A peek into the black box through the SHAP lenses

Why bother?

The insurance industry has always been more conservative when it comes to modelling, and with good reason. The financial and social impact of a wrongly predicted say, fraud detection model denying a policyholder’s entitlement is huge compared to a wrongly predicted “You may like this song!” message pushed from your favourite music player’s recommendation system.

As actuaries doing technical modelling for insurance premiums, some common “workarounds” would be to stick with our favourite GLMs since it is a lot more transparent, or map/tag your GBM results to that of a GLM in some way, just to get some comfort and clarity regarding what the model is doing.

While this seems to suffice, as data becomes increasingly available and insurance products continue to get more complex as technology evolves, the use of more robust models to handles these interactions will be inevitable for many prediction tasks across the whole insurance value chain, not just for claims modelling and fraud detection.

An introduction to SHAP values

We know that the best explanation for a model would be the model itself. We can see this from our GLM outputs, with the simplest case that we have come to love and learn in school looking like this:

While we can appreciate the simplicity and global interpretability of this, the more complex models’ (like ensemble methods and neural networks) outputs cannot be summarized into a nice, closed form formula like shown above. Even if we were to painstakingly compose all the functions in our 4-layer neural network and write it out in a formula, there would be no meaningful/actionable interpretation for the purposes of presenting or debugging.

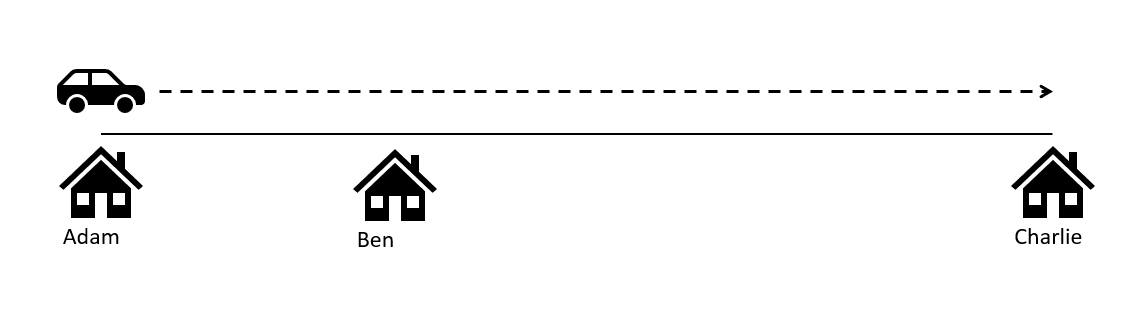
The approach taken to work around this the use of an “explanation model” on top of the original prediction model, which is defined as any interpretable approximation of the original model, (Lundberg, 2017). These explanation models provide *local interpretability* as opposed to *global*, which means providing an explanation as to why a certain prediction was made for a particular data point, instead of a “one-size-fits-all” formula like we see above. There are several popular explanation models that you may have seen (like LIME), but the focus in this article will be SHAP, which is introduced as a unified measure of feature importance, and the only model currently that fulfills a set of properties and axioms, which we will not dive into here.

SHAP stands for Shapley Additive Explanations and uses a game theory approach applied on machine learning to allocate “contributions” to the model features. Let us go through this by using a ride sharing analogy. Say we have 3 people:

* Adam
* Ben
* Charlie

They have decided to share a taxi back home, as shown in the simplified graph below.

TODO: Fix graph, Adam would not have to take a cab according to this picture lol.



The next logic question would then be how to split the cost of the ride fairly. If you are not psychotic, you would probably be happy to split the bill evenly and just forget about it. Though this method is not “fair” in anyway, the upside is that you would still have friends, which is arguably more important.

If we take it 1 step further, we might say that using a pro-rata approach by distance would be a good approximation to what’s fair for each person to pay, and you might be right. However, note that while distance would be the most obvious, it is not the only contributing factor to what the fare will end up being. By pro-rating it on distance, we are failing for account for other less obvious factors like traffic or any other interactions between those factors. While you might have gotten 1 step closer in trying to fairly allocate the fare, you would have probably lost your friends by this point.

So then, if we put this in the context of SHAP values, how we would fairly allocate the fair (at a very high-level) would be as follows:

* Take the taxi and record the fair 2^K times, where K is the number of people (3 in this case, so a total of 8 times). This is also known as the “power set”.
* Record the marginal increase/decrease to the total fair given the addition of each person
* Calculate the weighted\* sum of all these marginal contributions to get the SHAP contribution for each person.
* For any given taxi ride, the sum of SHAP contributions of Adam, Ben and Charlie will perfectly explain the difference between the average price of a taxi ride and the actual fare.

