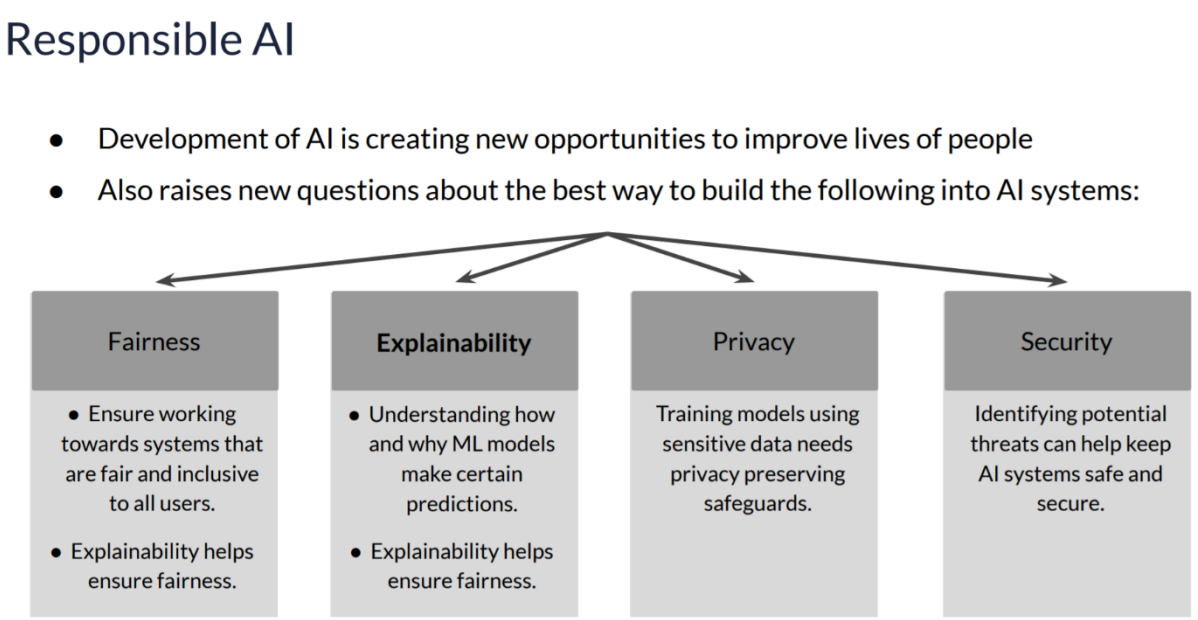
Machine Learning Modeling Pipelines in Production

In the third course of Machine Learning Engineering for Production Specialization, you will build models for different serving environments; implement tools and techniques to effectively manage your modeling resources and best serve offline and online inference requests; and use analytics tools and performance metrics to address model fairness, explainability issues, and mitigate bottlenecks.

Understanding machine learning and deep learning concepts is essential, but if you're looking to build an effective AI career, you need production engineering capabilities as well. Machine learning engineering for production combines the foundational concepts of machine learning with the functional expertise of modern software development and engineering roles to help you develop production-ready skills.

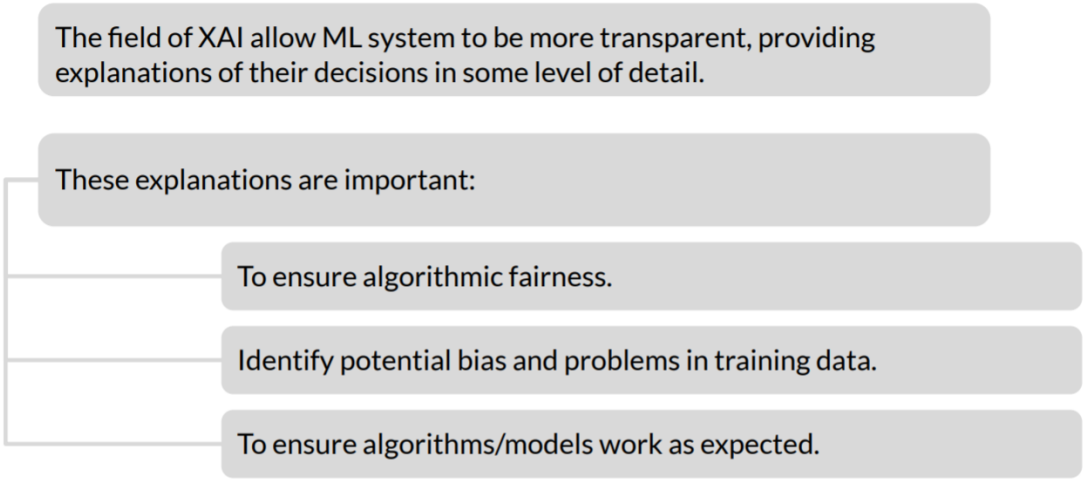
Week 5: Interpretability

Contents



- This week is all about interpretability, explaining to yourself and probably to other people why your model did what it did.
- It's increasingly important in a lot of production ML for a number of different reasons including fairness, regulatory requirements, and legal requirements, as well as just being able to understand your model better so that you can either fix problems with it or improve it.
- We'll start by discussing **explainable** AI.
- Interpretability is becoming increasingly important at the same time that it's becoming increasingly difficult with more and more complex models.
- But the good news is that the techniques for achieving interpretability are improving as well.
- Concretely, you could interpret that to mean that we're going to study interpretability.
- **Interpretability and explainability are parts of a larger field known as responsible AI**.
- The development of AI and the successful application of AI to more and more problems has resulted in the rapid growth of the ability to perform tasks which were previously not possible.
- This has created many great new opportunities.
- But at the same time, it also puts a lot of power into the results of models.
- There are sometimes questions about models and how responsibly they handle a number of factors which influence people and can cause harm.
- Issues of fairness, which we've talked about previously, are central to Responsible AI.
- Explainability, in a smaller sense, interpretability, are key to being able to do Responsible AI because we need to understand how models generated their results.
- Privacy is also part of responsible AI, since models often operate with Personally Identifiable Information or PII, and they are often trained with PII.
- Of course, security is also an issue and related to privacy, since one of the attacks that we've talked about is to pull the training data out of a model, which could mean pulling private information from a model.

Explainable Artificial Intelligence (XAI)



- The results generated by a model can be explained in different ways.
- One of the most advanced techniques is to create a model architecture that is **inherently explainable**.
- A simple example of this is a decision tree-based model, which by its nature is explainable.
- But they're increasingly more advanced and complex model architectures that are also designed to be inherently explainable.
- This is the field of Explainable Artificial Intelligence or XAI.
- Explainability is important in many ways. These include: ensuring fairness, looking for issues with bias in the training data, regulatory, legal, and branding concerns, and simply studying the internals of a model to optimize it to produce the best results.

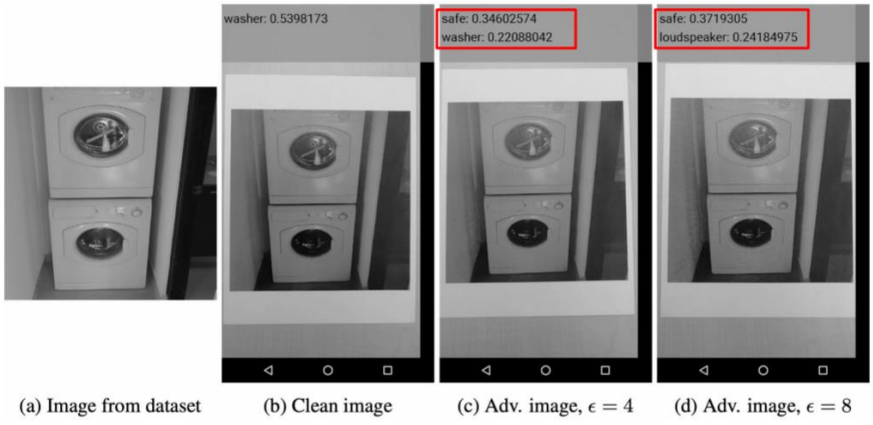
Need for Explainability in AI

1. Models with high sensitivity, including natural language networks, can generate wildly wrong results
2. Attacks
3. Fairness
4. Reputation and Branding
5. Legal and regulatory concerns
6. Customers and other stakeholders may question or challenge model decisions

- Why is explainability in AI so important? Well, fundamentally, it's because we need to explain the results and the decisions that are made by our models.
- This is especially true for models with high sensitivity, including natural language models, which when confronted with certain examples, can generate wildly wrong results.
- It also includes vulnerability to attacks which we need to evaluate on an ongoing basis and not just after an attack has already happened.

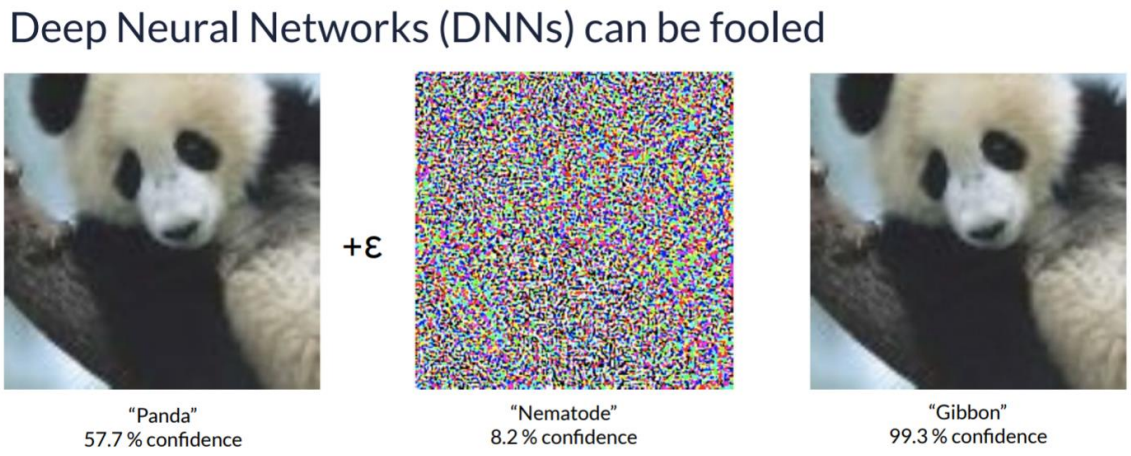
- Of course, fairness is a key issue, since we want to make sure that we're treating every user of our model fairly.
- This also impacts our reputation and branding, especially in cases where customers or other stakeholders may question or challenge our models decision.
- But really in any case, where we generate a prediction, and of course, there are legal and regulatory concerns, especially when someone is so unhappy that they challenge us and our model in court, or when our models result in an action that causes harm.

