PSE Project 1: On the Interpretability of Machine Learning

Chouliaras Andreas
achouliaras@inf.uth.gr

Pappas Apostolos
apopappas@inf.uth.gr

Stylianos Gkountouvas
sgkountouvas@inf.uth.gr

October 2019

1 Introduction

Understanding and trusting machine learning models and their results is a hallmark of good science. Analysts, engineers, physicians, researchers, scientists, and humans in general have the need to understand and trust models and modeling results that affect our work and our lives. For decades, choosing a model that was transparent to human practitioners or consumers often meant choosing straightforward data sources and simpler model forms such as linear models, single decision trees, or business rule systems. Although these simpler approaches were often the correct choice, and still are today, they can fail in real-world scenarios when the underlying modeled phenomena are nonlinear, rare or faint, or highly specific to certain individuals. Today, the trade-off between the accuracy and interpretability of predictive models has been broken. The tools now exist to build accurate and sophisticated modeling systems based on heterogeneous data and machine learning algorithms that will enable humans understand and trust these complex systems.

2 What is Machine Learning?

Machine learning is a set of methods that computers use to make and improve predictions or behaviors based on data.

Estimation of house prices, product recommendations, street sign detection, credit default prediction and fraud detection: All these examples have in common that they can be solved by machine learning. The tasks are different, but the approach is the same:

Step 1: Data collection. The more, the better. The data must contain the outcome you want to predict and additional information from which to make the prediction. For a street sign detector ("Is there a street sign in the image?"), you would collect street images and label whether a street sign is visible or not. For a credit default predictor, you need past data on actual loans, information on whether the customers were in default with their loans, and data that will help you make predictions, such as income, past credit defaults, and so on. For an automatic house value estimator program, you could collect data from past house sales and information about the real estate such as size, location, and so on.

Step 2: Enter this information into a machine learning algorithm that generates a sign detector model, a credit rating model or a house value estimator.

Step 3: Use model with new data. Integrate the model into a product or process, such as a self-driving car, a credit application process or a real estate marketplace website.

2.1 What is their advantage?

Machines surpass humans in many tasks, such as playing chess or predicting the weather. Even if the machine is as good as a human or a bit worse at a task, there remain great advantages in terms of speed, reproducibility and scaling. A once implemented machine learning model can complete a task much faster than humans, reliably delivers consistent results and can be copied infinitely. Replicating a machine learning model on another machine is fast and cheap. The training of a human for a task can take decades (especially when they are young) and is very costly.

2.2 But every rose has its thorns...

A major disadvantage of using machine learning is that insights about the data and the task the machine solves is hidden in increasingly complex models. You need millions of numbers to describe a deep neural network, and there is no way to understand the model in its entirety. Other models, such as the random forest, consist of hundreds of decision trees that “vote” for predictions. To understand how the decision was made, you would have to look into the votes and structures of each of the hundreds of trees. That just does not work no matter how clever you are or how good your working memory is. The best performing models are often blends of several models (also called ensembles) that cannot be interpreted, even if each single model could be interpreted. If you focus only on performance, you will automatically get more and more opaque models.

3 The Importance of Interpretability

If a machine learning model performs well, why do not we just trust the model and ignore why it made a certain decision? The problem is that a single metric, such as classification accuracy, is an incomplete description of most real-world tasks.

When it comes to predictive modeling, you have to make a trade-off: Do you just want to know what is predicted,or do you want to know why the prediction was made and possibly pay for the interpretability with a drop in predictive performance? In some cases, you do not care why a decision was made, it is enough to know that the predictive performance on a test dataset was good. But in other cases, knowing the 'why' can help you learn more about the problem, the data and the reason why a model might fail.

4 Taxonomy of Interpretability Methods

Methods for machine learning interpretability can be classified according to various criteria.

4.1 Intrinsic or post hoc?

This criteria distinguishes whether interpretability is achieved by restricting the complexity of the machine learning model (intrinsic) or by applying methods that analyze the model after training (post hoc). Intrinsic interpretability refers to machine learning models that are considered interpretable due to their simple structure, such as short decision trees or sparse linear models. Post hoc interpretability refers to the application of interpretation methods after model training.

