta et al. for a deeper dive on this topic.

2. **Attributions depend on the model**:

a. Real world data can encode rich information, beyond what's available in the training set, that the model itself did not or cannot learn, therefore this information can not be reflected through the lens of the model. A weak attribution doesn't necessarily mean there isn't a pattern or relationship between that feature and the outcome -- it could merely be that the model wasn't able to discover it!

b. When humans reason, they have access to information and concepts beyond what models can learn from the data they're trained on. Attributions can therefore be leveraged by humans to ascertain whether the model learned certain behaviors that are important for generalization ability.

c. Similar to how model predictions are prone to adversarial attacks, attributions are prone to the same as they involve interrogating models along some manifold. Adversarial robustness for both model predictions and model explanations is an active area of research, we refer the interested reader to seminal work [Explaining and Harnessing Adversarial Examples](#) by Ian J. Goodfellow et al. as well as more nascent research in the area of attributions robustness like [Interpretation of Neural Networks is Fragile](#) by Amirata Ghorbani et al.[14]

3. **Attributions depend on baselines[15]:**

a. The values of each attribution and interpretation thereof depend entirely on the choice of baseline, it's as important as knowing what questions to ask when seeking an explanation. One must never omit the baseline from any discussion of the attributions, and take care in choosing one useful for the problem.

b. We attempt to choose good, default baselines. For example, in AutoML Tables, we leverage univariate feature distribution stats computed on the training set to choose an uninformative baseline consisting of median values for numeric features and modes for categorical features. We currently have less context for models deployed on AI Platform Prediction, so users are encouraged to carefully set their baselines in the explanation metadata. For image inputs, the default baseline consists of two images, one black and one white which makes for a reasonable default for most use cases.

4. **Attributions are communicated at the level of input features, this entails some loss of information:**

---

[14] The interested reader can also refer to [counter-arguments](#) against these adversarial attacks.

[15] This point has been touched upon first in "Facts and Foils" from a human reasoning standpoint and in "Baselines and Counterfactuals" from an algorithmic standpoint.

a. For some applications, this can be the wrong level for the explanation as model behavior can not be fully reflected. Here is a simple, but artificial example: Suppose that the model inputs are boolean, and it implements a [XOR](#), i.e. the model outputs 1 if there are an odd number of 1's in the input, and 0 otherwise. The only reasonable attributions assign equal feature importance to every variable. If the human does not apriori know the model's logic, this output is not going to aid the human's understanding of the model's logic.

b. Attributions are useful when a fraction of the features are important to a prediction, the attribution method localizes these features, and the human checks if the important features are as expected. A practical example of this limitation occurring in practice is that vision models typically assign importance to textures instead of shapes, and for images where textures dominate, one may see widely scattered attributions across the image textures from which it's non-intuitive to understand the shape vs. texture attribution importance.

c. Feature interactions / higher order effects are redistributed to a single number per feature which impairs faithful communication of such effects. For example, in a sentiment detection application, negation "not great" and intensification "so nice" are examples of said interactions. Note that every attribution technique (Integrated Gradients or Sampled Shapley) does this computation in a reasonable way, but the two methods take slightly different approaches. The difference is captured by a simple but artificial example: Suppose the function is min(x1, x2), Sampled Shapley gives some attribution to both variables, whereas Integrated Gradients gives attributions entirely to the argmin.

5. **Attributions do not summarize the entire model behavior:**

   a. As we have discussed so far, attributions explain only one instance but they can be aggregated[16] across the data set to get a more global view. However, this is still contingent on the observed data and does not include all possible instances. Furthermore, your choice of aggregation method (e.g. functions like mean or median, weights like uniform or weighted, etc.) across the same attributions may produce different outcomes depending on your analysis needs.

6. **Attributions are efficient approximations of Shapley values:**

   a. The approximation process can possibly entail a large convergence error, depending on the attribution method parameters. For example, Integrated Gradients

---

[16] Please refer to the "*Aggregate Attributions*" section.

and XRAI have a number of steps parameter which determines how many steps are used along the gradients path to approximate the integral calculation. Similarly, Sampled Shapley has a number of paths parameter which determines the number of paths sampled for each input feature. Generally, the more complex the model, the more approximation steps are needed to reach reasonable convergence.

b. Another important consideration is that, while we've optimized for efficiency and low latency, this approximation process is still computationally expensive relative to e.g. a prediction request which entails a single forward pass. From an algorithmic complexity perspective, Integrated Gradients scales to the number of approximation steps, whereas Sampled Shapley scales to the number of paths * the number of input features, so the former can offer significant computational benefits when the feature space is large.

