Machine Learning Interpretability with LIME and SHAP

Omar A. Jiménez-Negrón

Data Mining & Machine Learning (ESMA4016-086)

Prof. Edgar Acuña

May 31, 2020

1 Introduction

A key challenge Machine Learning (ML) faces today is in relation to the so-called *accuracy-interpretability* tradeoff. Accuracy, of course, is the standard metric for evaluating the performance of a model, by measuring how often an algorithm classifies (or predicts) a data point correctly. Interpretability, on the other hand, has been defined in several ways. Perhaps one of the simplest ones was provided by [9]: *The ability to explain or to present in understandable terms to a human.*

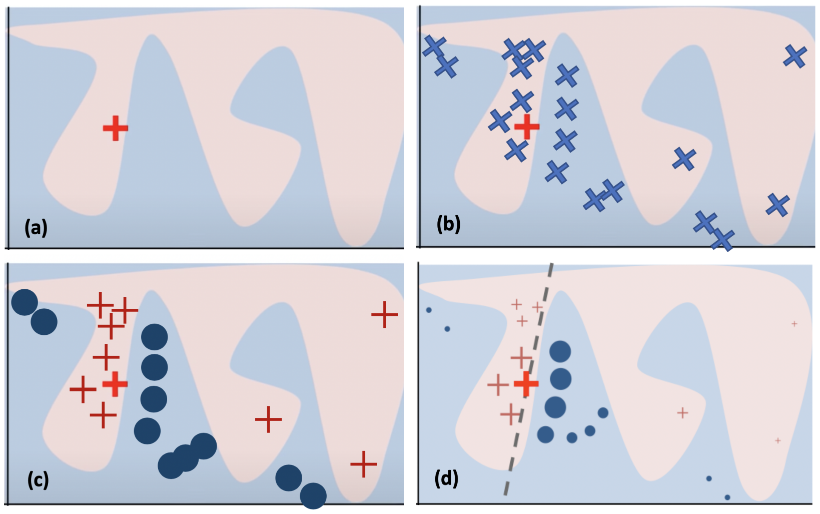
The Big Data era has led ML users and practitioners to often use and take advantage of the flexibility of highly complex models (e.g. neural networks, random forests), which properly trained can lead to very high accuracies. The main problem with these *complex* models is that, most of the times, they are *very* difficult to interpret, if not nearly impossible. This gives rise to the black box dilemma, in the sense that we know what goes in (input), what comes out (output), but we do not fully understand the reasons behind *why* algorithms act the way they do. Consequently, an important question is to what extent can we actually *trust* these models. Since AI in general is increasingly being used in our daily lives, one could argue that eventually many questions will arise of when these models go *wrong*. As an example, just imagine a machine learning model is responsible for rejecting your job application, denying you a loan, or even sending you to jail!

Because of this, in many important applications such as medicine, users may still be inclined towards the use of simpler models (e.g. linear/logistic regression, single decision tree), which *can* be interpretable within the right contexts. However, the restrictiveness of these *simple* models is usually accompanied by a lack of accuracy. There are thus two different approaches to the solution of this problem. Of course, one of those is thinking how can we try to make *simple* models more accurate, and there are a lot of research efforts going in this direction. The methods that I will be discussing and employing in this project (LIME and SHAP) focus on the inverse approach, i.e. trying to extract interpretability from already *complex* models.

1.1 Local Interpretable Model-agnostic Explanations (LIME)

One of the most important breakthroughs in the interpretability of *complex* models was developed by Ribeiro et. al [1]. In this work, the authors introduce what they have called the Local Interpretable Model-agnostic Explanations (LIME) method, an explainer capable of explaining the predictions of *any* classifier in an interpretable way and without making any assumptions of an underlying *complex* model f (as the keyword *Model-agnostic* might suggest!). In summary, LIME generates explanations by learning an **interpretable local model** g around the data point we are interested in. In principle, g can be any one of a class of potentially interpretable models $\mathcal{G}$, i.e. $g \in \mathcal{G}$, which include linear models, decision trees, among others. Here we will focus on linear models, since these are the ones the original authors focus on as well.