4.2 Model-specific or model-agnostic?

Model-specific interpretation tools are limited to specific model classes. The interpretation of regression weights in a linear model is a model-specific interpretation, since – by definition – the interpretation of intrinsically interpretable models is always model-specific. Tools that only work for the interpretation of e.g. neural networks are model-specific. Model-agnostic tools can be used on any machine learning model and are applied after the model has been trained (post hoc). These agnostic methods usually work by analyzing feature input and output pairs. By definition, these methods cannot have access to model internals such as weights or structural information.

4.3 Local or global?

Does the interpretation method explain an individual prediction or the entire model behavior? Or is the scope somewhere in between?

5 Scope of Interpretability

An algorithm trains a model that produces the predictions. Each step can be evaluated in terms of transparency or interpretability.

5.1 Algorithm Transparency

How does the algorithm create the model?

Algorithm transparency is about how the algorithm learns a model from the data and what kind of relationships it can learn. If you use convolutional neural networks to classify images, you can explain that the algorithm learns edge detectors and filters on the lowest layers. This is an understanding of how the algorithm works, but not for the specific model that is learned in the end, and not for how individual predictions are made. Algorithm transparency only requires knowledge of the algorithm and not of the data or learned model.

5.2 Global, Holistic Model Interpretability

How does the trained model make predictions?

You could describe a model as interpretable if you can comprehend the entire model at once [1]. To explain the global model output, you need the trained model, knowledge of the algorithm and the data. This level of interpretability is about understanding how the model makes decisions, based on a holistic view of its features and each of the learned components such as weights, other parameters, and structures. Which features are important and what kind of interactions between them take place? Global model interpretability helps to understand the distribution of your target outcome based on the features. Global model interpretability is very difficult to achieve in practice. Any model that exceeds a handful of parameters or weights is unlikely to fit into the short-term memory of the average human. It is debatable whether we can imagine a linear model with 5 features, because it would mean drawing the estimated hyperplane mentally in a 5-dimensional space. Any feature space with more than 3 dimensions is simply inconceivable for humans. Usually, when people try to comprehend a model, they consider only parts of it, such as the weights in linear models.

5.3 Global Model Interpretability on a Modular Level

How do parts of the model affect predictions?

A Naive Bayes model with many hundreds of features would be too big to keep in our working memory. And even if we manage to memorize all the weights, we would not be able to quickly make predictions for new data points. In addition, we need to have the joint distribution of all features in our head to estimate the importance of each feature and how the features affect the predictions on average. An impossible task. But we can easily understand a single weight. While global model interpretability is usually out of reach, there is a good chance of understanding at least some models on a modular level.

Not all models are interpretable at a parameter level. For linear models, the interpretable parts are the weights, for trees it would be the splits (selected features plus cut-off points) and leaf node predictions. Linear models, for example, look like as if they could be perfectly interpreted on a modular level, but the interpretation of a single weight is interlocked with all other weights. The interpretation of a single weight always comes with the footnote that the other input features remain at the same value, which is not the case with many real applications.

5.4 Local Interpretability for a Single Prediction

Why did the model make a certain prediction for an instance?

You can zoom in on a single instance and examine what the model predicts for this input, and explain why. If you look at an individual prediction, the behavior of the otherwise complex model might behave more pleasantly. Locally, the prediction might only depend linearly or monotonically on some features, rather than having a complex dependence on them.

5.5 Local Interpretability for a Group of Predictions

Why did the model make specific predictions for a group of instances?

Model predictions for multiple instances can be explained either with global model interpretation methods (on a modular level) or with explanations of individual instances. The global methods can be applied by taking the group of instances, treating them as if the group were the complete dataset, and using the global methods with this subset. The individual explanation methods can be used on each instance and then listed or aggregated for the entire group.

6 Interpretable Models

The easiest way to achieve interpretability is to use only a subset of algorithms that create interpretable models. Linear regression, logistic regression and the decision tree are commonly used interpretable models.

All interpretable models explained here are interpretable on a modular level, with the exception of the k-nearest neighbors method. The following table gives an overview of the interpretable model types and their properties. A model is linear if the association between features and target is modelled linearly. A model with monotonicity constraints ensures that the relationship between a feature and the target outcome always goes in the same direction over the entire range of the feature: An increase in the feature value either always leads to an increase or always to a decrease in the target outcome. Monotonicity is useful for the interpretation of a model because it makes it easier to understand a relationship.

Some models can automatically include interactions between features to predict the target outcome. You can include interactions in any type of model by manually creating interaction features. Interactions can improve predictive performance, but too many or too complex interactions can hurt interpretability. Some models handle only regression, some only classification, and still others both.