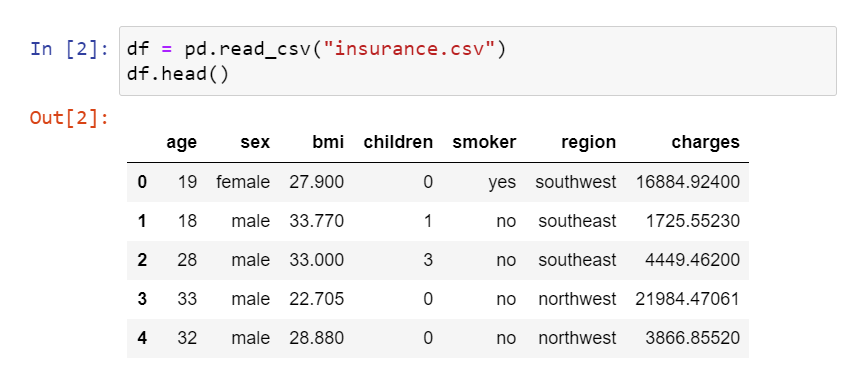
If we put this in the context of a statistical model, we can translate our 3 customers: Adam, Ben and Charlie in the features of our data, and the model response would be the outcome of the fare. However, just by translating the logic, we see that we’d have to rerun our model 8 times to get the responses in order to calculate the marginal contributions, and this is only for 3 features! You can imagine how much computational overhead there is even if we have a mid-sized dataset with maybe 20-30 columns. Thankfully, the good people working on the SHAP algorithm have made used of approximations and samplings to make this task feasible by using a form of the conditional expectation formulation and integrating out the “unused” features in each iteration:

Where S is some subset of features. Let’s go through the python implementation of the SHAP package with a simple insurance claims dataset.

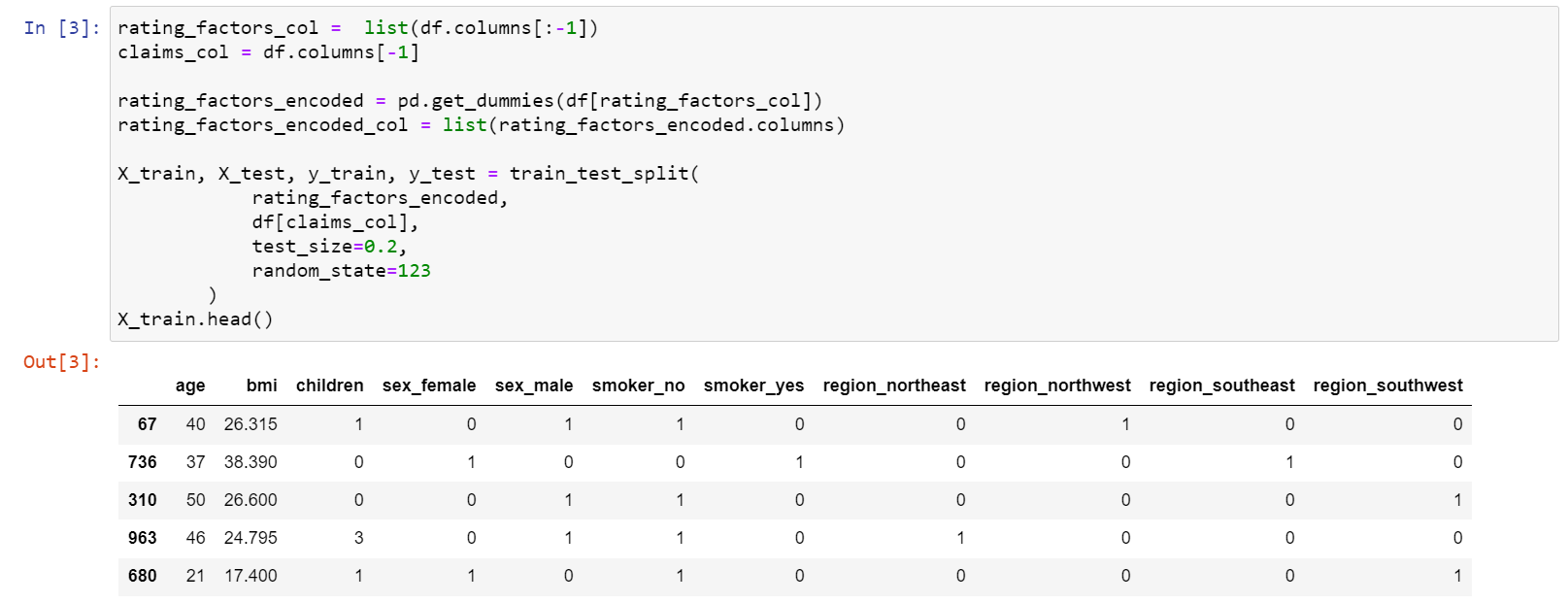
Example 1: SHAP Implementation in Python for Linear Regression

The dataset can be found here <https://github.com/sharmaroshan/Insurance-Claim-Prediction>.

