CS 6771J1, Deep Learning, Fall 2020
New Jersey Institute of Technology
Project 2 – Explainability

Group 4 Participants:
Paul Aggarwal
Navneet Kala
Akash Shrivastava

EXPLAINABLE AI

An important question in the field of machine learning is why an algorithm made a certain decision. This is important for a variety of reasons. As an end user, I am more likely to trust a recommendation if I understand why it was exposed to me. As an organization, understanding that customers made a purchase because this campaign was particularly effective can allow me to tailor my future outreach efforts.

Deep learning methods have been very effective for a variety of medical diagnostic tasks and has even beaten human experts on some of those. However, the black-box nature of the algorithms has restricted clinical use. Recent explainability studies aim to show the features that influence the decision of a model the most. The majority of literature reviews of this area have focused on taxonomy, ethics, and the need for explanations

WHAT DOES IT MEAN TO INTERPRET A MODEL (AND WHY IS IT SO HARD)?

Let's start by defining exactly what it means to interpret a model. At a very high level, we want to understand what motivated a certain prediction.

Trading off between interpretability and complexity

Let's consider a very simple model: a linear regression. The output of the model is

$$f(x_1, x_2, \ldots, x_n) = \phi_1 x_1 + \phi_2 x_2 + \ldots + \phi_n x_n$$

In the linear regression model above, we assign each of the features x_i a coefficient ϕ_i, and add everything up to get the output.

In this case, its super easy to find the importance of a feature; if ϕ_i has a large absolute value, then feature x_i had a big impact on the final outcome. However, there is also a drawback, which is that this model is so simple that it can only uncover linear relationships.

In order to uncover this more complicated relationship, we'll need a more complicated model.

However, as soon as we start using more complicated models, we lose the ease of interpretability which we got with this linear model. In fact, as soon as we try to start uncovering non-linear, or even interwoven relationships then it becomes very tricky to interpret the model.

This decision — between an easy to interpret model which can only uncover simple relationships, or complex models which can find very interesting patterns that may be difficult to interpret — is the trade off between interpretability and complexity.

This is additionally complicated by the fact that we might be interpreting a model because we are hoping to learn something new and interesting about the data. If this is the case, a linear model may not cut it, since we may already be familiar with the relationships it would uncover.

The ideal case would therefore be to have a complex model which we can also interpret.

How can we interpret complex models?

Thinking about linear regressions has yielded a good way of thinking about model interpretations:

We'll assign to each feature x_i a coefficient φ_i which describes — linearly — how the feature affects the output of the model. We've already discussed the shortcomings of this model, but let's analyze a little more

Across many data points, the coefficients φ will fail to capture complex relationships. But on an individual level, then they'll do fine, since for a single prediction, each variable will truly have impacted the model's prediction by a constant value.

What we've done here is take a complex model, which has learnt non-linear patterns in the data, and broken it down into lots of linear models which describe individual data points. Its important to note that these explanation coefficients φ are not the output of the model, but rather what we are using to interpret this model. By aggregating all of these simple, individual models together, we can understand how the model behaves across all the customers.

So, to sum up:

Instead of trying to explain the whole complex model, we are just going to try and explain how the complex model behaved for one data point. We will do this using a linear explanation model; let's call it g.

In addition, to further simplify the simple model, we won't multiply the coefficients φ by the original feature value, x. Instead, we will multiply it by 1 if the feature is present, and 0 if it is not.

We will do this for all the data points and aggregate it to get an idea of how the model worked globally.

$$g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z_i' = \text{bias} + \sum \text{contribution of each feature}$$

Here g(z') is a local surrogate model of the original model f(x) and φ_i is how much the presence of feature i contributes to the final output, which helps us interpret the original model.

Now that we have this framework within which to interpret complex models, we need to think about exactly what properties we want φ to capture to be useful.

SHAP ~ SHAPLEY VALUES (OR, HOW TO CALCULATE Φ?)

SHAP belongs to the class of models called ''additive feature attribution methods'' where the explanation is expressed as a linear function of features. Linear regression is possibly the intuition behind it.

Say we have a model house_price = 100 * area + 500 * parking_lot. The explanation is straightforward: with an increase in area of 1, the house price increase by 500 and with parking_lot, the price increase by 500.