From this table, you can select a suitable interpretable model for your task, either regression (regr) or classification (class):

Algorithm	Linear	Monotone	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
Naive Bayes	No	Yes	No	class
k-nearest neighbors	No	No	No	class,reg

6.1 Linear Regression

A linear regression model predicts the target as a weighted sum of the feature inputs. The linearity of the learned relationship makes the interpretation easy. Linear regression models have long been used by statisticians, computer scientists and other people who tackle quantitative problems.

Linear models can be used to model the dependence of a regression target y on some features x.The learned relationships are linear and can be written for a single instance i as follows:

$$y = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p + \epsilon$$

The predicted outcome of an instance is a weighted sum of its p features. The betas (β_j) represent the learned feature weights or coefficients. The first weight in the sum (β_0) is called the intercept and is not multiplied with a feature. The epsilon (β_ϵ) is the error we still make, i.e. the difference between the prediction and the actual outcome. These errors are assumed to follow a Gaussian distribution, which means that we make errors in both negative and positive directions and make many small errors and few large errors.

Various methods can be used to estimate the optimal weight. The ordinary least squares method is usually used to find the weights that minimize the squared differences between the actual and the estimated outcomes:

$$\hat{\boldsymbol{\beta}} = \arg\min_{\beta_0,\ldots,\beta_p} \sum_{i=1}^{n} \left(y^{(i)} - \left(\beta_0 + \sum_{j=1}^{p} \beta_j x_j^{(i)} \right) \right)^2$$

6.1.1 Advantages

The modeling of the predictions as a weighted sum makes it transparent how predictions are produced. And with Lasso we can ensure that the number of features used remains small.

Many people use linear regression models. This means that in many places it is accepted for predictive modeling and doing inference. There is a high level of collective experience and expertise, including teaching materials on linear regression models and software implementations. Linear regression can be found in R, Python, Java, Julia, Scala, Javascript, ...

Mathematically, it is straightforward to estimate the weights and you have a guarantee to find optimal weights (given all assumptions of the linear regression model are met by the data).

The biggest advantage of linear regression models is linearity: It makes the estimation procedure simple and, most importantly, these linear equations have an easy to understand interpretation on a modular level (i.e. the weights). This is one of the main reasons why the linear model and all similar models are so widespread in academic fields such as medicine, sociology, psychology, and many other quantitative research fields.

Estimated weights come with confidence intervals. A confidence interval is a range for the weight estimate that covers the “true” weight with a certain confidence. For example, a 95% confidence interval for a weight of 2 could range from 1 to 3. The interpretation of this interval would be: If we repeated the estimation 100 times with newly sampled data, the confidence interval would include the true weight in 95 out of 100 cases, given that the linear regression model is the correct model for the data.

6.1.2 Disadvantages

Linear regression models can only represent linear relationships, i.e. a weighted sum of the input features. Each nonlinearity or interaction has to be hand-crafted and explicitly given to the model as an input feature.

Linear models are also often not that good regarding predictive performance, because the relationships that can be learned are so restricted and usually over-simplify how complex reality is.

The interpretation of a weight can be unintuitive because it depends on all other features. A feature with high positive correlation with the outcome y and another feature might get a negative weight in the linear model, because, given the other correlated feature, it is negatively correlated with y in the high-dimensional space. Completely correlated features make it even impossible to find a unique solution for the linear equation.

6.2 Logistic Regression

Logistic regression models the probabilities for classification problems with two possible outcomes. It’s an extension of the linear regression model for classification problems.

The linear regression model can work well for regression, but fails for classification. Why is that? In case of two classes, you could label one of the classes with 0 and the other with 1 and use linear regression. Technically it works and most linear model programs will spit out weights for you. But there are a few problems with this approach:

A linear model does not output probabilities, but it treats the classes as numbers (0 and 1) and fits the best hyperplane (for a single feature, it is a line) that minimizes the distances between the points and the hyperplane. So it simply interpolates between the points, and you cannot interpret it as probabilities.

A linear model also extrapolates and gives you values below zero and above one. This is a good sign that there might be a smarter approach to classification.

Since the predicted outcome is not a probability, but a linear interpolation between points, there is no meaningful threshold at which you can distinguish one class from the other.