Deep Neural Networks (DNNs) can be fooled



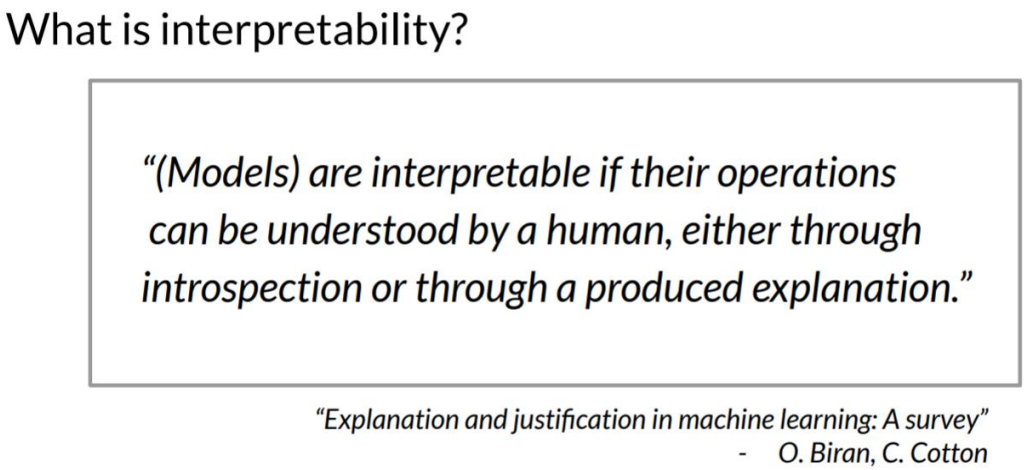
DNNs can be fooled into misclassifying inputs with no resemblance to the true category.

- Deep Neural Networks can be fooled into misclassifying inputs to produce results with no resemblance to the true category.
- This is easiest to see in examples of image classification, but fundamentally it can occur with **any** model architecture.
- The example on this slide demonstrates a black-box attack in which the attack is constructed without access to the model.
- The example is based on a phone app for image classification, using physical adversarial examples.
- What you see is a clean image of a washer from the dataset, the image A on the left, and that's used to generate adversarial images with various degrees of perturbation.
- The next was printing out the clean and adversarial images and then using the TensorFlow camera demo app to classify them.
- The clean image B is recognized correctly as a washer when perceived through the camera while increasing the adversarial perturbation in images C and D results in greater misclassification.
- The key result that is in image D, where the model thinks that the washer is either a safe or a loudspeaker, but definitely not a washer.
- Looking at the image, would you agree with the model? Maybe not.
- Can you see the adversarial perturbation that was applied? It's not that easy to see.



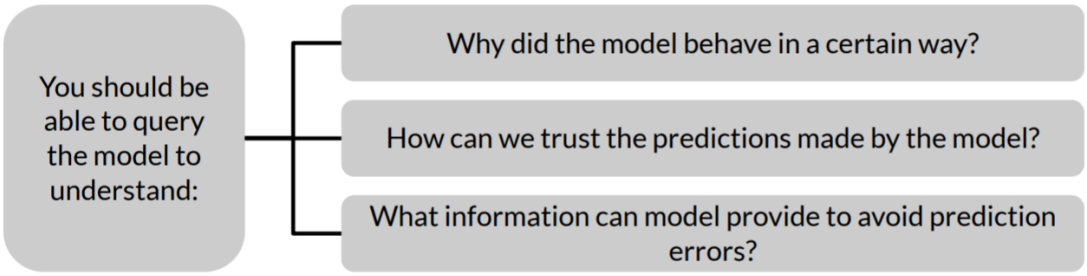
- This is perhaps the most famous example of this model attack by adding an **imperceptibly small amount of well-crafted noise**, an image of a panda can be misclassified as a gibbon, with 99.3 percent confidence.
- That's much higher than the original confidence that the model had, that it was a panda.

Model Interpretability Methods



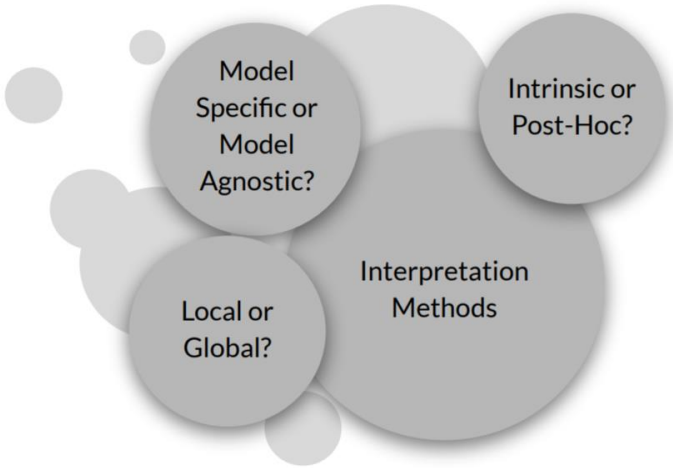
- Let's look now at some of the basic ways of interpreting models.
- There are **two** broad overlapping categories; techniques which can be applied to models in general, and model architectures which are inherently interpretable.
- Interpretability itself does not have a mathematical definition.
- Biran and Cotton provided one good definition of interpretability. They wrote that in systems, or in this case models, they are interpretable if their operations can be understood by a human either through **introspection** or through a **produced explanation**.
- In other words, if there's some way for a human to figure out why a model produced a certain result, then the model is interpretable.
- Practically speaking, however, the level of effort required needs to be feasible as well.
- One measure of interpretability of models is the amount of effort or analysis required to understand a given result.

What are the requirements?

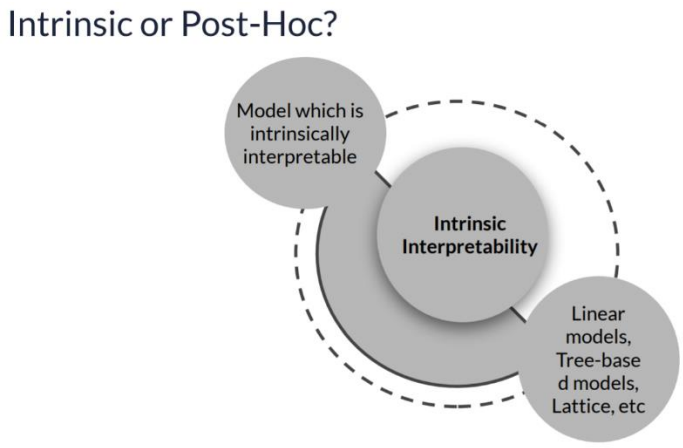


- Ideally, you should be able to query the model to understand the **what**, **why**, and **how** of its algorithmic decisions.
- **Why** did the model behave in a certain way?
- You should be able to identify and validate the relevant variables driving the model's outputs.
- Doing so will help you develop trust in the reliability of the predictive system even in unforeseen circumstances.
- This diagnosis will help ensure accountability and confidence in the safety of the model.
- **How** can we trust the predictions made by the model?
- Well, you should be able to validate any given data point to demonstrate to business stakeholders and peers that the model works as expected.
- This will help ensure the transparency of the model.
- **What** information can the model provide to avoid prediction errors?
- You should be able to query and understand latent variable interactions in order to evaluate and understand in a timely manner what features are driving predictions.
- This will help you to ensure the fairness of the model.

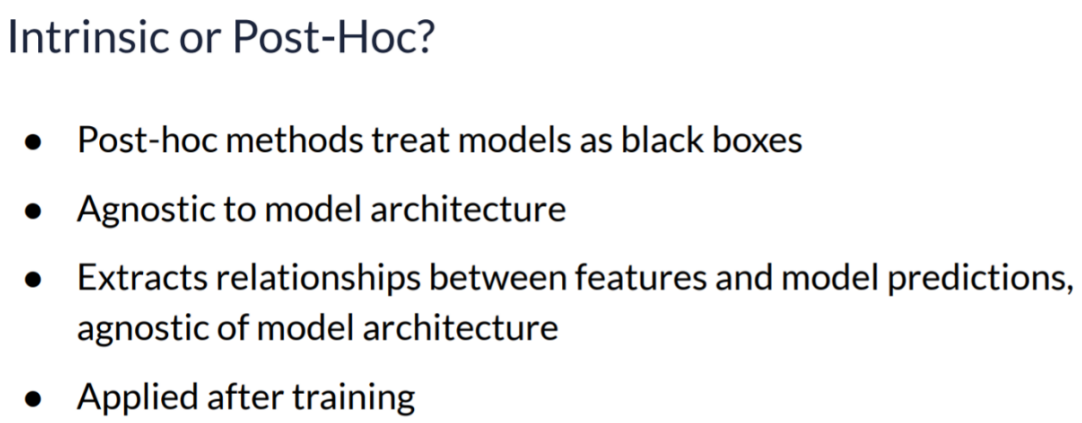
Categorizing Model Interpretation Methods



- There are some criteria that can be used for categorizing model interpretation methods.
- For example, interpretability methods can be grouped based on whether they're **intrinsic or post-hoc**.
- They could also be **model-specific** or **model agnostic**.
- They can also be grouped according to whether they are **local** or **global**.
- Let's discuss each of these criteria.



- One way of grouping model interpretability methods is by whether the model itself is intrinsically interpretable.
- Model architectures that are intrinsically interpretable have been around for a long time.
- The classic examples of this are **linear models and tree-based models**.
- More recently, however, **more advanced model architectures such as lattice models** have been developed to enable both interpretability and a high degree of accuracy on complex modeling problems.
- **Lattice models**, for example, can match or in some cases exceed the accuracy of neural networks.



