# Chapter 5 - Model-Agnostic Methods

## 1 Introduction

Let's look at methods that work for all kinds of models. If you do not use these, you should use an interpretable model or an interpretability method designed for the specific kind of model that you are using.

## 2 Partial Dependence Plot (PDP)

A PDP aims to plot the average model prediction as a function of feature value, for some feature of interest. On the $x$ axis, we vary our feature of interest. On the $y$ axis, we need to compute the average model prediction. To do this for a chosen value of our feature of interest, we simply consider each training example, plug in the chosen value of our feature of interest while keeping all other features untouched, compute the model prediction for this modified training example, and average the predictions for all the modified training examples.

The formula is as follows, where $x_S$ is the value for the feature of interest, and $x_C^{(i)}$ are the values for the other features for the $i$th example.

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

When showing a PDP, it helps to show a feature histogram below them because users can then see how frequent a particular feature value is in the dataset.

PDPs have two limitations. The first is that they assume the feature of interest is uncorrelated to all other features. To overcome this limitation, use ALE plots. The second limitation is that we may have heterogeneity in feature effect. Heterogeneity in feature effect means that for some examples, increasing the feature value increases the model prediction while for other examples, increasing the feature value decreases the model prediction. These effects might cancel out in a PDP and the user will not know that they exist. To overcome this limitation, use ICE plots.

## 3 Individual Conditional Expectation (ICE)

Again, pick a feature of interest, vary it and compute the model prediction for each modified training example. However, instead of averaging the predictions, plot a separate prediction vs. feature value curve for each training example. This will make heterogeneity in feature effect clearly visible to the user. You may also want to plot the derivative of each curve to make it clearer. Also, you can make it so that all the curves start at the same value (an anchor point).

ICE plots overcome the heterogenaity issue of PDP plots, but they still suffer from the correlation issue.

## 4 Accumulated Local Effects (ALE) Plot

Let's understand the correlation effect. Suppose we are trying to predict a person's age given measurements like their height, weight, and other factors. If height is our feature of interest, a PDP will vary height without changing weight, so we may get ridiculous modified training examples like a person who is 6 feet tall but weighs 30 pounds. This happens because PDP does not know that height and weight are correlated.

An M-plot can partially overcome this by only considering training examples where the height is similar to the chosen value of height. So, for height = 3 feet, we consider examples where height is near 3 feet (e.g. between 2.5 feet and 3.5 feet). We then modify them to have 3 feet height, compute the model prediction, and average over the predictions. The problem with M-plots is that our model might not use the height feature at all, instead relying only on the correlated weight feature, but since we are implicitly using height when we select training examples, the M-plot incorrectly shows that changing height does impact model prediction.

ALE plots overcome the limitation of M plots by computing differences in model predictions rather than using model predictions directly. The equation is:

$$\hat{\tilde{f}}_{j,ALE}(x) = \sum_{k=1}^{k_j(x)} \frac{1}{n_j(k)} \sum_{i:x_j^{(i)} \in N_j(k)} \hat{f}(z_{k,j}, x_{-j}^{(i)}) - \hat{f}(z_{k-1,j}, x_{-j}^{(i)}) \tag{2}$$

Basically, we take the domain for the feature of interest and break it into intervals. Then, for each example in an interval, we plug in the upper value of the interval (leave the other features untouched), compute the model prediction, repeat this with the lower value of the interval, and take the difference between the two predictions for each example. This difference is called the local effect. We then average the effects over all the examples.

If we want, we can also subtract the mean effect. So, the ALE is the main effect of the feature at a certain value compared to the average prediction of the data.

$$\hat{f}_{j,ALE}(x) = \hat{\tilde{f}}_{j,ALE}(x) - \frac{1}{n} \sum_{i=1}^{n} \hat{\tilde{f}}_{j,ALE}(x_j^{(i)}) \tag{3}$$

How do we pick the intervals? A common choice is to just use quantiles.

We can extend ALE plots to 2 dimensions as well, but this shows second order effects rather than first order effects. The equations are a lot uglier in this case too.

How do we handle categorical features? They do not have an order so we cannot make intervals. To impose an order, we can compute the similarity of two categories by measuring how similar the other features are. See the book for how to do this.

To look for correlated features, we can use ANOVA. Basically, we turn one feature into the target and try to predict it from the other features. We see which features are the most useful for predicting it.

ALE plots are efficient, interpretable, and work even when features are correlated. They can suffer if you don't use a good number of intervals and they are hard to implement.

Use ALE instead of PDP.

## 5 Feature Interaction

Sometimes, the interaction of two features is what really matters. The H statistic can help here. It can help us tell if two given features interact or whether a given feature interacts with some subset of the rest of the features.