A solution for classification is logistic regression. Instead of fitting a straight line or hyperplane, the logistic regression model uses the logistic function to squeeze the output of a linear equation between 0 and 1. The logistic function is defined as:

$$\text{logistic}(\eta) = \frac{1}{1 + exp(-\eta)}$$

And it looks like this:

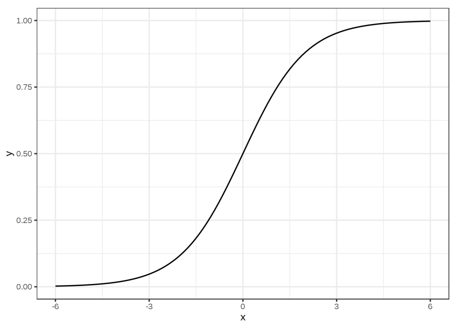

Figure 1: The logistic function. It outputs numbers between 0 and 1. At input 0, it outputs 0.5.

The step from linear regression to logistic regression is kind of straightforward. In the linear regression model, we have modelled the relationship between outcome and features with a linear equation:

$$\hat{y}^{(i)} = \beta_0 + \beta_1 x_1^{(i)} + \ldots + \beta_p x_p^{(i)}$$

For classification, we prefer probabilities between 0 and 1, so we wrap the right side of the equation into the logistic function. This forces the output to assume only values between 0 and 1:

$$P(y^{(i)} = 1) = \frac{1}{1 + exp(-(\beta_0 + \beta_1 x_1^{(i)} + \ldots + \beta_p x_p^{(i)}))}$$

6.2.1 Advantages and Disadvantages

Many of the pros and cons of the linear regression model also apply to the logistic regression model. Logistic regression has been widely used by many different people, but it struggles with its restrictive expressiveness (e.g. interactions must be added manually) and other models may have better predictive performance.

Another disadvantage of the logistic regression model is that the interpretation is more difficult because the interpretation of the weights is multiplicative and not additive.

Logistic regression can suffer from complete separation. If there is a feature that would perfectly separate the two classes, the logistic regression model can no longer be trained. This is because the weight for that feature would not converge, because the optimal weight would be infinite. This is really a bit unfortunate, because such a feature is really useful. But you do not need machine learning if you have a simple rule that separates both classes. The problem of complete separation can be solved by introducing penalization of the weights or defining a prior probability distribution of weights.

On the good side, the logistic regression model is not only a classification model, but also gives you probabilities. This is a big advantage over models that can only provide the final classification. Knowing that an instance has a 99% probability for a class compared to 51% makes a big difference.

Logistic regression can also be extended from binary classification to multiclass classification. Then it is called Multinomial Regression.

6.3 Decision trees

Linear regression and logistic regression models fail in situations where the relationship between features and outcome is nonlinear or where features interact with each other. Time to shine for the decision tree! Tree based models split the data multiple times according to certain cutoff values in the features. Through splitting, different subsets of the dataset are created, with each instance belonging to one subset. The final subsets are called terminal or leaf nodes and the intermediate subsets are called internal nodes or split nodes. To predict the outcome in each leaf node, the average outcome of the training data in this node is used. Trees can be used for classification and regression.

There are various algorithms that can grow a tree. They differ in the possible structure of the tree (e.g. number of splits per node), the criteria how to find the splits, when to stop splitting and how to estimate the simple models within the leaf nodes.

The interpretation is simple: Starting from the root node, you go to the next nodes and the edges tell you which subsets you are looking at. Once you reach the leaf node, the node tells you the predicted outcome. All the edges are connected by 'AND'.

Template: If feature x is [smaller/bigger] than threshold c AND ... then the predicted outcome is the mean value of y of the instances in that node.

The overall importance of a feature in a decision tree can be computed in the following way: Go through all the splits for which the feature was used and measure how much it has reduced the variance or Gini index compared to the parent node. The sum of all importances is scaled to 100. This means that each importance can be interpreted as share of the overall model importance.

6.3.1 Advantages

The tree structure is ideal for capturing interactions between features in the data.

The data ends up in distinct groups that are often easier to understand than points on a multi-dimensional hyperplane as in linear regression. The interpretation is arguably pretty simple.

The tree structure also has a natural visualization, with its nodes and edges.