The original implementation of LIME is designed to work with many data types (text, numerical, image). It does this in a variety of forms by mapping its original inputs through a mapping function $x = h_x(x^{'})$, where $x^{'}$ is a *simplified* or *interpretable* input. For example, an *interpretable* input for text classification could be a binary vector indicating the presence or absence of a word, while the classifier may use more complex features such as word embeddings, which are *not interpretable* for humans. On the other hand, for images, an *interpretable* representation could be a binary vector indicating the *presence* (or *absence*) of a contiguous patch of similar pixels (also known as a super-pixel), while the classifier may represent the image as a tensor. Here we will be using LIME for image analysis, since it is a good example to explain *why* a model is making a prediction and there are some very good built-in functions in the package to achieve this. Similarly, it should also be noted that LIME has also been recently extended to regression models, although here we will focus on classification exclusively. An intuitive step by step explanation of how LIME works is provided below.



. Figure 1: Toy 2D example to present intuition for LIME.

As mentioned previously, the goal of LIME is not to provide a *global* explanation of how a *complex* model is behaving, but rather a **local explanation** of *why* it makes a certain prediction. It is evident by looking at the decision function in Figure 1 that any linear model will be terrible if applied *globally*! But this is not necessarily the case if we use a *local* approach. We start by considering a single data point, the one we are trying to explain (denoted with a bright red cross in Figure 1a), around which we are interested in developing a surrogate model. The first thing LIME does is it will perturb features to generate similar examples around that point (Figure 1b). In the case of numerical features, LIME does this by drawing values from a Normal(0,1) and then performing the inverse operation of mean centering and scaling, subject to the means and standard deviations in the training data. For categorical features, LIME will sample according to the training distribution, and will make a binary feature that is equal to 1 when it has the same value as the data point being explained [5]. Simply put, it uses *frequency* as criterion rather mean. In the next step, LIME uses the *complex* model f to predict the labels for the new data (Figure 1c). At this point, the algorithm computes the distance between the perturbed data and the original instance, and assigns weights according to it. Again, we are interested in developing a *local* model; points far from the instance we are trying to explain are not very important. This is what LIME uses to generate the surrogate model, here shown Figure 1d.

The abovementioned explanation of how LIME works can be represented via an equation of the form

$$\xi(x) = \underset{g \in \mathcal{G}}{\operatorname{argmin}} \; \mathcal{L}(f, g, \pi_x) + \Omega(g) \tag{1}$$

where $\xi(x)$ is the explainer, the first term on the RHS represents a loss function defined as $\mathcal{L}(f, g, \pi_x) = \sum_{z,z' \in \mathcal{Z}} \pi_x(z)(f(z) - g(z'))^2$, which is in turn a function of both models and a local kernel $\pi_x(z) = \exp(-D(x,z)^2/\sigma^2)$, that sort of *defines* what *locality* means. The term $\Omega(g) = \infty \mathbb{1}[||w_g||_0 > K]$ acts in a way as a regularizer that *penalizes* the complexity of the surrogate model g, via the hyperparameter K. Thus, in the case of a linear regression, $\Omega(g)$ is simply the number of non-zero coefficients, while for decision trees it may be the depth of the tree. In general, we are interested in minimizing the first term, while keeping the second one low enough to ensure both **local fidelity** and **interpretability**! Of course, the actual mathematics is somewhat more complex than what I have discussed here, and the solution of (1) is not a straightforward task (technically, it is solved using penalized linear regression). However, for the current purpose, it is more than enough to understand how LIME generates the explanations for the test predictions later on.

As will become clearer later on, LIME also has some drawbacks: it *may* be unstable and inconsistent sometimes. Moreover, although certainly being one of the most important breakthroughs in interpretability methods, it is certainly *not the only way to look at the problem.* In fact, many other explanation methods have been developed in the last couple of years that use other approaches (e.g. DeepLIFT, QII, Shapley regression values, Shapley sampling, etc). Thus, a valid question is *what is the relationship between LIME and other explanation methods? When is one preferable over the other?*

1.2 SHapley Additive exPlanation (SHAP)

The goal of SHAP is, essentially, to address this problem. To achieve this, it employs a **game theoretic approach**, which is why before proceeding into the details some background on game theory is necessary to properly understand it. The fundamental concept behind SHAP are the **Shapley values** (as the acronym SHAP might suggest), a concept introduced in 1951 by American mathematician and Nobel Prize-winning economist Lloyd Shapley.