- Let's consider Post-Hoc methods.
- Post-hoc methods treat models as black boxes and often don't distinguish between different model architectures.
- They tend to treat all models the same, and are applied after training to try to examine particular results to understand what caused the model to generate them.
- There are some methods, however, especially for convolutional networks that do inspect the layers within the network to try to understand how results were generated.
- There is, however, always some level of uncertainty about whether the interpretation of the reasons for certain results is correct or not, since Post-hoc methods don't evaluate the actual sequence of operations that led to the generation of the results.
- In general, an intrinsically interpretable model provides a higher degree of certainty as to why it generated a particular result.
- Examples of these analyses include feature importance and partial dependency plots.

Types of results produced by Interpretation Methods

Feature Summary Statistics

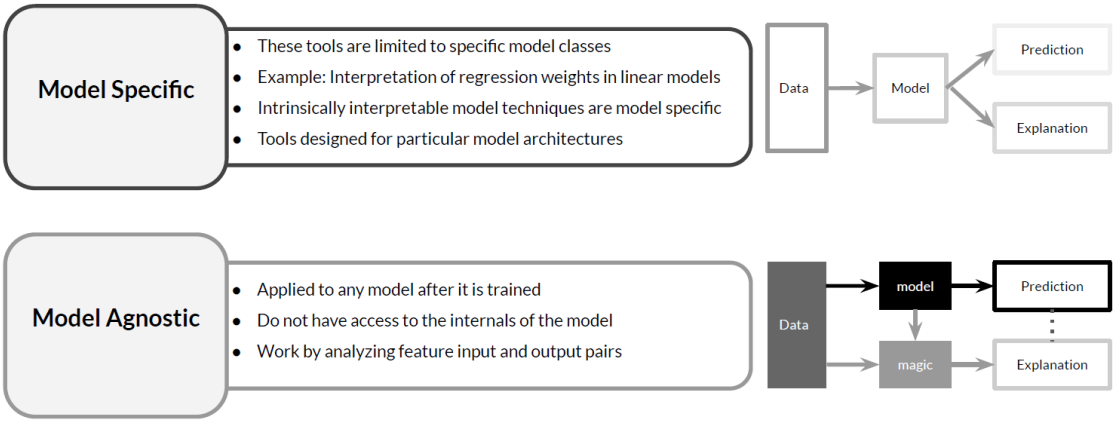
Feature Summary Visualization

Model Internals

Data point

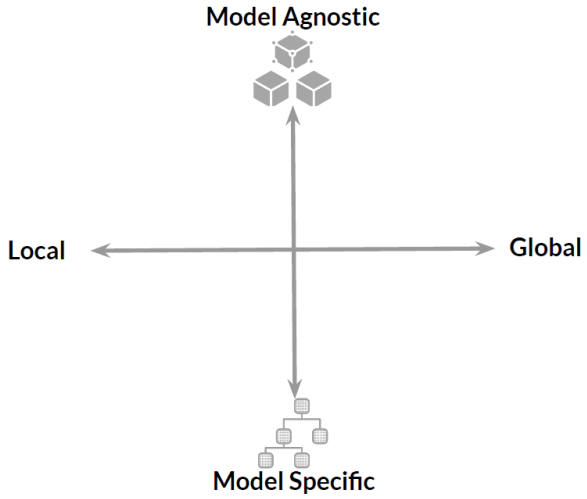
- The various interpretation methods can also be roughly classified according to the type of results they produce.
- Some methods create a summary of feature statistics. Some methods return a single value for a feature.
- For example, the feature importance returns a single number per feature.
- A more complex example would be **pairwise feature interaction strength**, which associates a number with each pair of features.
- Some methods rely on visualization to summarize features. For example, partial dependence plots.
- Partial dependence plots are curves that show a feature and its average predicted output.
- In this case, drawing the curve is more meaningful and intuitive than simply representing the values in a table.
- Some model-specific methods look at model internals.
- The interpretation of intrinsically interpretable models falls into this category.
- For example, in case of less complex models such as linear models, you can look at the learned weights to produce an interpretation.
- Similarly, the learned tree structure in a tree-based model serves as an interpretation.
- In lattice models, the parameters of each layer are the output of that layer, which makes it relatively easy to analyze, understand, and debug each part of the model.
- Some methods examine particular data points. One such method is **counterfactual** explanations.
- Counterfactual explanations are used to explain the prediction of a data point.
- In order to do so, it finds another data point by changing some features so that the predicted output changes in a relevant way.
- The change should be significant. For example, the new data point should be predicting a different class.

Model Specific or Model Agnostic



- Model-specific methods are limited to specific model types.
- For example, the interpretation of regression weights and linear models is model-specific.
- By definition, the techniques for interpreting intrinsically interpretable models are model-specific, but model-specific methods are not limited to intrinsically interpretable models.
- There are also tools that specifically focus on neural network interpretation.
- Model agnostic methods are not specific to any particular model.
- They can be applied to any model after it's trained. Essentially, they are Post-hoc methods.
- These methods **do not have access to the internals** of the model such as weights and parameters, etc.
- They usually work by analyzing feature input and output pairs and trying to infer relationships.

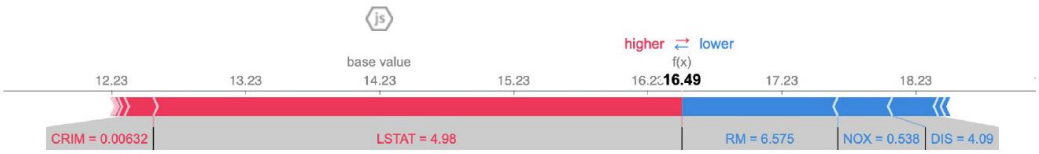
Interpretability of ML Models



- In addition to grouping interpretation methods as model agnostic or model specific, they can also be grouped by whether they generate interpretations which are local or global.

Local or Global?

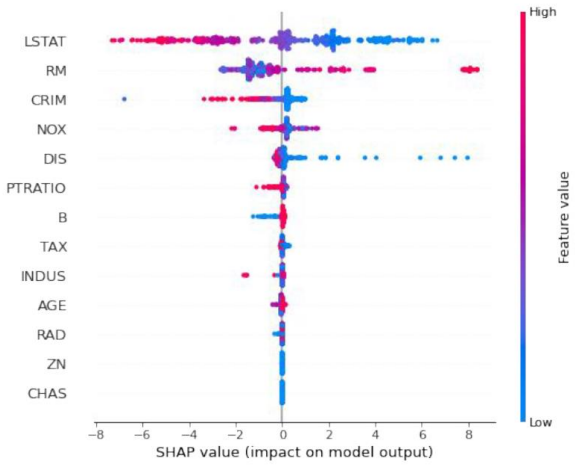
- Local: interpretation method explains an individual prediction.
- Feature attribution is identification of relevant features as an explanation for a model.



- Interpretability methods can be local or global based on whether the method explains an individual prediction or it explains the entire model behavior. Sometimes the scope can be in between local and global.
- A local interpretability method explains an individual prediction.
- For example, feature attribution in the prediction of a single example in the dataset.
- Feature attributions measure how much each feature contributed to the predictions for a given result.
- Here's a feature attribution using a library called SHAP for the prediction of a single example by a gradient boosted ensemble tree trained on the Boston house price dataset.
- This example is used to demonstrate what a **local explanation** looks like (from the SHAP library).
- The diagram shows the contribution of features in pushing model output from the base value towards the actual model output.
- Features in red push the model towards a higher value, and features in blue try to push the model output towards a lower value.

Local or Global?

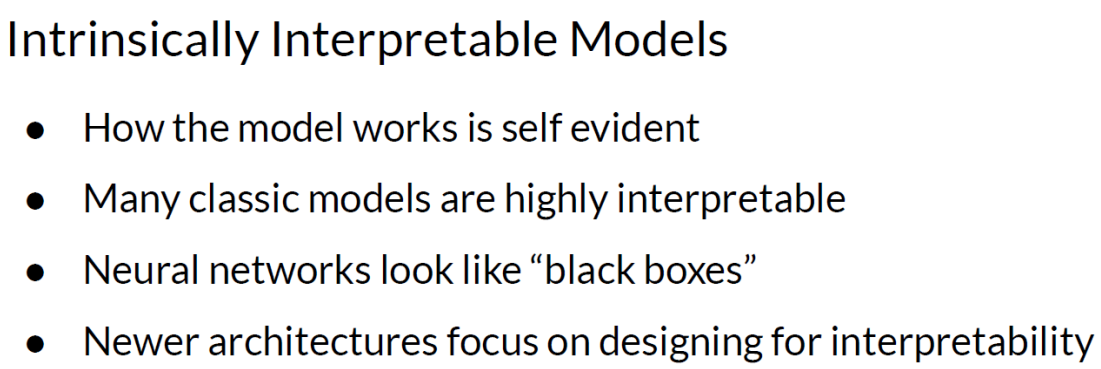
- Global: interpretation method explains entire model behaviour
- Feature attribution summary for the entire test data set



- Interpretability methods can also be global. Global methods explain the entire model behavior.
- For example, if the method creates a summary of feature attributions for predictions on the entire test set, then it can be considered global.
- Here's an example of a **global explanation** created by the **SHAP** library.
- It shows feature attributions, the SHAP value of every feature for every sample for predictions in the Boston house prices dataset.

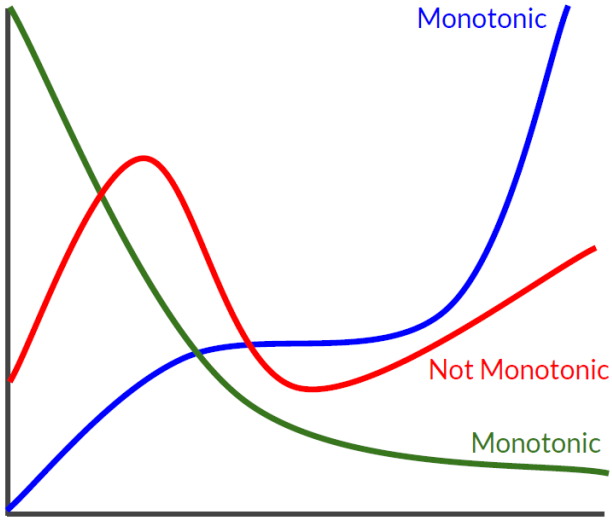
- The color represents the feature value. Red is high and blue is low.
- You can see how as LSTAT, the percent of lower status of the population, increases, it tends to lead to a decrease in the predicted house price.
- Since this explanation shows an overview of attributions of all features on all instances in the dataset, it would be considered global.