Trees create good explanations.They automatically invite to think about predicted values for individual instances as counterfactuals: "If a feature had been greater / smaller than the split point, the prediction would have been y1 instead of y2. The tree explanations are contrastive, since you can always compare the prediction of an instance with relevant"what if"-scenarios (as defined by the tree) that are simply the other leaf nodes of the tree. If the tree is short, like one to three splits deep, the resulting explanations are selective. A tree with a depth of three requires a maximum of three features and split points to create the explanation for the prediction of an individual instance. The truthfulness of the prediction depends on the predictive performance of the tree. The explanations for short trees are very simple and general, because for each split the instance falls into either one or the other leaf, and binary decisions are easy to understand.

6.3.2 Disadvantages

Trees fail to deal with linear relationships. Any linear relationship between an input feature and the outcome has to be approximated by splits, creating a step function. This is not efficient.

This goes hand in hand with lack of smoothness. Slight changes in the input feature can have a big impact on the predicted outcome, which is usually not desirable.

Trees are also quite unstable. A few changes in the training dataset can create a completely different tree. This is because each split depends on the parent split. And if a different feature is selected as the first split feature, the entire tree structure changes. It does not create confidence in the model if the structure changes so easily.

Decision trees are very interpretable – as long as they are short. The number of terminal nodes increases quickly with depth. The more terminal nodes and the deeper the tree, the more difficult it becomes to understand the decision rules of a tree. A depth of 1 means 2 terminal nodes. Depth of 2 means max. 4 nodes. Depth of 3 means max. 8 nodes. The maximum number of terminal nodes in a tree is 2 to the power of the depth.

6.4 RuleFit

The RuleFit algorithm learns sparse linear models that include automatically detected interaction effects in the form of decision rules.

The linear regression model does not account for interactions between features. Would it not be convenient to have a model that is as simple and interpretable as linear models, but also integrates feature interactions? RuleFit fills this gap. RuleFit learns a sparse linear model with the original features and also a number of new features that are decision rules. These new features capture interactions between the original features. RuleFit automatically generates these features from decision trees. Each path through a tree can be transformed into a decision rule by combining the split decisions into a rule. The node predictions are discarded and only the splits are used in the decision rules:

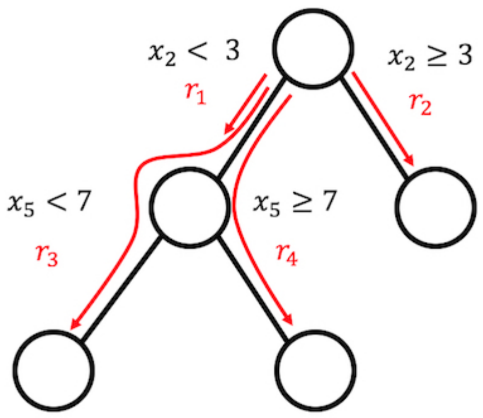

Figure 2: Rules can be generated from a tree with 3 terminal nodes.

Where do those decision trees come from? The trees are trained to predict the outcome of interest. This ensures that the splits are meaningful for the prediction task. Any algorithm that generates a lot of trees can be used for RuleFit, for example a random forest. Each tree is decomposed into decision rules that are used as additional features in a sparse linear regression model (Lasso).

RuleFit also comes with a feature importance measure that helps to identify linear terms and rules that are important for the predictions. Feature importance is calculated from the weights of the regression model. The importance measure can be aggregated for the original features (which are used in their "raw" form and possibly in many decision rules).

RuleFit also introduces partial dependence plots to show the average change in prediction by changing a feature. The partial dependence plot is a model-agnostic method that can be used with any model, and is explained in the chapter: Partial dependence plots.

6.4.1 Advantages

RuleFit automatically adds feature interactions to linear models. Therefore, it solves the problem of linear models that you have to add interaction terms manually and it helps a bit with the issue of modeling nonlinear relationships.

RuleFit can handle both classification and regression tasks.

The rules created are easy to interpret, because they are binary decision rules. Either the rule applies to an instance or not. Good interpretability is only guaranteed if the number of conditions within a rule is not too large. A rule with 1 to 3 conditions seems reasonable to me. This means a maximum depth of 3 for the trees in the tree ensemble.

Even if there are many rules in the model, they do not apply to every instance. For an individual instance only a handful of rules apply (= have a non-zero weights). This improves local interpretability.