Explanation Model Metadata

When we designed our explainability offering, we needed to overcome the challenge of supporting arbitrary models in order to achieve parity with both AI Platform Prediction and AutoML.

Furthermore, we wanted to build a unifying API supporting various input data modalities as well as encodings (one/multi hot, embeddings, discretized numeric, etc.) in a way that integrates well with TensorFlow-native graph operations to ensure our users don't need to write custom code or change their models in order to leverage the functionality. A simple example to illustrate this need is, say we're given a model with a categorical feature country which is embedded into a lower dimensional space (e.g. 32) and fed as one of the components of a deep model's input layer. This representation allows the network to natively cluster countries into a more statistically efficient distributed representation (e.g. learning concepts like continents, per-capita GDP classes, etc.). Our implementation is able to return attributions back into the individual country feature which is typically what users expect.

In order to support explanations without requiring changes in the model building code, we ask users to point to us relevant tensors in their graph. Namely, users need to identify the output tensor they want the explanation for and to compile a group of input tensors to be attributed for the output tensor values. An input feature can be represented with a single tensor or multiple tensors in certain cases. For example, tensors holding categorical features usually feed to a dense representation such as embedding and one/multi-hot vectors. Hence, we ask users to combine all these semantically connected tensors in a single dictionary along with other parameters such as encoding method and baselines.

This metadata is meant to accompany the saved model in the same folder as it contains relevant information about the tensors in the corresponding model. It is serialized in json format to a file named "explanation_metadata.json". An example can be seen below:

```json
{

  "framework": "Tensorflow",

  "inputs": {

    "age": {                            # name of the feature

      "input_tensor_name": "age:0",    # name of tensor to make attributions to

      "input_baselines": [33],         # baseline for the feature

    },

    "marital_status": {                 # name of the feature

      "input_tensor_name": "marital_status:0",  # input tensor

      "encoded_tensor_name": "marital_status_embedding:0", # name of the tensor
for encoded representation of the input tensor.

      "encoding": "combined_embedding", # encoding type

      "input_baselines": ["single"], # baseline for the input tensor.

    },

  },

  "outputs": {

    "logits": {                         # name of the output

      "output_tensor_name": "scores:0", # name of the tensor to explain

    }

  }

}
```

Note that AutoML Tables users don't need to supply the metadata file, as we're able to automatically infer it when building the model, along with a reasonable uninformative baseline derived from the model's training data statistics.

Visualizations with the What-If Tool

The [What-If Tool](#), built by Google AI's [PAIR team](#), is an interactive visual interface designed to help visualize data and enable probing of ML models with minimal coding. We've [integrated it with AI Platform](#), enabling practitioners to seamlessly analyze and probe their models deployed on AI Platform Prediction.

Working closely with the PAIR team, we've added support to natively visualize attributions for AI Platform Prediction models, giving model builders insight into the features that contributed to the model's predictions This in turn enables deeper analysis to unlock more insights through the combination of counterfactual probing, datapoint similarity, and model comparison.

You can use the What-If Tool with AI Platform models in many different notebook environments. It currently supports Colaboratory, Jupyter, and Cloud AI Platform notebooks. As shown in the example below, you can invoke the tool in a few lines of code by passing it a set of datapoints to test and a pointer to the deployed model:

```
config_builder = (WitConfigBuilder(test_examples)

  .set_ai_platform_model('your-gcp-project', 'gcp-model-name', 'model-version')

  .set_target_feature('thing-to-predict'))

WitWidget(config_builder)
```

If the AI Platform model has been set up to provide explanations, then the What-If Tool will visualize them with no additional setup.

When examining the results of running the model on a selected datapoint in the What-If Tool, the feature values will be colored and ordered by the attribution values of the features, with dark red indicating large negative attributions and dark blue indicating large positive attributions, and the baseline score will be shown for additional context. The below examples use a regression model with two input features, "x" and "z", which tries to predict the ground truth label "y".

The feature attribution values can also be used to create custom visualizations in the tool. Below, a user has created a scatter plot of the attributions of the two input features of a model, with each datapoint colored by the absolute value of the error of the model on that datapoint, so darker colored datapoints are the ones with higher error. This custom plot shows that the model performs worse when the input feature "x" has very small or large attributions.