Intrinsically Interpretable Models



- Since the early days of statistical analysis and Machine Learning, there had been model architectures which are intrinsically interpretable.
- Let's look at those now along with some more recent advances, and learn about how they can improve interpretability.
- What exactly do we mean by an intrinsically interpretable model?
- One definition is that the workings of the model are transparent enough and intuitive enough that they make it relatively easy to understand how the model produced a particular result by examining the model itself.
- Many classic models are highly interpretable, such as **tree-based models and linear models**.
- But although we've seen neural networks that are able to produce really amazing results, one of the issues with them is that they tend to be very opaque, especially the larger and more complex architectures, which makes them black boxes when we're trying to interpret them.
- That limits our ability to interpret their results and requires us to use post-hoc analysis tools to try to understand how they reached a particular result.
- However, newer architectures have been created, which are designed specifically for interpretability and yet retain the power of deep neural networks. This continues to be an active field of research.

Monotonicity improves interpretability

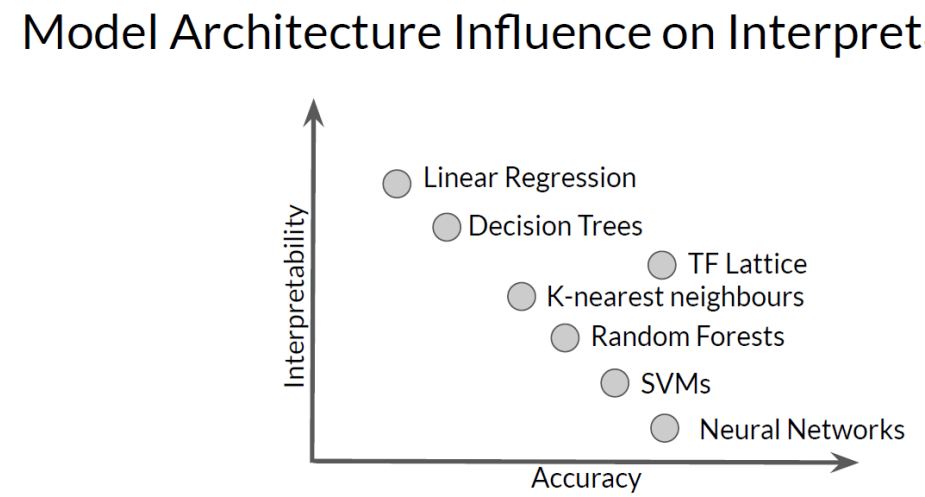


- One key characteristic which helps improve interpretability is when features are monotonic.
- Monotonic means that the contribution of the feature towards the model result either consistently increases or decreases or stays even as the feature value changes.
- The key factor here is the **consistency**.
- This matches the domain knowledge for many features in many problems.
- When you're trying to understand the model results, if features are monotonic, it matches your intuition about the reality of the world that you're trying to model.
- For example, if you're trying to create a model to predict the value of a used car, when all other features are held constant, the more miles on the car, the less the value should be.
- But you don't expect a car with more miles to be worth more than it was with less miles, all other things being equal.
- This matches your knowledge of the world and, so your model should match it too and the mileage feature should be monotonic.
- In this graph, the blue and green curves are monotonic, while the red curve is not because it does not consistently increase or decrease or remain the same.

Interpretable Models

Algorithm	Linear	Monotonic	Feature Interaction	Task
Linear regression	Yes	Yes	No	regr
Logistic regression	No	Yes	No	class
Decision trees	No	Some	Yes	class, regr
RuleFit	Yes*	No	Yes	class, regr
K-nearest neighbors	No	No	No	class, regr
TF Lattice	Yes*	Yes	Yes	class, regr

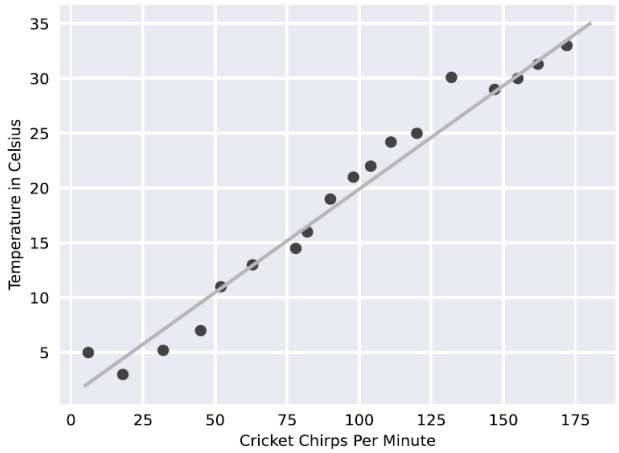
- Let's look at a few architectures which are considered interpretable.
- First, linear models are very interpretable because linear relationships are easy to understand and interpret, and **the features of linear models are always monotonic**.
- Some other model architectures have linear aspects to them. For example, when used for regression, RuleFit Models are linear.
- In all cases, TF Lattice Models use linear interpolation between lattice points, which we'll learn about soon.
- Some models can automatically include feature interactions or include constraints on feature interactions.
- In theory, you can include feature interaction in all models through feature engineering.
- Interactions which match domain knowledge tend to make models more interpretable.



Interpretability vs Accuracy Trade off

- Depending on the characteristics of the loss surface that you're trying to model, the more complex model architectures can achieve high-accuracy.
- This often comes at a price in terms of interpretability.
- For many of the reasons discussed earlier, interpretability can be a strict requirement of models, so you need to find a balance between models that you can interpret, and models that generate the accuracy that you need.
- Again, some newer architectures have been created which deliver far greater accuracy but also deliver good interpretability.
- Tensorflow Lattice is one example of these kinds of architectures.

Classics: Linear Regression



- Probably the ultimate in interpretability is our old friend, linear regression.
- I don't know about you, but for me, this was the first type of model that I learned about.
- It is very easy to understand the relationship between feature contributions, even for a multivariate linear regression.
- As feature values increase or decrease, their contribution to the model results also increases or decreases.
- This example shown here, models the number of chirps per minute that a cricket will make based on the temperature of the air.
- This is a very simple linear relationship and so linear regression models it well.
- By the way, this also means that when you're out at night, if you listen carefully to the crickets and count how many chirps they make, you can measure the air temperature. Check out Dolbear's Law to learn more.

Interpretation from Weights

Linear models have easy to understand interpretation from weights

- Numerical features: Increase of one unit in a feature increases prediction by the value of corresponding weight.
- Binary features: Changing between 0 or 1 category changes the prediction by value of the feature's weight.
- Categorical features: one hot encoding affects only one weight.

- Of course, the actual contribution of a feature to the results of the model will depend on its weight.
- This is especially easy to see for linear models.
- For numerical features, an increase or decrease of one unit in a feature increases or decreases the prediction based on the value of the corresponding weight.
- For binary features, prediction is increased or decreased by the value of the weight, based on whether the features value is one or zero.

- Categorical features are actually divided up into several individual features with one-hot encoding, each of which has a weight.
- With one-hot encoding, only one of the categories will be set, so only one of the weights will be included in the model result.

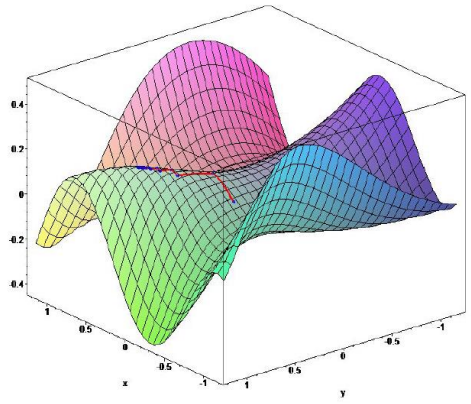
Feature Importance

- Relevance of a given feature to generate model results
- Calculation is model dependent
- Example: linear regression model, t-statistic

- How can we determine the relevance of a given feature for making predictions?
- Feature importance tells us how important the feature is for generating a model result.
- The more important feature is, the more we want to include it in our feature vector.
- But feature importance for different models is calculated differently because different models calculate results differently.
- For linear regression models, the **absolute value of a feature's t-statistic** is a good measure of that feature's importance.
- **The t-statistic is the learned or estimated weight of the feature scaled by its standard error.**
- The importance of a feature increases as its weight increases.
- But the more variance the weight has; in other words, the less certain we are about the correct value of the weight, the less important the feature is.

More advanced models: TensorFlow Lattice

- Overlaps a grid onto the feature space and learns values for the output at the vertices of the grid
- Linearly interpolates from the lattice values surrounding a point

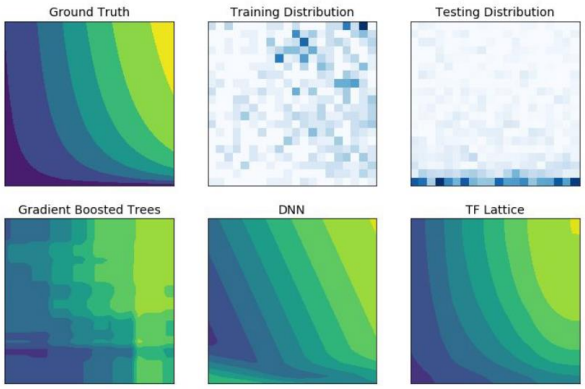


- You may not be familiar with lattice models. Let's take a look at how they work.
- Lattice models overlay a grid onto a feature space and set the values of the function that it's trying to learn at each of the **vertices** (aka edges) of the grid.
- As prediction requests come in, if they don't fall directly on a vertex, then the result is **interpolated** using **linear interpolation** from the nearest vertices of the grid.

- One of the benefits is to regularize the model and greatly reduce sensitivity, even to examples that are outside the coverage of the training data, by imposing a regular grid on under the feature space.