6.4.2 Disadvantages

Sometimes RuleFit creates many rules that get a non-zero weight in the Lasso model. The interpretability degrades with increasing number of features in the model. A promising solution is to force feature effects to be monotonic, meaning that an increase of a feature has to lead to an increase of the prediction.

The end product of the RuleFit procedure is a linear model with additional fancy features (the decision rules). But since it is a linear model, the weight interpretation is still unintuitive. It comes with the same “footnote” as a usual linear regression model: “... given all features are fixed.” It gets a bit more tricky when you have overlapping rules.

6.5 Naive Bayes Classifier

The Naive Bayes classifier uses the Bayes’ theorem of conditional probabilities. For each feature, it calculates the probability for a class depending on the value of the feature. The Naive Bayes classifier calculates the class probabilities for each feature independently, which is equivalent to a strong (= naive) assumption of independence of the features. Naive Bayes is a conditional probability model and models the probability of a class C_k as follows:

$$P(C_k|x) = \frac{1}{Z}P(C_k)\prod_{i=1}^{n}P(x_i|C_k)$$

The term Z is a scaling parameter that ensures that the sum of probabilities for all classes is 1 (otherwise they would not be probabilities). The conditional probability of a class is the class probability times the probability of each feature given the class, normalized by Z. This formula can be derived by using the Bayes’ theorem.

Naive Bayes is an interpretable model because of the independence assumption. It can be interpreted on the modular level. It is very clear for each feature how much it contributes towards a certain class prediction, since we can interpret the conditional probability.

6.6 K-Nearest Neighbors

The k-nearest neighbor method can be used for regression and classification and uses the nearest neighbors of a data point for prediction. For classification, the k-nearest neighbor method assigns the most common class of the nearest neighbors of an instance. For regression, it takes the average of the outcome of the neighbors. The tricky parts are finding the right k and deciding how to measure the distance between instances, which ultimately defines the neighborhood.

The k-nearest neighbor model differs from the other interpretable models presented because it is an instance-based learning algorithm. How can k-nearest neighbors be interpreted? First of all, there are no parameters to learn, so there is no interpretability on a modular level. Furthermore, there is a lack of global model interpretability because the model is inherently local and there are no global weights or structures explicitly learned.

Maybe it is interpretable at the local level? To explain a prediction, you can always retrieve the k neighbors that were used for the prediction. Whether the model is interpretable depends solely on the question whether you can 'interpret' a single instance in the dataset. If an instance consists of hundreds or thousands of features, then it is not interpretable. But if you have few features or a way to reduce your instance to the most important features, presenting the k-nearest neighbors can give you good explanations.

7 Model-Agnostic Methods

Separating the explanations from the machine learning model (= model-agnostic interpretation methods) has some advantages. The great advantage of model-agnostic interpretation methods over model-specific ones is their flexibility. Machine learning developers are free to use any machine learning model they like when the interpretation methods can be applied to any model. Anything that builds on an interpretation of a machine learning model, such as a graphic or user interface, also becomes independent of the underlying machine learning model. Typically, not just one, but many types of machine learning models are evaluated to solve a task, and when comparing models in terms of interpretability, it is easier to work with model-agnostic explanations, because the same method can be used for any type of model.

7.1 Partial Dependence Plot (PDP)

The partial dependence plot (short PDP or PD plot) shows the marginal effect one or two features have on the predicted outcome of a machine learning model. A partial dependence plot can show whether the relationship between the target and a feature is linear, monotonic or more complex. For example, when applied to a linear regression model, partial dependence plots always show a linear relationship.

7.1.1 Advantages

The computation of partial dependence plots is intuitive: The partial dependence function at a particular feature value represents the average prediction if we force all data points to assume that feature value. In my experience, lay people usually understand the idea of PDPs quickly.

If the feature for which you computed the PDP is not correlated with the other features, then the PDPs perfectly represent how the feature influences the prediction on average. In the uncorrelated case, the interpretation is clear: The partial dependence plot shows how the average prediction in your dataset changes when the j-th feature is changed. It is more complicated when features are correlated, see also disadvantages.

Partial dependence plots are easy to implement.

The calculation for the partial dependence plots has a causal interpretation. We intervene on a feature and measure the changes in the predictions. In doing so, we analyze the causal relationship between the feature and the prediction. The relationship is causal for the model – because we explicitly model the outcome as a function of the features – but not necessarily for the real world!