SHAP tries to come up with such a model for each data point. Instead of the original feature, SHAP replaces each feature (x_i) with binary variable (z_i') that represents whether xi is present or not:

The solution to finding the values of φ predates machine learning. In fact, it has its foundations in game theory.

Consider the following scenario: a group of people are playing a game. As a result of playing this game, they receive a certain reward; how can they divide this reward between themselves in a way which reflects each of their contributions?

There are a few things which everyone can agree on; meeting the following conditions will mean the game is 'fair' according to Shapley values:

- The sum of what everyone receives should equal the total reward
- If two people contributed the same value, then they should receive the same amount from the reward
- Someone who contributed no value should receive nothing
- If the group plays two games, then an individual's reward from both games should equal their reward from their first game plus their reward from the second game

It turns out that there is only one method of calculating φ so that it respects all above rules.

Lloyd Shapley introduced this method in 1953 (which is why values of φ calculated in this way are known as Shapley values).

The Shapley value for a certain feature i (out of n total features), given a prediction p (this is the prediction by the complex model) is

$$\phi_i(p) = \sum_{S \subseteq N/i} \frac{|S|!(n-|S|-1)!}{n!}(p(S \cup i) - p(S))$$

There's a bit to unpack here, but this is also much more intuitive than it looks. At a very high level, what this equation does is calculate what the prediction of the model would be without feature i, calculate the prediction of the model with feature i, and then calculate the difference:

$$\text{Importance of i} = p(\text{with i}) - p(\text{without i})$$

This is intuitive; we can just add features and see how the model's prediction changes as it sees new features. The change in the model's prediction is essentially the effect of the feature.

However, the order in which we add features is important to how we assign their values.

SIMPLE TOY EXAMPLE EXPLAINING THE SHAPLEY FORMULA
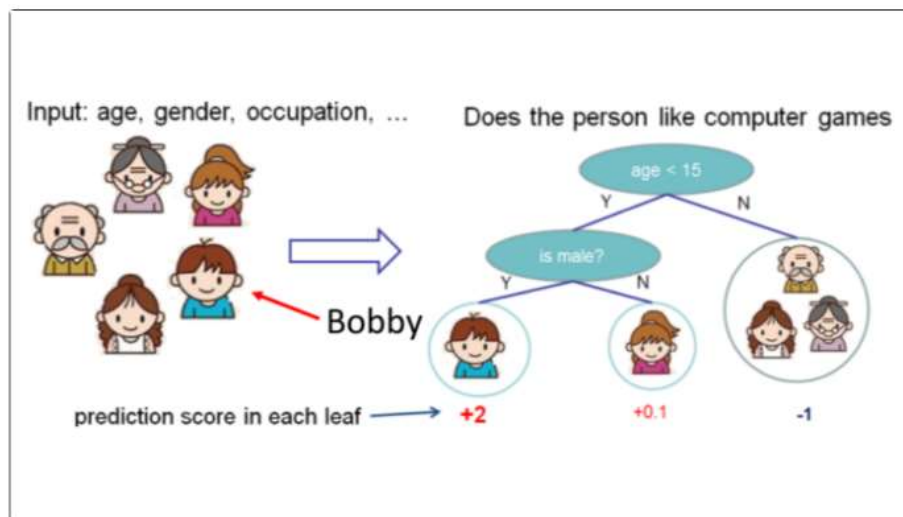
First, we'll see Bobby's age, and then his gender.
When the model sees Bobby's age, it will take him left on the first split. Then, since it doesn't have a gender yet, it will assign him the average of the leaves below, or (2 + 0.1) / 2 = 1.05. So the effect of the age feature is 1.05.

Then, when the model learns he is male, it will give him a score of 2. The effect of the gender feature is therefore 2−1.05=0.95.

So in this scenario, $\phi\_Age\ Bobby$=1.05 and $\phi\_Gender\ Bobby$=0.95.

Next, lets say we see his gender, and then his age.

In the case where we only have a gender, the model doesn't have an age to split on. It therefore has to take an average of all the leaves below the root.

First, the average of the depth 2 leaves: (2 + 0.1) / 2 = 1.05. This result is then averaged with the other depth 1 leaf: (1.05 + (-1)) / 2 = 0.025. So, the effect of the gender feature is 0.025.

Then, when the model learns he is 14, it gives him a score of 2. The effect of the age feature is then (2–0.025)=1.975.