Lastly, feature attributions can be used to analyze aggregate performance across different slices of the test dataset. Below is a view of the performance of a model across 10 different slices of a dataset, where each slice represents a group of test datapoints with a similar feature attributions for the input feature "x". It can be seen that slices with higher attributions for input feature "x" have lower error on average than slices with lower attributions. Note that the table is sorted by the "mean error" measure.

Sort by
Mean error

attributions__x (10 values) ⓘ

Feature Value	Count	Mean error	Median error	Mean absolute error	Median absolute error	Mean squared error	Median squared error
[3.6804547430305306, 290)	74	94.750	95.328	94.750	95.328	8999.964	9087.452
[290, 570)	84	78.983	78.934	78.983	78.934	6255.349	6230.570
[570, 850)	78	64.258	64.708	64.258	64.708	4150.509	4187.150
[850, 1140)	84	48.217	47.847	48.217	47.847	2347.065	2289.311
[1140, 1420)	68	32.397	32.161	32.397	32.161	1069.978	1034.299
All datapoints	800	24.625	23.662	42.071	38.089	2567.475	1450.743
[1420, 1700)	85	18.006	19.024	18.006	19.024	348.833	361.946
[1700, 1990)	86	1.562	1.618	3.933	3.259	22.289	10.618
[1990, 2270)	82	-13.395	-13.244	13.395	13.497	201.453	182.164
[2270, 2550)	70	-28.578	-28.994	28.578	29.039	835.794	843.253
[2550, 2837.0513197214177]	89	-42.443	-42.027	42.443	42.155	1823.472	1777.074

Usage Examples

Following are several possible use cases of our explainability offering which fit into various stages of the model development and deployment lifecycles. This is not meant to be an exhaustive list, more so a set of ideas on how these tools can be generally helpful.

Model Development Workflow

During model development, explainability insights can be a powerful complement to model evaluation and a key enabler in closing the loop and achieving better models.
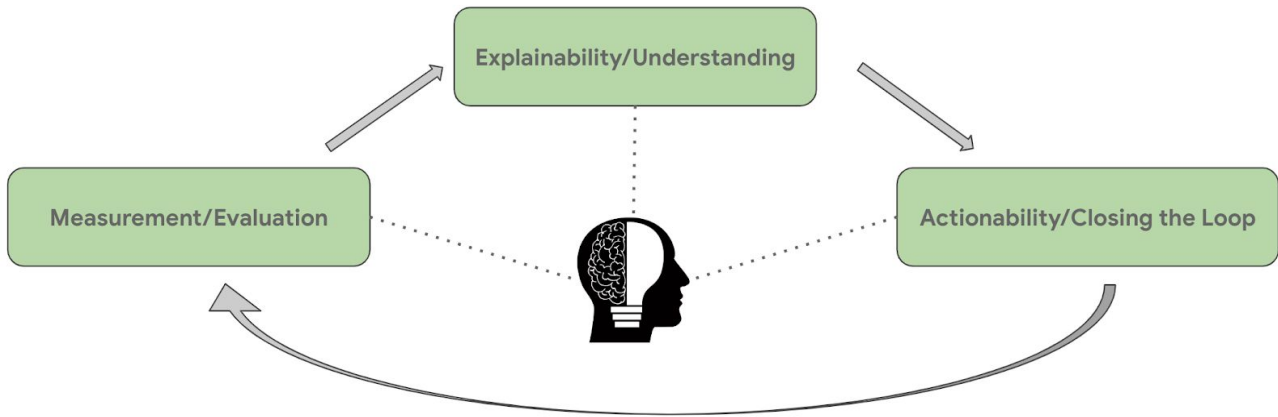

The diagram above is a good illustration of this iteration process, which is centered around the model builder.

- The first step is measuring performance, such as AUC/ACC/F1/RMSE, and evaluating a model's failure modes. This is a fairly standard step in ML development workflows.
- The second step is meant to unlock insights about what the model has learned, model builders are looking at their data through the lens of the model. One can debug failure cases in the hope of finding patterns, as well as spot check success cases to ensure the model isn't cheating and relying on signals that won't generalize.
- The third step is ultimately about taking action, either on the data or the model, informed by the unlocked insights.