An intuitive way to explain the origin of Shapley values is to consider a *cooperative game* consisting of several *players*, each of which contributes marginally and, by the end of the *game*, they reach an arbitrary final amount of *money*. Of course, something that must be considered is *how do we actually divide money between players in a fair way* (assuming each contributes in a different way)? To define *fairness*, Shapley introduces a set of properties that must be satisfied when considering *fairness*. Out of all these, the two that will be most relevant within the ML context are the *local accuracy (addivitity)* and *consistency (monotonicity)* properties. *Addivitity*, simply put, refers to the fact that the sum of *money* players get together should be the final game result. On the other hand, *consistency* refers to the fact that a player that contributes more to the final outcome should not earn less *money*. The main idea here is that he proved Shapley values are in fact *unique* solutions to this problem, i.e. distributing the *money* in a *fair* way considering these properties. Mathematically, this can be represented in the following form:

$$\phi_i = \sum_{S \subseteq F \setminus \{i\}} \frac{|S|!(|F| - |S| - 1)!}{|F|!} [f_{S \cup \{i\}}(x_{S \cup \{i\}}) - f_S(x_S)] \tag{2}$$

where ϕ_i is the Shapley value for *player* i, $|F|$ is the total number of players and the sum extends over all subsets S of F not containing player i. The formula can be interpreted by considering the last term as the *marginal* contribution of each player, i.e. how the final outcome of the *game* is affected by their absence. The Shapley value is then the average over every possible subset of players S.

It may thus seem natural to extend these concepts to the context of machine learning, in which models become the equivalent of the *game* and features are analogous to the *players*. Thus, the goal now is to examine the *marginal* contribution of a feature to the output of the model, and averaging this over every possible subset of features will give the Shapley value of that particular feature. This is exactly what the developers of SHAP did. The key discovery in the work of Lundberg et al. that allows the connection between game theory and ML is the realization that most of the current explanations methods (including LIME, and others not discussed here) produce explanation models of the following form:

$$g(z^{'}) = \phi_0 + \sum_{i=1}^{M} \phi_i z_i^{'} \tag{3}$$

i.e. essentially linear functions of binary variables, which they term **Additive Feature Attribution Methods**. Within an ML context, the previously mentioned *fairness* properties take relatively simple mathematical forms. In the case of *additivity*, it is defined as $f(x) = g(x^{'}) = \phi_0 + \sum_{i=1}^{M} \phi_i x_i^{'}$, which is clearly similar to Equation 3 (the only real difference is that *additivity* has to be explained in terms of the *original data* and the explanation model is generated using the *perturbed data*, as was explained in the previous section with LIME). Similarly, *consistency* can be represented as $f_x^{'}(z^{'}) - f_x^{'}(z^{'} \setminus i) \geq f_x(z^{'}) - f_x(z^{'} \setminus i)$. The latter simply states that if we have a model and we *reach in* and make it depend more on a feature, then the attribution for that feature *should not go the other way.* This is *very* important within the ML context, because violating *consistency* would suggest you cannot trust feature orderings based on your attributions, *even within the same model.*

Taking all of this into consideration, the authors derive the following theorem regarding the computation of Shapley values within an ML context:

$$\phi_i(f, x) = \sum_{z^{'} \subseteq x^{'}} \frac{|z^{'}|!(M - |z^{'}| - 1)!}{M!} [f_x(z^{'}) - f_x(z^{'} \setminus i)] \tag{4}$$

which by inspection is also clearly analogous to Equation 2. The problem is that the *exact* computation of Equation 4 is a computationally demanding task — it is actually an NPhard problem! For this reason, it should often be approximated. This is where the main contribution of SHAP comes into play: it takes advantage of the previously mentioned relationship to sort of *draw strings* from explaining methods. SHAP has connections with many explanation methods but here we will focus on Kernel SHAP (model-*agnostic* — related to LIME) and Tree SHAP (model-*specific*), since the classifier from the second dataset we will be analyzing (Bank Marketing) uses an XGBoost model. In summary, the main difference between Kernel SHAP and Tree SHAP is that the first represents an *exponential*-time run time, while the latter represents a *polynomial*-time run time, which actually allows for an *exact* solution! The algorithm behind this is very complicated, for further details see Ref. 2.