More advanced models: TensorFlow Lattice

- Enables you to **inject domain knowledge** into the learning process through common-sense or policy-driven shape constraints
- Set constraints such as monotonicity, convexity, and how features interact



- However, Tensor-Flow lattice models go beyond simple lattice models.
- TensorFlow lattice further allows you to **add constraints and inject domain knowledge** into the model.
- These graphs show the benefit of regularization and domain knowledge.
- Compare the one on the top left to the one on the bottom right, and notice how close the model is to the ground truth compared to other kinds of models.
- When you know that certain features in your domain are monotonic, or convex, or that one or more features interact, you can inject that knowledge into the model as it learns.
- For interpretability, this means that feature values and results are likely to match your domain knowledge, for what you expect your results to look like.
- You could also express relationships or interactions between features to suggest that one feature reflects trust in another feature.
- For example, a higher number of reviews makes you more confident in the average star rating of a restaurant. You might have considered that yourself when shopping online.
- All of these constraints based on your domain knowledge or what you know about the world that you're trying to model, help the model produce results that make sense, which helps make them interpretable.
- Also, **since the model uses linear interpolation between vertices, it also has many of the benefits of linear models in terms of interpretability**.

TensorFlow Lattice: Accuracy

Accuracy

- TensorFlow Lattice achieves accuracies comparable to neural networks
- TensorFlow Lattice provides greater interpretability

- But along with all the benefits of added constraints based on domain knowledge, TensorFlow lattice models also have a level of accuracy on complex problems that is similar to deep neural networks.
- Of course, TensorFlow lattice models are also easier to interpret than neural networks.

TensorFlow Lattice: Issues

Dimensionality

- The number of parameters of a lattice layer **increases exponentially** with the number of input features
- Very Rough Rule: Less than 20 features ok without ensembling

- However, lattice models do have a weakness. **Dimensionality** is their kryptonite.
- The number of parameters of a lattice increases exponentially with the number of input features, which creates problems with scaling for datasets with a large number of features.
- As a rough rule of thumb, you're probably okay with **20 or less features**.
- But this will also depend on the number of vertices that you specify.
- There is another way to deal with this dimensionality kryptonite however, and that's by using ensembling, but that is beyond the scope of this presentation.

Model Agnostic Methods

Model Agnostic Methods

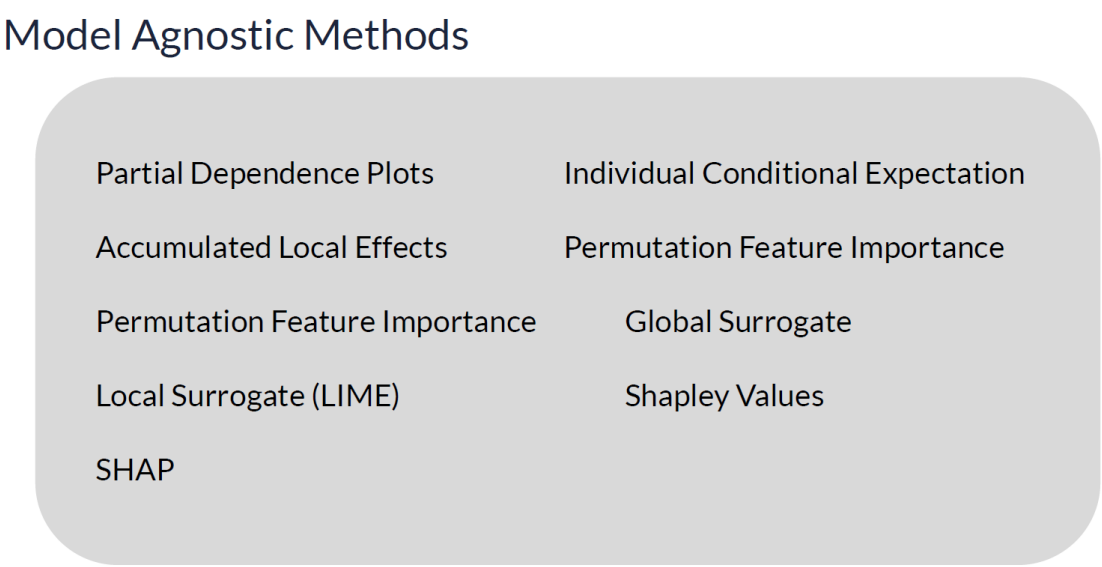
These methods separate explanations from the machine learning model.

Desired characteristics:

- Model flexibility
- Explanation flexibility
- Representation flexibility

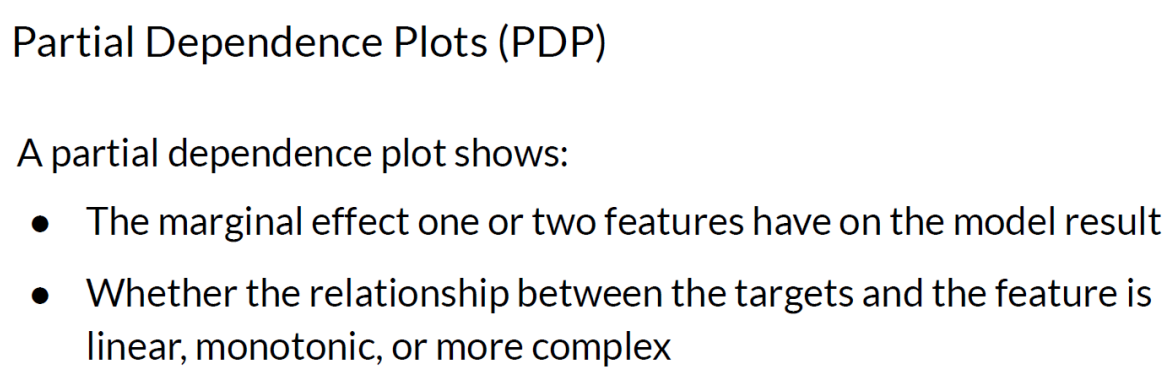
- Unfortunately, you can't always work with models that are intrinsically interpretable.
- For a variety of reasons you may be asked to try to interpret the results of models that are not inherently easy to interpret.
- Fortunately, there are several methods available that are not specific to particular types of models.
- In other words, they are model agnostic. Let's look at some of those now.
- Model agnostic methods **separate the explanations from the model**.
- These methods can be applied to any model after it's been trained.
- For example, they can be applied to linear regression or decision trees or even black box models like neural networks.

- The desirable characteristics of a model agnostic method include model flexibility and explanation flexibility.
- The explanation shouldn't be limited to a certain type.
- The method should be able to provide an explanation as a formula or in some explanations it can be graphical like perhaps for feature importance.
- These methods also need to have representation flexibility.
- The feature representations used should make sense in the context of the model being explained.
- Let's take the example of a text classifier that uses word embeddings.
- It would make sense for the presence of individual words to be used in the explanation in this case.



- There are many model agnostic methods that are currently being used, too many to go into all of them in detail here.
- I'll just discuss a few to give you a flavor for the kinds of things that you can do with model agnostic methods to explain the results generated by your models

Partial Dependence Plots



- One widely used method is partial dependence plots or PDP. Let's start with PDP.
- Partial dependence plots help you to understand the effects that particular features have on the model results that you're seeing, and the type of relationship between those features, and the targets or labels in your training data.

- It typically concentrates on the **marginal impact** caused by one or two features on the model results.
- Those relationships could be linear and or monotonic, or they could be of some more complex type.
- For example, for a linear regression model, a partial dependence plot will always show a linear monotonic relationship.
- **PDP is a global method** since it considers all instances and evaluates the global relationship between the features and the results.

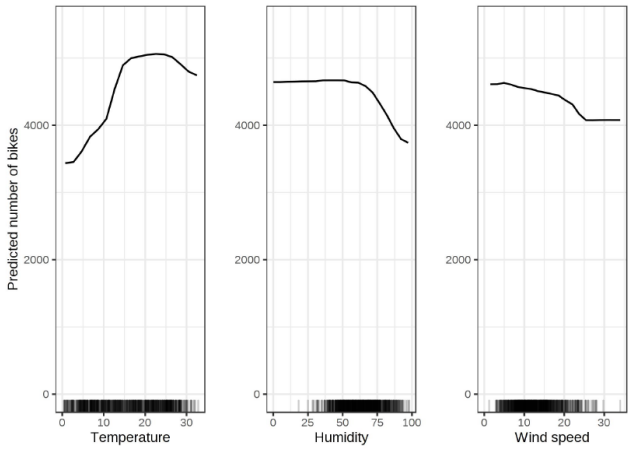
Partial Dependence Plots

The partial function f_{xs} is estimated by calculating averages in the training data:

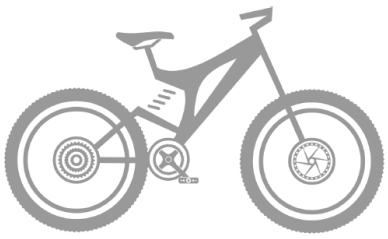
$$\hat{f}_{x_S}(x_S) = \frac{1}{n}\sum_{i=1}^{n}\hat{f}(x_S, x_C^{(i)})$$

- The partial function $\mathbf{f_{xs}}$ is estimated by calculating averages in the training data, also known as the Monte Carlo method.
- This equation shows the estimation of the partial function where ***n*** is the number of examples in the training data set, ***s*** is the features that we're interested in, and **c** is all of the other features.
- **The partial function tells us what the average marginal effect on the results is, for given values of the features in s**.
- In this formula, $\mathbf{x_c^i}$ are feature values for the features that we're not interested in.
- **PDP makes the assumption** that the features in ***c*** are **not correlated** with the **features** in ***s***.
- If this assumption is violated, the averages calculated will include data points that are very unlikely or even impossible.