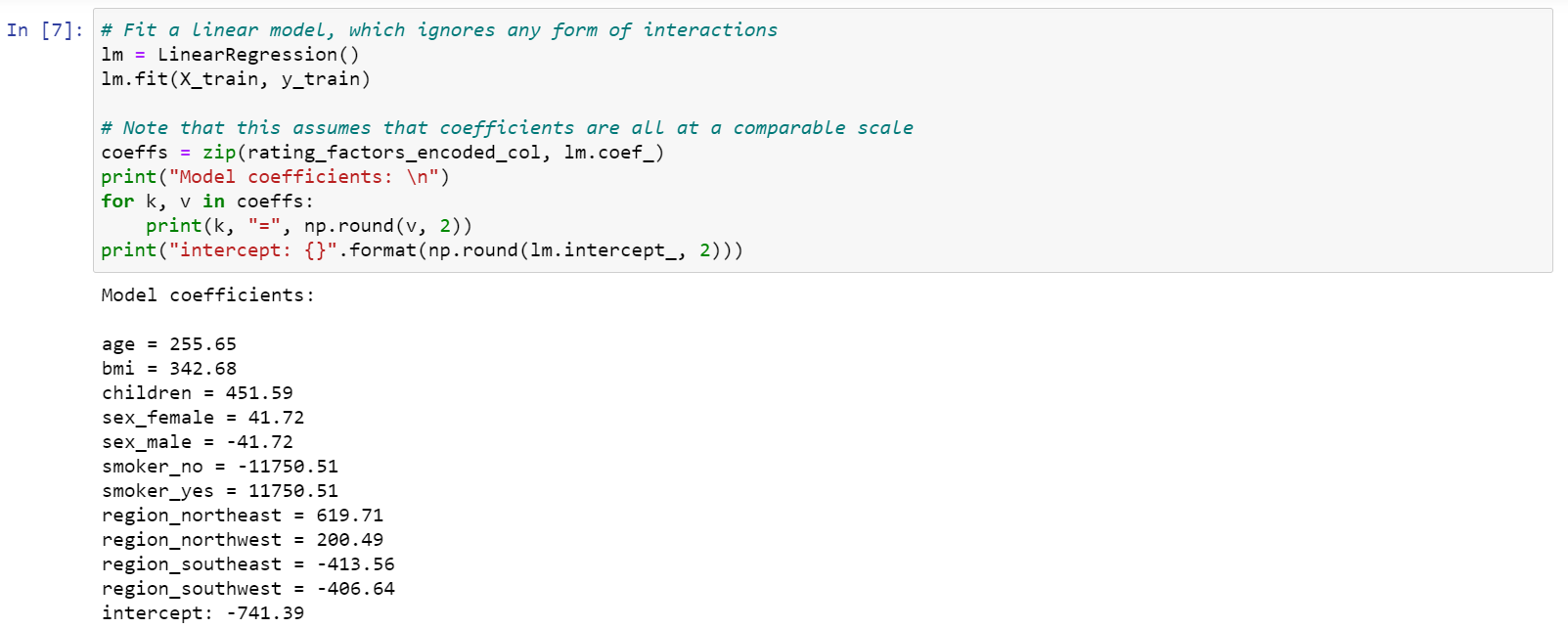
It is a simple 6-feature data set of policyholders with a response showing how much the claims costs were.



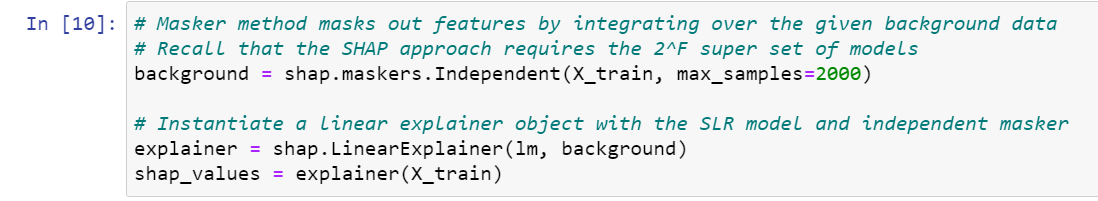
Since the focus of this article is to understand SHAP and the model validation part of this pipeline, we would not spend too much time on optimizing hyperparameters and model selection. Keeping that in mind, the next step is just to do some pre-processing of the categorical variables as well as split the dataset into training and testing sets.



Next, we just instantiate a simple linear regression object using sci-kit learn, which (unsurprisingly) returns a set of coefficients.



The SHAP library in Python makes use of explainer objects to compute SHAP values and can be calculated with 3 simple lines shown below. TODO: Explain maskers.Independent.



The package then offers a wide variety of plots for easy interpretation, but for our purposes, let’s use the waterfall graph as shown below for the 2nd observation of the training dataset.

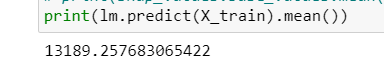
TODO: Fix the formatting

TODO: Decide on whether to discuss relationship with PD plots

TODO: Write some unit tests to check reasonableness of results



Here, we can see that the start point of the waterfall is E[f(X)], which just represents the global average of the data, or the null model.



We then slowly tag on the SHAP contributions for each feature until we end up with the actual model prediction, f(x) which in this case, is $33,251.56. We can tell instantly that the driving force for this risk would be whether she is a smoker, which brought the predicted claims costs up by a whooping $18.8k from the global average.

Example 2: SHAP Implementation in Python for GBMs

Let’s now do the same but with a more complex ensemble model like a GBM.

TODO: Recreate waterfall graph again

TODO: decision – Compare the PD plots? Look at vertical dispersion due to interactions?

Summary