A key problem with the LIME formulation is that we have to *choose* parameters heuristically (e.g. *how* do we actually define what *local* means?). Moreover, the game theoretic formulation from SHAP suggests *all LIME parameters are forced under local accuracy (additivity) and consistency*! The original parameters from LIME do not recover the Shapley values, but the SHAP developers derive the expressions for the *fairness* properties to be satisfied, the most important one being the local kernel:

$$\pi_{x'}(z') = \frac{(M-1)}{(M choose|z'|)|z'|(M-|z'|)} \tag{5}$$

In other words, authors have actually found a way to compute Shapley values from game theory using weighted linear regression, which is fundamentally different than random sampling from a permutation. Not only this, but results from their paper suggest it is actually a *better* way to calculate them, with a reduced variance compared to traditional methods.

1.3 Convolutional Neural Networks (CNN)

CNN's are a type of deep neural networks and are the most widely used algorithm in image analysis. The basis of a CNN are the convolutional layers, which make use of a linear mathematical operation called convolution. A convolution is, simply put, how the input is modified by filters. The main difference between a convolution layer and a dense layer is that the latter learns global patterns in its global input space, while the convolutional layers learn local patterns in small windows of two dimensions. Intuitively, one could say the goal of convolution layers is to detect visual features. The shallower layers in ConvNets usually detect *simple* patterns such as different geometries (edges, lines) and textures. The deeper these networks go, filters become more sophisticated and are able to identiy more *complex* patterns (e.g. body parts, distinguishing between different types of animals). Of course, ConvNets easily fall into the category of *complex* models that are difficult to interpret.

1.4 XGBoost

XGBoost is a gradient boosting tree-based machine learning algorithm initially developed by Tianqi Chen [7], coincidentally another former member of the research group of Carlos Guestrin from the University of Washington. Gradient boosting methods are learning algorithms that develop predictive models by combining the estimates of a set of simpler, weaker models (decision trees). Over the years XGBoosting has become very popular, particularly due to routinely winning Kaggle competitions.

2 Jupyter Notebook - Dogs vs Wolves

I have shared the notebook (and dataset) via Google Drive, since unfortunately there were some incompatibility issues when uploading it to GitHub since it was done using Google Colab. As mentioned previously, we will be training an image classifier. The dataset was obtained from Kaggle (see Ref. 3), and is composed of 2002 total images of both dogs and wolves already located in different folders, making it a binary classification problem. It is also already partitioned into training (80%) and test (20%) portions, which results in 800/201 total training and testing images for dogs and 800/201 total training and testing images for wolves. In general, the most important libraries used in the notebook are TensorFlow and Keras, which provide the necessary classes and functions to build and train the neural network. Two important things to point out here is that the code was written to be used (in principle) with Google Colab and TensorFlow 2.0.0, which is why some errors might be obtained when running it with newer versions of the latter (other earlier versions might also work as well).

Due to the data type and size of the dataset, the latter was uploaded into my Google Drive separately, which is why the first section of the code consists of loading it from there, authenticating my credentials, unzipping it, storing it, and ensuring it was stored properly. It should be emphasized that images are actually fed into the model in batches using the `ImageDataGenerator class` from Keras. The next section of the code is the data preprocessing. Since we are dealing with pixel data, the normalization is done in a relatively simple way, which is scaling it by dividing it by 255, since it is the maximum value it can achieve.