Detecting Data Issues

It's not always possible to detect issues with data with standard model evaluation even when following ML best practices of splitting train/validation/test sets and/or k-fold cross-validation. This is best illustrated by an example. An image pathology model is trained to detect various diseases from chest X-Ray scans, the model's quality looks great on the test/holdout set (in fact, perhaps it looks too good to be true):
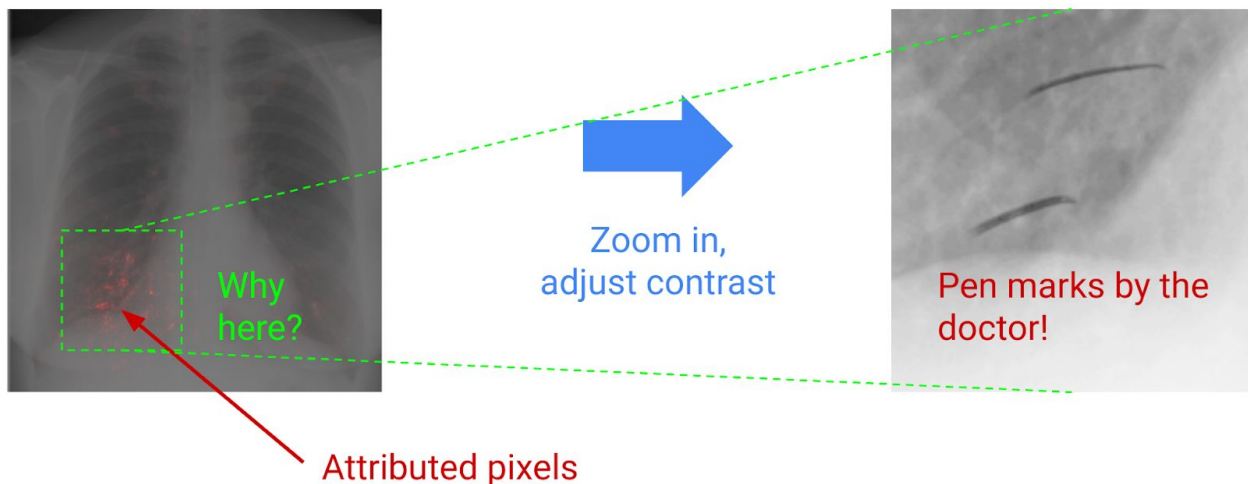
As can be seen above, the attributions were clustering around a seemingly odd region in the X-Ray. Upon closer examination, this area is where radiologist left pen marks. The model learned to rely on these pen marks, which is clearly not desirable from the perspective of being able to generalize to new/unseen instances. Should this model have been deployed, it would be running on images without these pen marks, and its performance would've been markedly worse than the holdout set results.

This is a specific image modality instance of the more general target leakage scenario which happens across any data modality. Another example of this happening on tabular data is from the Safe Driver Kaggle dataset. A model builder may inadvertently include the "id" feature, identifying drivers, which is highly correlated with the target. A random train/test split would have the same driver id in both sets and would therefore result in overly optimistic model evaluation results. On

the other hand, looking at the attributions for a few instances, or overall aggregate attributions, the fact that the model is overly relying on that single feature becomes an obvious flag as can be seen in the following screenshot from AutoML Tables UI:

Feature column name	Column ID	Data type	Status ↓	Value	Local feature importance ❓
Customers	5511192083064422400	Numeric	Required	1	1,214.861 ▬▬▬▬▬
Date	78170350922781116352	Timestamp	Required	2015-10-15	0.000 ▏
DayOfWeek	8393495844581539840	Categorical	Required	7	-103.563 ▪
Open	8969956596884963328	Categorical	Required	0	63.015 ▪
Promo	3205349073850728448	Categorical	Required	1	87.930 ▪

Iterating on Models

Per the "*Aggregate Attributions*" section, knowing the most influential features gives practitioners the ability to iterate on model inputs in a more informed manner, especially for model families where more traditional feature importance measures aren't readily available like deep neural nets.

A common workflow to illustrate this is training a large model on a wide variety of available signals, then refitting a slimmer model by removing signals that weren't considered influential, thereby reducing the model's in-memory and compute footprint without sacrificing much model quality. In some situations, model quality even improves by virtue of removing noisy features and/or focusing the model's learning capacity on fewer high quality signals. A closely related workflow can be applied to categorical features, leveraging fine grained attribution scores to pick the most influential feature values and delegate less relevant values to OOV or hash buckets. This can come in particularly handy for high-cardinality categorical features (e.g. >100k values) and has advantages over alternative techniques like picking the most frequently occurring values which ignore discriminative ability (said differently, oftentimes, less frequent features pack more information about the label and should be retained). Other, more general workflows are possible in this space that make informed model iteration easier and more effective. For instance, some Google Ads teams are using these workflows to debug and better understand their models. With aggregate attributions, they're able to see a ranking of features by importance. This makes it much easier for them to see how their models are performing, and experiment with adding and removing features.