Partial Dependence Plots: Examples



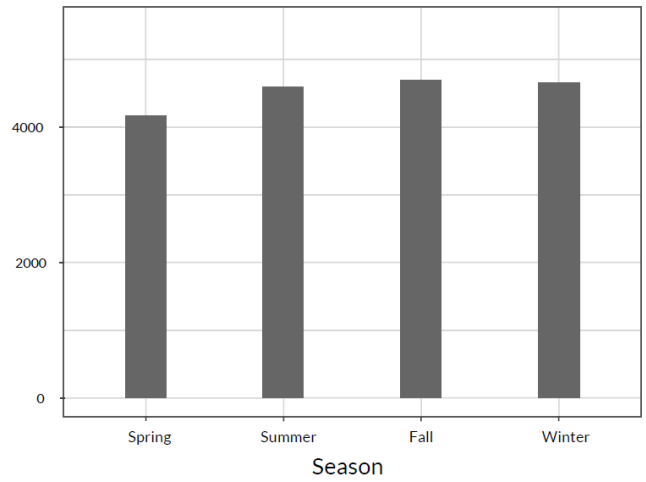
PDP plots for a linear regression model trained on a bike rentals dataset to predict the number of bikes rented



- Here's an example of a random forest model trained on a bike rentals data set, to predict the number of bikes rented per day, given a set of features which include temperature, humidity and wind speed.
- These are the partial dependence plots for temperature, humidity and wind speed.

- Notice that as the temperature increases up to about 15 degrees Celsius, more people are likely to rent a bike.
- This makes sense because people like to ride bikes when the weather is nice, and at that temperature I'd say, it's just starting to get nice.
- But notice that this trend level is off and then starts to fall off between above about 25 degrees Celsius.
- You can also see that humidity is a factor and above about 60% humidity, people start to get less interested in riding bikes.
- How about you, do these plots match your bike riding preferences?

PDP for Categorical Features



- To calculate a PDP for categorical features, we **force all instances to have the same category value**.
- Here's the plot for the categorical features 'season' in the bike rentals dataset.
- It has four possible values, spring, summer, fall and winter.
- In order to calculate the PDP for summer, we force all instances in the data set to have value equals summer for the feature 'season'.
- This plot shows different values for season. Notice that there is not much difference of an effect if a change in seasons on bike rentals, except for in spring when the number of rentals is somewhat lower.
- Frankly, I wouldn't expect that to be the case, but that's what the data is telling us.

Advantages of PDP

- Computation is intuitive
- If the feature whose PDP is calculated has no feature correlations, PDP perfectly represents how feature influences the prediction on average
- Easy to implement

- There are some clear advantages to using PDP first.
- The results tend to be intuitive, especially when the features are **not** correlated.
- When they're not correlated, a PDP's plot shows the average prediction changes when a feature has changed.

- The interpretation of a PDP is also usually causal, in the sense that if we change a feature and measure the changes in the results, we expect the results to be consistent.
- Finally, PDP is fairly easy to implement with some packages listed in the reference section at the end of this lesson.

Disadvantages of PDP

- Realistic maximum number of features in PDP is 2
- PDP assumes that feature values have no interactions

- Like most things, however, there are some disadvantages to PDP.
- Realistically, you can **only really work with two features at a time**, because humans have a hard time visualizing more than three dimensions.
- But I'm really not sure that I would blame PDP for that.
- More serious limitation is the **assumption of independence**.
- PDP assumes that the features that you're analysing aren't correlated with other features.
- As we learned in our discussion of feature selection, it's a good idea to eliminate correlated features anyway.
- But, if you still have correlated features, PDP doesn't work quite right.
- For example, suppose you want to predict how fast a person walks given the person's height and weight.
- PDP will assume that height and weight aren't correlated, which is obviously a false assumption.
- As a result, we might include people with a height of two meters and a weight of 50 kg, which is a bit unrealistic even for fashion models, although when I Googled this, I was shocked to learn that some are actually pretty close.
- But anyway, you get the idea, correlated features are bad

Permutation Feature Importance

Permutation Feature Importance

Feature importance measures the increase in prediction error after permuting the features

Feature is **important** if:

- Shuffling its values increases model error

Feature is **unimportant** if:

- Shuffling its values leaves model error unchanged

- Now let's take a look at permutation feature importance.
- Permutation feature importance is just one way of measuring the importance of a feature.

- Permuting a feature is just one way of breaking the relationship between a feature and the model result, essentially by assigning a nearly random value to the feature.
- For permutation feature importance, we measure the importance of a feature by measuring the **increase in the prediction error after permuting that feature**.
- A feature is important if shuffling its values increases the model error, because in this case, **the model relied on the feature for the prediction.**
- A feature is unimportant if shuffling its values leaves the model error unchanged, because in this case, the model ignored the feature.
- If we find that we have unimportant features, then we should really consider removing them from our feature vector.

Permutation Feature Importance

- Estimate the original model error
- For each feature:
  - Permute the feature values in the data to break its association with the true outcome
  - Estimate error based on the predictions of the permuted data
  - Calculate permutation feature importance
  - Sort features by descending feature importance

- This is the basic algorithm.
- The inputs are the model, the features, the labels or targets, and our error metric.
- You start by measuring the model error with all of the true values of the features.
- Next, you started **iterative process** for each feature where you first **permute the values of the feature** that you're examining, and **measure the change in the model error**.
- You can either express the feature importance as a ratio of the permuted error to the original error, or as the difference between the two.
- You can then sort by the feature importance to determine the least important features.

Advantages of Permutation Feature Importance

- Nice interpretation: Shows the increase in model error when the feature's information is destroyed.
- Provides global insight to model's behaviour
- Does not require retraining of model

- Let's consider the advantages of permutation feature importance.
- Permutation feature importance has a nice interpretation because feature importance is the increase in the model error when the feature's information is destroyed.

- It's a highly compressed global insight into the model's behavior
- Since by permuting the feature, you also destroy the interaction effects with other features, it also shows the interactions between features.
- This means that it accounts for both the main feature effects and the interaction effects on model performance.
- A big advantage is that it **doesn't require retraining the model**.
- Some other methods suggest deleting a feature, retraining the model, and then comparing the model error.
- Since retraining a model that can take a long time, not requiring that is a big advantage.

Disadvantages of Permutation Feature Importance

- It is unclear if testing or training data should be used for visualization
- Can be biased since it can create unlikely feature combinations in case of strongly correlated features
- You need access to the labeled data

- Let's look at some of the disadvantages inherent to permutation feature importance.
- One odd disadvantage is that it's unclear whether you should use your training data or your test data to measure permutation feature importance.
- There are pluses and minuses to both.
- Like PDP, correlated features are once again a problem.
- You also need to have access to the original labeled training data set.
- So if you're getting the model from someone else and they don't give you that, then you can't use permutation feature importance.

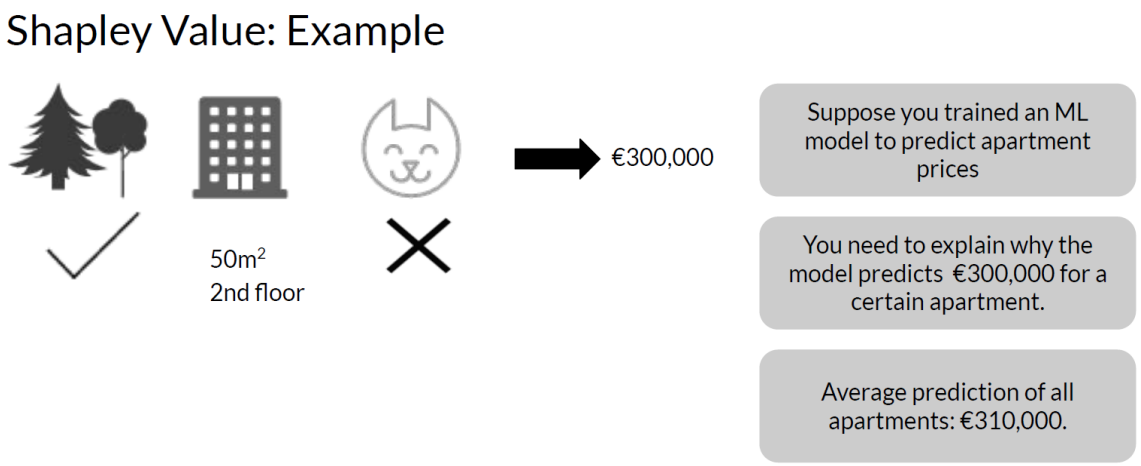
Shapley Values

Shapley Value

- The Shapley value is a method for assigning payouts to players depending on their contribution to the total
- Applying that to ML we define that:
  - Feature is a “player” in a game
  - Prediction is the “payout”
  - Shapley value tells us how the “payout” (feature contribution) can be distributed among features

- A key concept in measuring feature importance and the contribution of each feature to the models results is the Shapley value. Let's discuss that now.

- The Shapley value is a concept from Cooperative Game Theory.
- It was named in honor of Lloyd Shapley who introduced it in 1951 and won the Nobel Prize in Economics for it in 2012.
- To each cooperative game, it assigns a unique distribution among the players of the total surplus value generated by the coalition of all players.
- Here's how it works in game theory.
- Imagine that a group of players cooperates and that results in an overall gain because of their cooperation.
- Since some players may contribute more than others, or may possess different bargaining power, how should we distribute the gains among the players?
- Or phrased differently, how important is each player to the overall cooperation, and what payoff can he or she reasonably expect?
- The Shapley value provides one possible answer to this question.
- So how do you apply this to ideas in ML?
- For machine learning and interpretability, the players are the features of the data set, and we're using the Shapley value to determine how much each feature contributes to the results.
- As you might expect the prediction is the payout.
- Knowing how the features contribute will help you understand what the important factors were in generating the models results.
- Because it's not specific to any particular type of model, it can be used regardless of the model architecture.



- That was a quick overview of the ideas behind Shapley value.
- Let's now focus on a concrete example. Suppose you trained an ML model to predict apartment prices.
- You need to explain why the model predicts a €300,000 price for a certain apartment.
- What data do you have to work with?
- Well, in this example the apartment is 50 square meters. It's located on the second floor, it has a park nearby, and cats are banned.
- The average prediction for all apartments is €310,000.

Shapley Value

Term in Game Theory	Relation to ML	Relation to House Prices Example
Game	Prediction task for single instance of dataset	Prediction of house prices for a single instance
Gain	Actual prediction for instance - Average prediction for all instances	Prediction for house price (€300,000) - Average Prediction(€310,000) = -€10,000
Players	Feature values that contribute to prediction	'Park=nearby', 'cat=banned', 'area=50m^2', 'floor=2nd'

- Shapley values come from game theory, so let's clarify how to apply them to machine learning interpretability.
- The game is the prediction task for a **single instance** of the data set.
- The gain is the actual prediction for this instance minus the average prediction for all instances.
- The players are the feature values of the instance that collaborate to produce the gain
- In the apartment example, the feature values park equals nearby, cat equals banned, area equals 50 square meter, and floor equals 2nd, work together to achieve the prediction of €300,000.