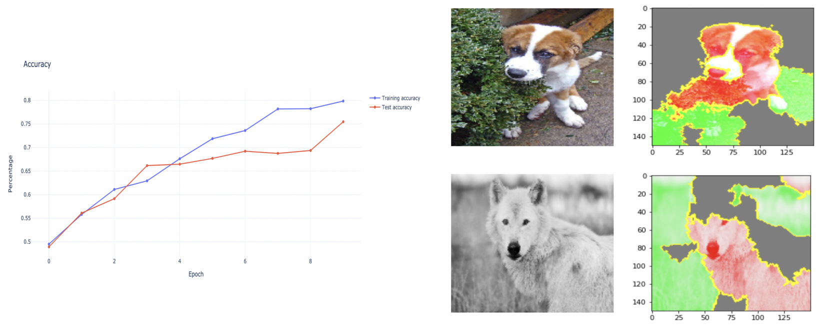
2.1 CNN Setup/Training

Since we will be building an image classifier, a CNN is a natural choice for the model. The proposed network architecture is very simple, since the goal of the project is to focus on the interpretability and not necessarily developing the most accurate model. To summarize, the proposed architecture uses the `Sequential model` from the Keras library with three (3) convolutional layers (`Conv2D class`), each followed by a pooling layer (`MaxPooling2D class`). As usual, the last part of the architecture consists of flattening the network and two (2) additional fully connected layers (using `Flatten` and `Dense` functions from Keras). Additional details of the network include: it uses a rectified linear (ReLU) activation function and for the optimizer, it uses the Adam gradient descent optimizer (`Adam class`), which has been successfully adopted recently in the Deep Learning community. Finally, the performance of the model will be evaluated using accuracy as criterion, followed by individual prediction explanations using LIME, since we will be analyzing images and it is by far the least computationally expensive between the two methods described here.

The results are shown in Figure 2, and are surprisingly good. Obtained training and test validation accuracies are 79.84% and 75.46% respectively, which are arguably *very good* results for image classification, especially with such a simple model architecture. Moreover, the difference between these values is small, suggesting there may be negligible *overfitting*.

2.2 LIME Explanations

The last section of the code consists of a very simple LIME implementation. It can be summarized as installing it via a simple `pip install lime` and importing the `LimeImageExplainer class`, which we then use to explain individual predictions from the test dataset. In addition to this, some preprocessing needs to be done in order to have images in the appropriate format for LIME, such as converting it to a `numpy` array. After this, the `explain_instance` method is called, which in turn needs several arguments to be specified (image as a `numpy` array, the `predict` method, `top_labels` — which really does not matter here because we are doing binary classification, and `num_samples` which we set as the default value of 100 to prevent it from getting too slow).



. Figure 2: Accuracy and LIME explanations.

It was previously pointed out that a 75% test accuracy was obtained, so *naturally* one would expect reasonably good explanations for predictions. However, this is *not* the case! As a first example, we consider the single prediction of a dog that was classified correctly as one (see Figure 2), but the explanation is not convincing. It is simply looking at the grass, and not considering any parts of the image about the dog to classify it as one. In fact, it is even considering the face of the dog as evidence for *not* classifying it as a dog! In contrast, we also consider an image of a wolf that was *incorrectly* classified as a dog. When we look at the explanation, the same thing is observed: the model is simply looking at the background to make the classification, which in the case of this wolf is very similar to the majority of the training data for dogs (grass-like background). Thus a fair conclusion is that, although this is a classifier with a (relatively good) test accuracy of 75%, you should *not* trust it! One could argue this is actually a grass/snow detector and not a husky/wolf classifier.

On the other hand, there is an obvious reason behind this: it was done on purpose to demonstrate the *importance* of interpretability. The original dataset was actually *forced* with around 200 additional images of dogs in the grass obtained from a different dataset, that in a way act as bias sources. A similar experiment was done in the original LIME paper but with a much smaller (and different) dataset. Another evident reason is the simplicity of the CNN architecture — one should not expect such a simple network to be an *extraordinary* image classification model that distinguishes well-enough these images, some of which may be very alike. I hypothesize that a *better* model and a dataset that is not so *inherently biased* could certainly give better results and explanations. But this is obviously beyond the scope of this project, since the present example illustrates the concept well.

3 Jupyter Notebook - Bank Marketing Data Set

Similarly, I have shared the second notebook via Google Drive. On the other hand, this dataset was obtained from the UCI website [4]. In this case the data type is numerical, and it contains 41,188 instances with information about direct marketing campaigns of a Portuguese banking institution. Features contain information such as the client's age, job, marital status, education, duration of cellphone calls, and some others. The goal is to predict whether a client will subscribe a term deposit or not, making it another binary classification task, as in the previous example. Minimal preprocessing is required (e.g. there are no missing values). The most important preprocessing step in this case is the encoding of categorical features, here done using the `ColumnTransfer` from sklearn, which simplifies the application of `OneHotEncoding` to categorical features and leaves numerical ones intact. The dataset is then divided into 80% training and 20% test portions, as was done in the previous section.