Another important form of model iteration is through augmenting a model's optimization objective, leveraging attributions along with some human priors to infuse domain knowledge that either aids the learning process or constrains the model from learning undesirable associations. An example application in NLP model fairness is presented in [Incorporating Priors with Feature Attribution on Text Classification](#) by Frederick Liu and Besim Avci.

Closing The Loop With Key Stakeholders

In addition to the data scientists and ML Engineers, there are several stakeholders in the model deployment lifecycle:

- Executives and other business stakeholders who make go/no-go decisions on whether to deploy the model.
- Product managers who would like more control over the model's behavior, for example through policy layers.
- Domain Experts who would like to apply their expertise to improve the model in collaboration with model builders.
- End Users who want to understand a model prediction to incorporate it into their decision-making process.

The model builders ultimately need to establish some level of trust with these stakeholders in terms of the model's behavior and fallback policies to avoid catastrophic failures.

There are many examples in the literature of how explainability can help close this loop, from the loan officer deciding on whether to approve a loan based not only on the model's prediction but also on the attributions to the input signals as well as their own best judgement, to the doctor diagnosing disease by assessing medical images in a computer-assisted fashion. An example of the latter can be seen in the study [Using a Deep Learning Algorithm and Integrated Gradients Explanation to Assist Grading for Diabetic Retinopathy](#), which assessed the impact of model assistance in the form of predictions as well as integrated gradients heatmaps (the same as a user would get from AI Platform Prediction).

The study found that "*both types of assistance increased the sensitivity of readers for any level of Diabetic Retinopathy without a significant impact on specificity*" and that "*with increased transparency, model assistance can boost reader performance beyond what is achievable by the model or reader alone.*" Notably, the study also concluded that showing heatmaps when no disease is predicted can in fact be harmful, which underlines the need for careful experiment design when leveraging explainability in end user scenarios - please refer to "*Human Analysis Pitfalls*" section.

Another example where explainability helped close the loop with stakeholders is from our work on Search. The team building [Google Discover](#) cards used explanations to confirm they could entrust ML to rank their cards, instead of using a rule-based system, consequently earning the approval of key stakeholders to launch the model. In this example, it's being used to show a card for a sporting event that's currently happening, which was determined to be a key signal in the model through attributions.
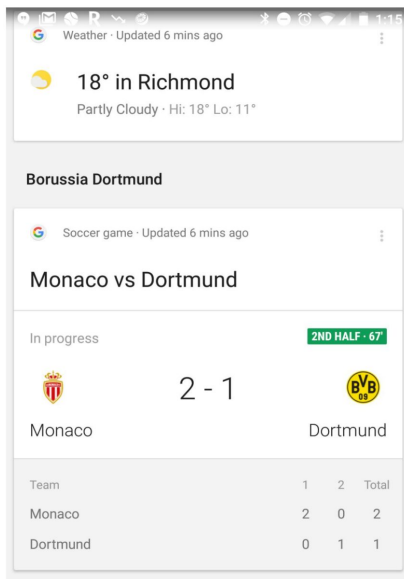
Prediction Auditing and Model Monitoring

Once a model is deployed and is serving live traffic, there are a few ways explainability can help:

- Auditing predictions, especially for the rare/unlikely class. For example, say we've deployed a model to monitor for fraudulent transactions, we can leverage attributions on the percentage of traffic that the model tags as fraud to record into logs which can be analyzed by a human reviewer if/when needed.

- Monitoring the model to ensure it's operating correctly. Traditional techniques monitor for training/serving skew on the model's feature values and predictions. This can be augmented by feature attributions which helps along several dimensions:

- Monitoring attribution skew can be unified across various types of input features (boolean, numeric, categorical, etc.) since attributions have a dense numeric representation. This leads to another dimension of skew monitoring which goes beyond predictions and can be applied on unlabeled instances.
- Intuitively, one should be more worried about more influential features having skew rather than features that are generally less influential. Attributions give us a way to set skew thresholds and tradeoff sensitivity and specificity in a more informed manner.

This idea has proven useful at Google and a retrospective analysis of a hospital machine learning model deployment in [Explainable AI for Trees: From Local Explanations to Global Understanding](#) by Scott M. Lundberg et al. shows some interesting findings in Figure 5.