<mark>So in this scenario, $\phi\_Age\ Bobby$=1.975 and $\phi\_Gender\ Bobby$=0.025.</mark>

Which value should we assign $\phi\_Age\ Bobby$? If we assign $\phi\_Age\ Bobby$ a value of 1.975, does this mean we assign $\phi\_Gender\ Bobby$ a value of 0.025 (since, by rule 1 of Shapley fairness, the total coefficients must equal the final prediction of the model for Bobby, in this case 2)?

This is far from ideal, since it ignores the first sequence, in which $\phi\_Gender\ Bobby$ would get 0.95 and $\phi\_Age\ Bobby$ would get 1.05.

What a Shapley value does is consider both values, calculating a weighted sum to find the final value. This is why the equation for $\phi_i(p)$ must permute over all possible sets of S of feature groupings (minus the feature i we are interested in). This is described in $S\subseteq N/i$ below the summation, where N is all the features.

How are the weights assigned to each component of the sum? It basically considers how many different permutations of the sets exist, considering both the features which are in the set S (this is done by the $|S|!$) as well as the features which have yet to be added (this is done by the $(n-|S|-1)!$. Finally, everything is normalized by the features we have in total.

SUMMARY

In summary, Shapley values calculate the importance of a feature by comparing what a model predicts with and without the feature. However, since the order in which a model sees features can affect its predictions, this is done in every possible order, so that the features are fairly compared.

Unfortunately, going through all possible combinations of features quickly becomes computationally unfeasible. Luckily, the SHAP library introduces optimizations which allow Shapley values to be used in practice. It does this by developing model specific algorithms, which take advantage of different model's structures. For instance, SHAP's integration with gradient boosted decision trees takes advantage of the hierarchy in a decision tree's features to calculate the SHAP values. This allows the SHAP library to calculate Shapley values significantly faster than if a model prediction had to be calculated for every possible combination of features.

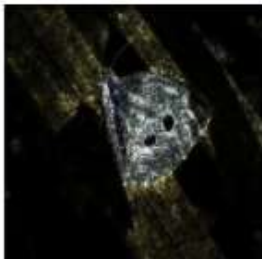## ADVANCED EXPLAINABLE AI FOR IMAGE CLASSIFICATION

Integrated Gradients(IG) :

For linear models, ML practitioners regularly inspect the products of the model coefficients and the feature values in order to debug predictions. Gradients (of the output with respect to the input) is a natural analog of the model coefficients for a deep network, and therefore the product of the gradient and feature values is a reasonable starting point for an attribution method

Formally, suppose we have a function $F : R\ n \rightarrow [0, 1]$ that represents a deep network. Specifically, let $x \in R\ n$ be the input at hand, and $x\ 0 \in R\ n$ be the baseline input. For image networks, the baseline could be the **black image**, while for text models it could be the zero embedding vector. We consider the straight-line path (in $R\ n$) from the baseline $x\ 0$ to the input x, and compute the gradients at all points along the

path. Integrated gradients are obtained by cumulating these gradients. Specifically, integrated gradients are defined as the path integral of the gradients along the straight-line path from the baseline x 0 to the input x.

$$\mathrm{IntegratedGrads}_i(x) ::= (x_i - x_i') \times \int_{\alpha=0}^{1} \frac{\partial F(x' + \alpha \times (x - x'))}{\partial x_i}\, d\alpha$$