Shapley Value

Goal :
Explain the difference between the actual prediction (€300,000) and the average prediction (€310,000): a difference of -€10,000.

Feature	Contribution
'park-nearby'	€30,000
size-50	€10,000
floor-2nd	€0
cat-banned	-€50,000
Total: -€10,000 (Final prediction - Average Prediction)	

One possible explanation

- Our goal is to explain the difference between the actual prediction €300,000 and the average prediction of €310,000, which is the loss of €10,000.
- One possible explanation could be the park nearby contributed €30,000, size of 50 square meter contributed €10,000, floor equals 2nd contributed zero, cat equals banned contributed negative €50,000.
- The contributions add up to negative €10,000, which is the final prediction minus the average predicted apartment price.
- You can think of that as the absolute value €10,000, or you could also think about it as the percentage of the average which is about 3.3%.
- This is one possible explanation, but how did we get these numbers?

Advantages of Shapley Values

Based on solid theoretical foundation.
Satisfies Efficiency, Symmetry, Dummy, and Additivity properties

Value is fairly distributed among all features

Enables contrastive explanations

- Unlike perhaps any other method of interpreting model results, Shapley values are based on a solid theoretical foundation.
- Other methods make intuitive sense, which is an important factor for interpretability, but don't have the same **rigorous theoretical foundation**.
- This is one of the reasons that Shapley was awarded the Nobel Prize for his work.
- The theory defines four properties which must be satisfied: Efficiency, Symmetry, Dummy, and Additivity.
- One key advantage of Shapley values that they're fairly distributed among the feature values of an instance.
- Some have argued that Shapley might be the only method to deliver a full explanation
- In situations where the law requires explainability, like the **EU right to explanation** for example, some feel that **Shapley value might be the only legally compliant method** because it's based on a solid theory and distributes the effects fairly.
- The Shapley value also allows contrastive explanations.
- Instead of comparing a prediction to the average prediction of the entire data set, you could compare it to a subset or even to a single data point.
- This **ability to contrast is something that local models like Lime don't have**.

Disadvantages of Shapley Values

- Computationally expensive
- Can be easily misinterpreted
- Always uses all the features, so not good for explanations of only a few features.
- No prediction model. Can’t be used for “what if” hypothesis testing.
- Does not work well when features are correlated

- Like any method Shapley has some disadvantages.
- Probably the most important is that it’s computationally expensive, which in a large percentage of real world cases means that it's only feasible to calculate at approximate solution.
- It can also be easily misinterpreted.

- **The Shapley value is not the difference of the predicted value after removing the feature from the model training**
- **Rather, it is the contribution of a feature value to the difference between the actual prediction and the mean prediction**.
- If you want to only explain a few of your features, Shapley is probably the wrong method to use, because Shapley always uses all the features.
- Humans prefer **selective explanations** such as those produced by **Lime** and other similar methods, so those might be the better choice for explanations that lay persons have to deal with.
- Or another possible solution is to use SHAP which is based on the Shapley value but can provide explanations with only a few features. We'll discuss SHAP next.
- Unlike some other methods, Shapley does not create a model.
- This means you can't use it to test changes in the inputs, such as if I change to a 200 square meter apartment, how does it change the prediction?
- And finally, like many other methods, it does not work well when the features are correlated, but you already know that you should have removed correlated features from your feature vector when you were doing feature selection.
- So that's not a problem for you, right? Well, hopefully anyway, but something to be aware of.

SHapley Additive exPlanations (SHAP)

SHAP

- SHAP (SHapley Additive exPlanations) is a framework for Shapley Values which assigns each feature an importance value for a particular prediction
- Includes extensions for:
  - TreeExplainer: high-speed exact algorithm for tree ensembles
  - DeepExplainer: high-speed approximation algorithm for SHAP values in deep learning models
  - GradientExplainer: combines ideas from Integrated Gradients, SHAP, and SmoothGrad into a single expected value equation
  - KernelExplainer: uses a specially-weighted local linear regression to estimate SHAP values for any model

- Now let's take a look at the open source SHAP library, which is a powerful tool for working with Shapley values and other similar measures.
- SHAP, which is short for Shapley additive explanations, is a game theoretic approach to explain the output of any machine learning model, which makes it **model agnostic**.
- It connects optimal credit allocation with local explanations using the classic Shapley values from game theory and their related extensions, which have been the subject of several recent papers.
- Remember that Shapley created his initial theory in 1951, and more recent researchers have been extending his work.
- It assigns each feature and importance value for a particular prediction and includes some very useful **extensions**, many of which are based on this recent theoretical work.

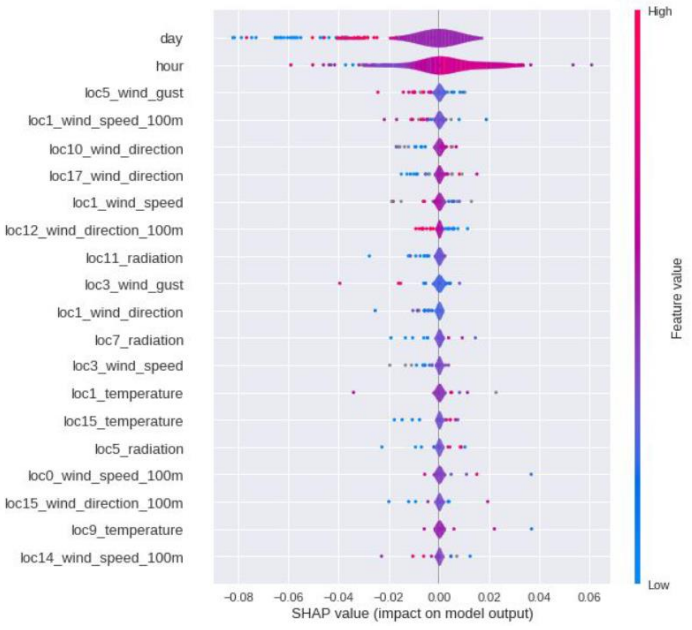
- These include **tree explainer**, a high speed exact algorithm for tree ensembles, **deep explainer**, a high speed approximation algorithm for Shap values in deep learning models, **gradient explainer**, which combines ideas from integrated gradients, Shap, and smooth grad into a single expected value equation, and **kernel explainer**, which uses a specially weighted local linear regression to estimate Shap values for any model.
- It also includes several plots to visualize the results, which helps you to interpret the model.

SHAP Explanation Force Plots

- Shapley Values can be visualized as forces
- Prediction starts from the baseline (Average of all predictions)
- Each feature value is a force that increases (red) or decreases (blue) the prediction

- You can visualize Shapley values as forces.
- **Each feature value is a force** that either increases or decreases the prediction.
- The prediction starts from the baseline, which for Shapley values is the average of all the predictions.
- In a force plot, each Shapley value is displayed as an arrow that pushes the prediction to increase (positive values shown here in red) or decrease (negative values shown here in blue).
- These forces meet at the prediction to balance each other out.

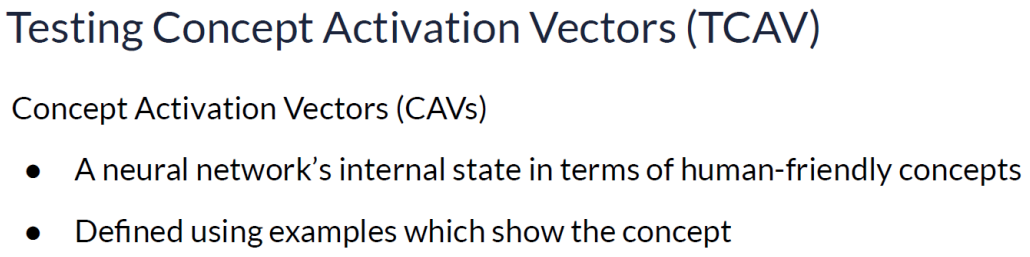
SHAP Summary Plot



- **A summary plot combines feature importance with feature effects**.
- Each point on the summary plot is a **Shapley value for a feature and an instance.**
- The color represents the value of the feature from low (blue) to high (red)
- Overlapping points are jittered in the y axis direction, so we get a sense of the distribution of Shapley values per feature, and features are ordered according to their importance.
- So in this example we can quickly see that the **two most important features** are day and hour

- In a SHAP dependence plot, a feature value is plotted on the x-axis and the SHAP value plotted on the y-axis.
- From the plot in this example, you can see that the correlation between hour and the loc_15 temperature feature is low.
- In this example, in the early hours of the morning, right after midnight, the energy price drops during the afternoon hours between 10 and 12, the price increases, and then drops back down after three. Then the price again increases at night after 8 O'clock.

Testing Concept Activation Vectors (TCAV)



- Let's get started with an advanced approach known as testing concept activation vectors.
- The interpretation of deep learning models is a challenge due to their size, complexity and often opaque internal state.
- In addition, many systems, such as image classifiers operate on **low level features** rather than high level concepts.

- To address these challenges, the team at Google introduced **concept activation vectors**, or **CAVs**, which provide an interpretation of a neural networks internal state, in terms of human friendly concepts
- High level human friendly concepts are defined using sets of example input data for the model under inspection
- For instance, to define the concept *curly*, a set of images of hairstyles and texture of images can be used for an image model.
- Note that these examples are not constrained to training data.
- They can be defined using new user provided data
- Examples have been shown to be effective means of interfacing with models for both expert and non-expert users.