For a pair of features that do not interact, we can use their partial dependence (see the PDP section) functions to get the following decomposition:

$$PD_{jk}(x_j, x_k) = PD_j(x_j) + PD_k(x_k) \tag{4}$$

If a given feature does not interact with the rest of the features, we have

$$\hat{f}(x) = PD_j(x_j) + PD_{-j}(x_{-j}) \tag{5}$$

By observing how much the actual partial dependence differs from this non-interacting case, we can get a measure of interaction strength called the H statistic. Here are the H statistics for the two feature and single feature cases described before:

$$H_{jk}^2 = \sum_{i=1}^{n} [PD_{jk}(x_j^{(i)}, x_k^{(i)}) - PD_j(x_j^{(i)}) - PD_k(x_k^{(i)})]^2 / \sum_{i=1}^{n} PD_{jk}^2(x_j^{(i)}, x_k^{(i)}) \tag{6}$$

$$H_j^2 = \sum_{i=1}^{n} [\hat{f}(x^{(i)}) - PD_j(x_j^{(i)}) - PD_{-j}(x_{-j}^{(i)})]^2 / \sum_{i=1}^{n} \hat{f}^2(x^{(i)}) \tag{7}$$

Notice that the H statistic takes $O(n^2)$ time to run. To speed it up, you can sample from the training set. The statistic varies between 0 and 1 where 0 means no interaction. There's also a hypothesis test to check if there is interaction, but your model needs to somehow make sure there is no interaction between any features (null hypothesis), which may not be possible for all models. Finally, you can also use the H statistic in classification when the output is a probability.

To use the H statistic in practice, first compute the single feature statistic for all features. Then, for highly interacting features, compute all the pairwise statistics involving the feature of interest. You can also plot 2D partial dependence plots here.

The H statistic suffers for strongly correlated features, does not work on image pixel features, and it can be hard to tell whether the statistic is large or small.

## 6 Feature Importance

First, compute the total error of the model on the training examples. Next, for your feature of interest, permute the feature values in the training examples. Compute the total error again. Compute the difference (or ratio) between the two errors. The larger the error, the more important the feature is to the model. This is called permutation feature importance.

With the above approach, you can compute the feature importance of all your features. You could also use the testing examples instead of the training examples (it's not clear whether this is better or not).

For your error function, use $1 - AUC$ for classification and mean absolute error for regression.

Using the ratio instead of difference makes the importance comparable accross different problems.

By the way, another way to compute feature importance is to just remove the feature and look at how error changes. The error change will not be as substantial as permuting the feature because the model will learn how to use the other features in a better way (or make heavier use of a correlated feature). This is also computationally intensive because you need to retrain. Do NOT try retraining the model on a sample of the data because your model could have higher error simply because the training set is smaller.

The downsides of this approach is sometimes permutations can yield nonsensical training examples (e.g. shuffling height while keeping weight fixed). Also, if you have two correlated features, they might split up feature importance between themselves.

## 7 Global Surrogate

We train an interpretable (surrogate) model, like a linear model or decision tree, to approximate a black box model. Basically, we take a dataset of examples, label them with the black box model, and train the surrogate model on the labeled examples. $R^2$ can tell us how well the surrogate model matches the black box model. If the $R^2$ is near 1, you can use the surrogate model in place of the black box model. If $R^2$ is near 0, the surrogate model is not useful.

If you only want a local surrogate, you can sample points such that points near the part of space you care about are sampled with higher weight.

## 8 Local Surrogate (LIME)

Suppose you are given an example and you want to know why your model made a particular prediction for it. We first generate new examples by randomly perturbing the given example. We then weight each example by its proximity to the given example and train an interpretable model on the (generated example, model prediction for generated example) pairs. Lasso regression is a good surrogate model because we can choose the regularization term so that it does not use too many features in the model.

How do you perturb data examples? For tabular data, we select each feature by drawing from a Gaussian parameterized by the feature mean and feature variance. We measure distance between points using the exponential smoothing kernel (picking the kernel width is hard, especially for high dimensional data). For text data, we randomly remove words. For image data, we find superpixels in the image (clusters of pixels with similar colors) and randomly turn the superpixels on or off.

The biggest problem with LIME is measuring the distance between two data points. In high dimensional data, it's very hard to find a good proximity measure ( the exponential kernel does not work well). Also, LIME explanations are not very stable in practice (nearby points sometimes have very different explanations).

## 9 Shapley Values

Let's imagine the prediction is a payout that the features (game players) compete for. This game theoretical idea is the basis for Shapley values.

Suppose we have an example and a feature of interest. Take a subset of the (feature name, feature value) pairs from the example (except the feature of interest) and call this a coalition. Randomly pick other examples that contain this coalition. Compute the model prediction for each of them, compute the difference from the prediction for the original example, and average the differences. This is the marginal contribution of the feature of interest to the coalition. We can repeat this for all possible coalitions. Thus, the Shapley value is the average of all the marginal contributions to all possible coalitions. Obviously, this is expensive to do because there could be many coalitions. So, we need a way to approximate it.

So, we use Monte-Carlo sampling. Suppose we have an example of interest (call it $x$) and a feature of interest (call it feature $j$). We pick a random example (call it $z$). We pick a random permutation of feature values and apply it to both the random example and example of interest to get $z_o$ and $x_o$, respectively. Then, we construct two new examples. The first is $x_{+j} = x_{o,1}, ..., x_{o,j-1}, x_{o,j}, z_{o,j+1}, ..., z_{o,p}$ and the second is $x_{-j} = x_{o,1}, ..., x_{o,j-1}, z_{o,j}, z_{o,j+1}, ..., z_{o,p}$. The marginal contribution is then $\hat{f}(x_{+j}) - \hat{f}(x_{-j})$. We do this $M$ times and average the marginal contributions to get the Shapley value for feature $j$. We then compute the Shapley value for all features.

Shapley values offer more truthful explanations than LIME, so use them if you need very accurate explanations. It also has solid game theory behind it.