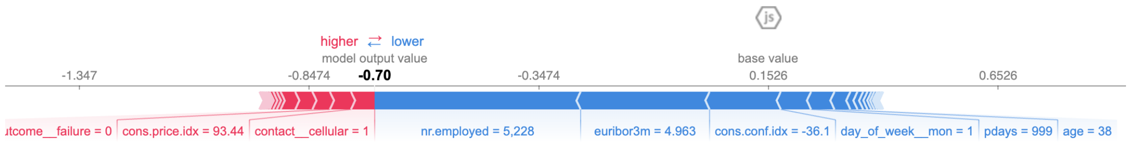
3.1 XGBoost Setup/Training

The model chosen in this case was XGBoost (`XGBClassifier` from sklearn) since, again, in this project we are interested in interpreting the results of *complex* models. There is some slight parameter tuning: we control the depth of the tree with `max_depth` (set equal to the default value of 10) and the minimum sum of weights of all observations required in a child `min_child_weight`, which is set at equal to 1. Another key thing to point out here is that (as shown at the beginning of the notebook) the classes are highly imbalanced: there are 36548 and 4640 instances corresponding to *no* and *yes*. The argument `scale_pos_weight` in the model tries to mitigate this potential bias source, which we set equal to 9 since around 90% of the data belongs to one class. The performance of the classifier is then evaluated using 10-fold cross validation and its predictions are explained using SHAP. The results are very good, since the classifier was found to have 84.74% test accuracy.

3.2 SHAP Explanations

The last section of the notebook consists of the SHAP implementation, which is also very simple. It can be summarized as installing it via a simple `pip install shap` and importing it, while predictions are explained using the `TreeExplainer` method. It should be pointed out that there are several other methods for calculating SHAP values, including a model-agnostic class called `KernelExplainer`, which is obviously the one that relates to

LIME. However, the latter is not used here because it is computationally demanding and only provides an approximation! The `TreeExplainer` method is much more efficient to compute and it is also *exact*, as was explained previously in the Introduction section.



. Figure 3: SHAP explanations.

Results have a relatively simple interpretation. The standard method for visualizing individual model predictions in SHAP is `force_plot`, which is included here in Figure 3. In this case, the horizontal axis represents the SHAP values, and colors represent the effect features have for that particular instance. Pink color represents features that are *pushing* the prediction towards class 1, which in this dataset corresponds to customers that subscribed to a term deposit. Similarly, blue colors represent features that are pushing it towards class 0, i.e. clients that don't subscribe. Another important element of this plot is the base value, which is simply the mean of the model output in the *background* dataset. For the `TreeExplainer` class, this is set as the training set by default. A negative SHAP value (such as in this case, -0.7) will result in a classification of class 0, while a positive SHAP value will result in a classification of class 1. Finally, the second and third plots give an intuition of what is happening *globally* with the model. One of those is actually another `force_plot`, but with a group of 500 clients instead of just one. The last plot is simply a comparison between feature values and how these impact the calculated SHAP values.

Should we trust this classifier? In this case, one would have to perform a more rigorous analysis and perhaps even consult domain knowledge experts, since it is not as simple as classifying a dog or a wolf. However, it certainly seems to be better than the previous case, which was completely biased and very difficult to trust, since here we can see that there is a more reasonable balance between accuracy and explanations. Features such as whether the client was contacted by phone calls and a low Consumer Price Index (CPI) push *in favor* of a client subscribing, as well as higher interest rates pushing it towards not subscribing. All of these make sense when we think about it. On the other hand, explanations suggest, for example, that the number of employees in the bank increasing pushes the clients towards *not* subscribing (perhaps there are too many of them calling clients and are annoying them?). But, in general, the explanations seem to make sense.