## shap.GradientExplainer:

GradientExplainer explains a model using expected gradients (an extension of integrated gradients). Expected gradients an extension of the integrated gradients method (Sundararajan et al. 2017), a feature
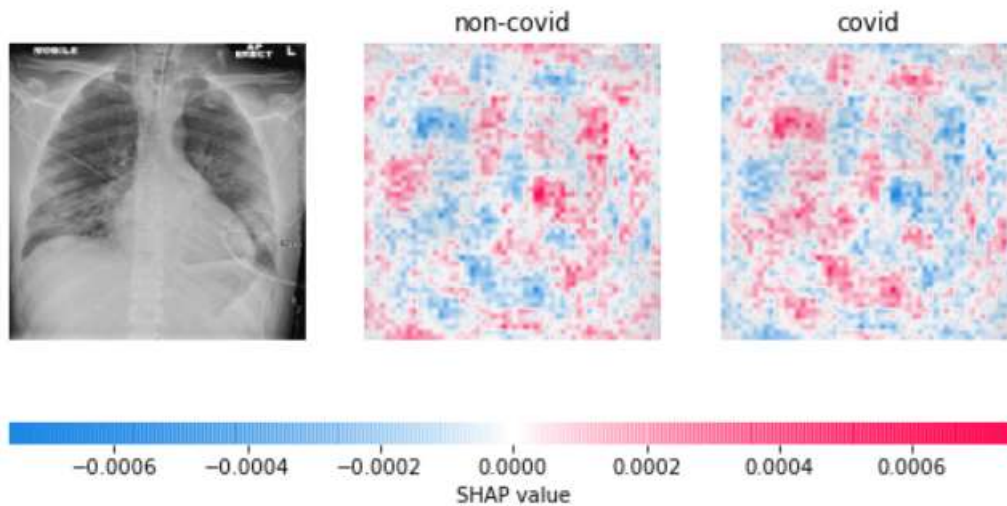
attribution method designed for differentiable models based on an extension of Shapley values to infinite player games (Aumann-Shapley values). Integrated gradients values are a bit different from SHAP values, and require a single reference value to integrate from. As an adaptation to make them approximate SHAP values, expected gradients reformulates the integral as an expectation and combines that expectation with sampling reference values from the background dataset. This leads to a single combined expectation of gradients that converges to attributions that sum to the difference between the expected model output and the current output.

Expected gradients combines ideas from Integrated Gradients, SHAP, and SmoothGrad into a single expected value equation. This allows an entire dataset to be used as the background distribution (as opposed to a single reference value) and allows local smoothing. If we approximate the model with a linear function between each background data sample and the current input to be explained, and we assume the input features are independent then expected gradients will compute approximate SHAP values. In the example below we have explained how Gradient Explainer model impacts the output probabilities.

```
Predicted Class        non-COVID-19
p(non-COVID-19)            0.805962
p(COVID-19)                0.194038
Name: 53, dtype: object
```
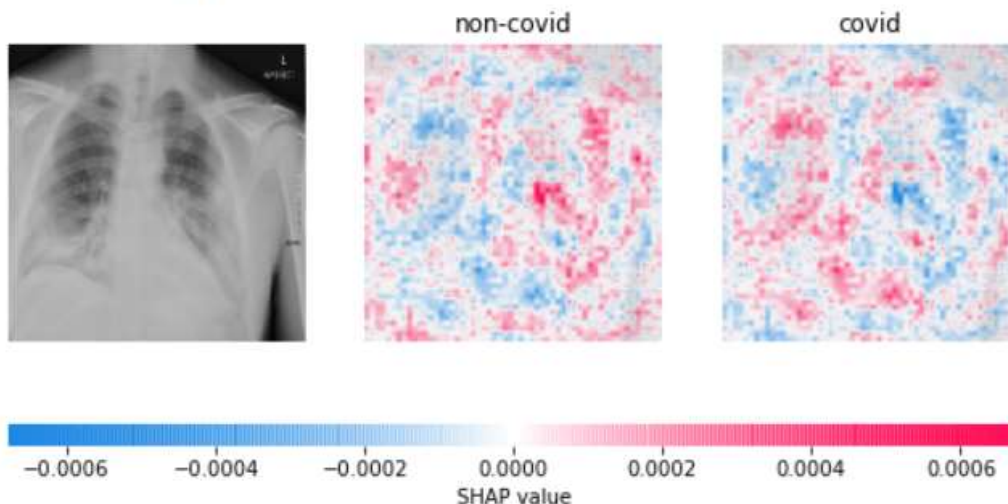
```
Predicted Class          COVID-19
p(non-COVID-19)          0.0535442
p(COVID-19)              0.946456
Name: 49, dtype: object
```


REFERENCES:

http://proceedings.mlr.press/v70/sundararajan17a/sundararajan17a.pdf

https://gabrieltseng.github.io/2018/06/20/SHAP.html

https://storage.googleapis.com/cloud-ai-whitepapers/AI%20Explainability%20Whitepaper.pdf

http://www.unofficialgoogledatascience.com/2017/03/attributing-deep-networks-prediction-to.html

https://meichenlu.com/2018-11-10-SHAP-explainable-machine-learning/

https://shap.readthedocs.io/en/latest/generated/shap.GradientExplainer.html