Example Concepts

- You can use concept activation vectors to sort examples or in this case images with respect to their relationship to a concept.
- This is useful for qualitative confirmation that the concept activation vectors correctly reflect the concept of interest.
- As a concept activation vector encodes the **direction of a concept in the vector space** of a bottleneck, we can compute cosine similarity between a set of pictures of interest to the concept activation vector in order to sort the pictures.
- Note that the pictures being sorted are not used to train the concept activation vector.
- Shown here are two concepts of interest: (i) CEO and (ii) model women, and the images that have been sorted by similarity to the concept.
- On the left are sorted images of stripes with respect to a concept activation vector learned from a more abstract concept, CEO, collected from image net.
- The top three images are the most similar to the CEO concept and look like pinstripes, which may relate to the ties or suits that a CEO may wear, which provides confirmation of the **idea that CEOs are more likely to wear pinstripes than horizontal stripes**.
- I'm told that horizontal stripes can look unprofessional, but I'm hardly a fashion expert, which you can tell by the way I dress.
- On the right are sorted images of neckties with respect to a 'model women' concept activation vector.
- Again, the top three images are the most similar to this concept, which in this case is model women and show women in neckties, but out of the bottom three images, show men in neckties.
- This also suggests that concept activation vectors can be a standalone similarity sorter to sort images to reveal any biases in the example images from which the concept activation vector is learned.

LIME

Local Interpretable Model-agnostic Explanations (LIME)

- Implements local surrogate models - interpretable models that are used to explain individual predictions
- Using data points close to the individual prediction, LIME trains an interpretable model to approximate the predictions of the real model
- The new interpretable model is then used to interpret the real result

- Now let's briefly discuss LIME, which is a popular and well-known framework for producing **local explanations**.
- LIME is a popular and well-known framework for creating local interpretations of model results.
- The idea is quite intuitive. First, forget about the training data and imagine that you only have a black-box model where you can input data points and get the predictions of the model.
- You can probe the black-box as often as you'd want.
- Your goal is to understand why the model made a certain prediction.
- **LIME tests what happens to the predictions when you give variations of your data to the model**.
- LIME generates a new dataset consisting of permuted samples and the corresponding predictions of the model.
- With this new dataset, LIME then trains an interpretable model, which is weighted by the distance from the sampled instances to the result that we're interpreting.
- The interpretable model can be anything that is easily interpretable, like a linear model or a decision tree.
- The new model should be a reasonably good approximation of the model results locally, but it does not have to be a good global approximation.
- This kind of accuracy is called **local fidelity**.
- You then explain the prediction by interpreting the new local model.

AI Explanations

Google Cloud AI Explanations for AI Platform

Explain why an individual data point received that prediction

Debug odd behavior from a model

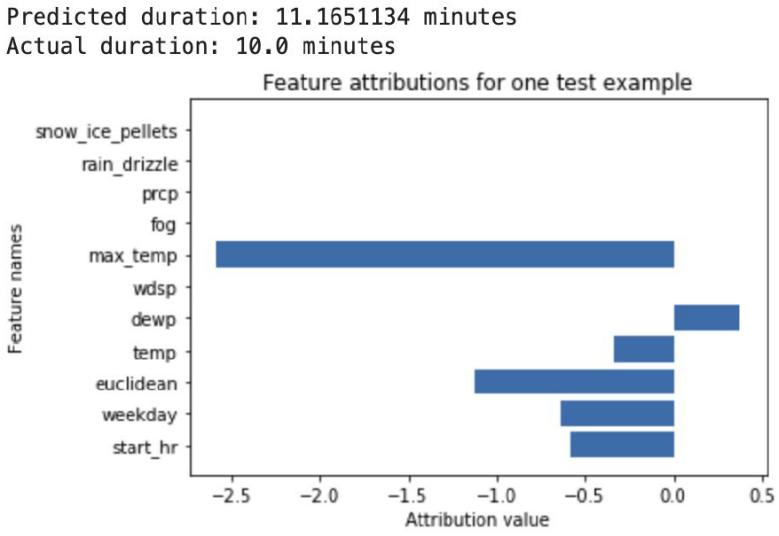
Refine a model or data collection process

Verify that the model's behavior is acceptable

Present the gist of the model

- There are also Cloud-based tools and services which can be very valuable for interpreting your model results.
- Let's look at one of these now, **Google's AI Explanations** service.
- Another option for interpretability is to leverage managed services from Cloud providers, such as Google's AI Explanations for AI Platform.
- AI Explanations integrates feature attributions into Google's AI Platform Prediction service.
- AI Explanations helps you understand your model's output for classification and regression tasks.
- Whenever you request a prediction on AI Platform, AI Explanations tells you how much each feature in the data contributed to the predicted results.
- You can then use this information to verify that the model is behaving as expected and identify any bias in your models, and get ideas for ways to improve your model and your training data.

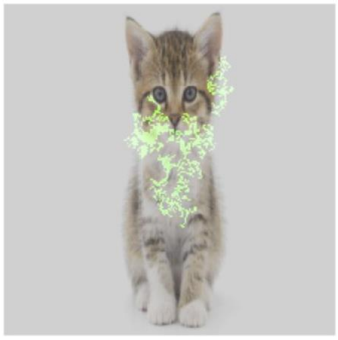
AI Explanations: Feature Attributions



Tabular Data Example

- **Feature attributions** indicate how much each feature contributed to the given prediction.
- AI Explanations works with Google's AI Platform Prediction managed service.
- When you request predictions from your model normally using AI Platform Predictions, you only get the predictions.
- However, when you request explanations, you get both the predictions and the feature attribution information for those predictions.
- There are also visualizations provided to help you understand the feature attributions.
- This chart shows an example of feature attributions for a tabular dataset.

AI Explanations: Feature Attributions



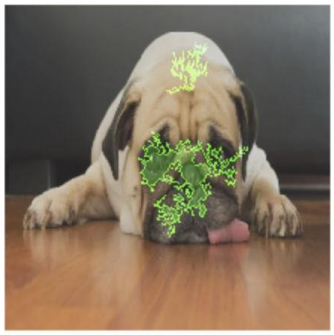
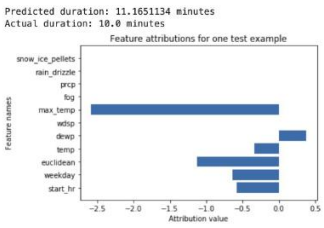


Image Data Examples

- These are examples of the visualizations for image data.
- They include an overlay for each image, **highlighting which pixels in the image contributed most strongly to the resulting prediction**.

AI Explanations: Feature Attribution Methods



- AI Explanations currently offers three methods of feature attribution.
- These include **sampled Shapley**, **integrated gradients**, and **XRAI**.
- **But ultimately, all of these methods are based on Shapley values**.
- We've discussed Shapley enough that we don't need to go over that again, but let's look at the other two methods, integrated gradients and XRAI.

AI Explanations: Integrated Gradients

A gradients-based method to efficiently compute feature attributions with the same axiomatic properties as Shapley values

- Integrated gradients is a different way to generate feature attributions with the same axiomatic properties as Shapley values based on using gradients.
- It is orders of magnitude **more efficient than the original Shapley-Shubik method when applied to deep networks**.

- In the integrated gradients method, the gradient of the prediction output is calculated with respect to the features of the input along an integral path.
- The gradients are calculated at different intervals based on a scaling parameter that you can specify.
- For image data, imagine the scaling parameter as a slider that is scaling all the pixels of the image to black.
- By saying that the gradients are integrated, it means that they are first averaged together, and then the element-wise product of the average gradients and the original input is calculated.

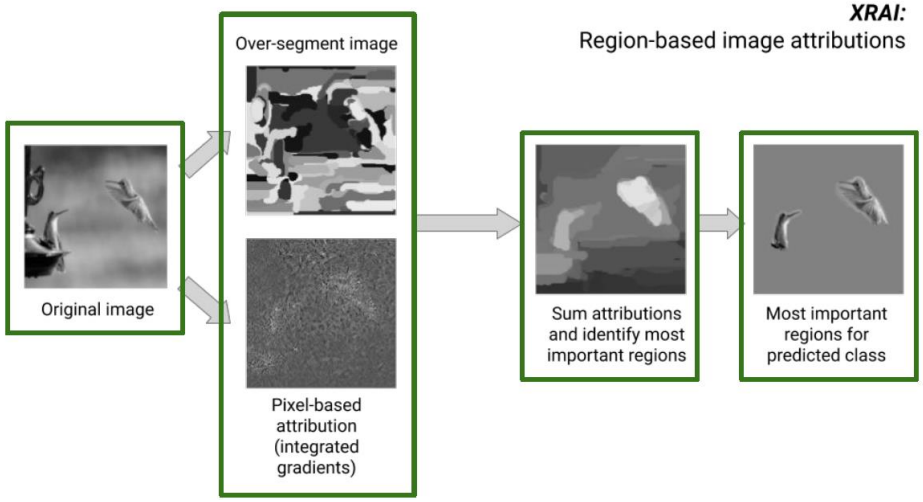
AI Explanations: XRAI (eXplanation with Ranked Area Integrals)

XRAI assesses overlapping regions of the image to create a saliency map

- Highlights relevant regions of the image rather than pixels
- Aggregates the pixel-level attribution within each segment and ranks the segments

- XRAI or eXplanation with Ranked Area Integrals is **specifically focused towards image classification**.
- The XRAI method extends the integrated gradients method with additional steps to determine which regions of the image contribute most to a given prediction.
- XRAI performs pixel-level attribution for the input image using the integrated gradients method.
- Independently, a pixel-level attribution XRAI also over-segments the image to create a patchwork of small regions.
- XRAI then aggregates the pixel-level attribution within each segment to determine the attribution density of that segment.
- It then ranks each segment and orders them from the most to the least positive.
- This determines which areas of the image are the most **salient** or contribute the most strongly to a given prediction.

AI Explanations: XRAI (eXplanation with Ranked Area Integrals)



- Here's an example of an image of hummingbirds.
- Starting at the left, the original image is both segmented into different regions and the pixel-level attributions are calculated using integrated gradients.
- The attributions within each region are then summed and the regions are ranked to determine the results which are the most important.
- As you can see in the image on the right, the most important parts of this image are the hummingbirds themselves.

References

- Explainable AI
- Responsible AI
- Interpretable Machine Learning - A Guide for Making Black Box Models Explainable.
- TensorFlow Lattice
- Monotonic Calibrated Interpolated Look-Up Tables
- Permutation feature importance
- A Unified Approach to Interpreting Model Predictions
- Explainable AI for Trees: From Local Explanations to Global Understanding
- TCAV
- LIME
- AI Explanations