4 Conclusion

It has recently been projected that AI could add $15 trillion to the world economy by 2030. On one side, ML and software infrastructure are both expected to continue evolving and improving steadily in the near future, as they have done over the last few decades. So what is *really*, if anything, preventing the industry sector from capitalising on this opportunity? One could argue that a very plausible reason is precisely the black box dilemma. It seems evident that there is still work to be done before we are able to fully adopt machine learning models in many real world applications, particularly those where the need of understanding *why* models predict something is important. A model that is difficult or impossible to interpret inherently creates confusions and doubts, and often represents significant business risks for industry. Thus, in applications where ML becomes a decision-making tool, business owners and stakeholders are increasingly asking what does AI *mean* for them, how can they harness its potential and what are the risks of using it. These questions are often difficult to answer properly in a non-technical manner considering traditional tools: *a CEO will not simply trust the weights or the parameters (which he probably does not understand) of your machine learning model when making a crucial decision involving his company.* Of course, as shown in this project, users themselves also see a lot of benefits from being able to interpret a model: a Data Scientist/IT guy may want to know how can he improve/monitor/debug it, and domain knowledge experts may want to know if these decisions are fair/make sense. Moreover, *society* as a whole has the ethical responsibility of demanding that these systems are operating in line with basic ethical principles, *especially* with the current situations in the world such as in the US (e.g. imagine we had AI systems trained on racist historical data sending people to jail because of their bias!).

However, model interpretability is sometimes overlooked, perhaps because typical machine learning advances and applications have dealt with areas where understanding *why* models make predictions is mostly unimportant. For example, it really does not matter *why* Netflix recommends you a movie you didn't eventually like, or *why* AlphaGo made the legendary *move 37* against Lee Sedol in 2016 (unless you are a professional Go player and want to steal it). At most, the consequences are that you spend a couple of dollars in a product you did not like, or you lose a game. Both are equally trivial. On the other hand, knowing *why* a model makes a particular prediction within a scientific context could make the difference between life and death or perhaps even discovering previously unknown laws of physics! As a scientist, I thus strongly believe that for machine learning to truly revolutionize science and many other daily world applications it has to go much deeper than winning Kaggle competitions with high accuracy models. That being said, explanation methods have definitely aided in partially solving this problem and are being increasingly used in industry. Companies such as Microsoft (in their Cloud pipeline), sports teams such as the Portland Trailblazers, hospitals such as the Cleveland Clinic in Ohio (for cancer subtypes identification), car companies such as Rolls Royce (for optimizing manufacturing processes), banking institutions such as the Bank of England, and possibly thousands of others are already using them extensively to interpret and improve the performance of their current models.

This project focused on discussing arguably the two (2) most well-known methods for the explanation and interpretation of predictions of *complex*, so-called black box models.

Both LIME and SHAP have had very good acceptance within the scientific community, with each of these currently having a total of 2979 and 1036 citations respectively, according to Google Scholar. As was discussed, each of these methods has its own advantages and disadvantages. The general perception is that SHAP is the most robust and *better* method, due to the idea that it represents a *unification* of explanation methods as the paper title suggests. My personal opinion, however, is this will ultimately depend on the particular case study. In addition to this, it is worthwhile mentioning that original authors and many other researchers both in academia and industry are still actively investigating these methods. These contributions will help to further improve them and will surely result in the development of other novel approaches for the interpretation of ML models.

5 References

[1] M. T. Ribeiro, S. Singh, and C. Guestrin. Why should I trust you?: Explaining the predictions of any classifier. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2016, pp. 1135 - 1144.

[2] S. M. Lundberg and S. I. Lee. A unified approach to interpreting model predictions. In: *Proc. Adv. NIPS*. 2017, pp. 4768 - 4777.

[3] https://www.kaggle.com/harishvutukuri/dogs-vs-wolves

[4] https://archive.ics.uci.edu/ml/datasets/bank+marketing

[5] https://github.com/marcotcr/lime

[6] https://github.com/slundberg/shap/

[7] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In: *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785?794. ACM, 2016.

[8] https://towardsdatascience.com/idea-behind-lime-and-shap-b603d35d34eb

[9] Finale Doshi-Velez and Been Kim. 2017. Towards a rigorous science of interpretable machine learning. arXiv:1702.08608v2