7.1.2 Disadvantages

The realistic maximum number of features in a partial dependence function is two. This is not the fault of PDPs, but of the 2-dimensional representation (paper or screen) and also of our inability to imagine more than 3 dimensions.

Some PD plots do not show the feature distribution. Omitting the distribution can be misleading, because you might overinterpret regions with almost no data. This problem is easily solved by showing a rug (indicators for data points on the x-axis) or a histogram.

The assumption of independence is the biggest issue with PD plots. It is assumed that the feature(s) for which the partial dependence is computed are not correlated with other features.

Heterogeneous effects might be hidden because PD plots only show the average marginal effects. Suppose that for a feature half your data points have a positive association with the prediction – the larger the feature value the larger the prediction – and the other half has a negative association – the smaller the feature value the larger the prediction. The PD curve could be a horizontal line, since the effects of both halves of the dataset could cancel each other out. You then conclude that the feature has no effect on the prediction. By plotting the individual conditional expectation curves instead of the aggregated line, we can uncover heterogeneous effects.

7.2 Individual Conditional Expectation (ICE)

Individual Conditional Expectation (ICE) plots display one line per instance that shows how the instance's prediction changes when a feature changes.

The partial dependence plot for the average effect of a feature is a global method because it does not focus on specific instances, but on an overall average. The equivalent to a PDP for individual data instances is called individual conditional expectation (ICE) plot. An ICE plot visualizes the dependence of the prediction on a feature for each instance separately, resulting in one line per instance, compared to one line overall in partial dependence plots. A PDP is the average of the lines of an ICE plot. The values for a line (and one instance) can be computed by keeping all other features the same, creating variants of this instance by replacing the feature's value with values from a grid and making predictions with the black box model for these newly created instances. The result is a set of points for an instance with the feature value from the grid and the respective predictions.

What is the point of looking at individual expectations instead of partial dependencies? Partial dependence plots can obscure a heterogeneous relationship created by interactions. PDPs can show you what the average relationship between a feature and the prediction looks like. This only works well if the interactions between the features for which the PDP is calculated and the other features are weak. In case of interactions, the ICE plot will provide much more insight.

There is a problem with ICE plots: Sometimes it can be hard to tell whether the ICE curves differ between individuals because they start at different predictions. A simple solution is to center the curves at a certain point in the feature and display only the difference in the prediction to this point. The resulting plot is called centered ICE plot (c-ICE).

Another way to make it visually easier to spot heterogeneity is to look at the individual derivatives of the prediction function with respect to a feature. The resulting plot is called the derivative ICE plot (d-ICE). The derivatives of a function (or curve) tell you whether changes occur and in which direction they occur. With the derivative ICE plot, it is easy to spot ranges of feature values where the black box predictions change for (at least some) instances.

7.2.1 Advantages

Individual conditional expectation curves are even more intuitive to understand than partial dependence plots. One line represents the predictions for one instance if we vary the feature of interest.

Unlike partial dependence plots, ICE curves can uncover heterogeneous relationships.

7.2.2 Disadvantages

ICE curves can only display one feature meaningfully, because two features would require the drawing of several overlaying surfaces and you would not see anything in the plot.

ICE curves suffer from the same problem as PDPs: If the feature of interest is correlated with the other features, then some points in the lines might be invalid data points according to the joint feature distribution.

If many ICE curves are drawn, the plot can become overcrowded and you will not see anything. The solution: Either add some transparency to the lines or draw only a sample of the lines.

In ICE plots it might not be easy to see the average. This has a simple solution: Combine individual conditional expectation curves with the partial dependence plot.

7.3 Accumulated Local Effects (ALE) Plot

Accumulated local effects describe how features influence the prediction of a machine learning model on average. ALE plots are a faster and unbiased alternative to partial dependence plots (PDPs).

If features of a machine learning model are correlated, the partial dependence plot cannot be trusted. The computation of a partial dependence plot for a feature that is strongly correlated with other features involves averaging predictions of artificial data instances that are unlikely in reality. This can greatly bias the estimated feature effect.

What can we do to get a feature effect estimate that respects the correlation of the features? We could average over the conditional distribution of the feature, meaning at a grid value of x1, we average the predictions of instances with a similar x1 value. The solution for calculating feature effects using the conditional distribution is called Marginal Plots, or M-Plots (confusing name, since they are based on the conditional, not the marginal distribution). Wait, did I not promise you to talk about ALE plots? M-Plots are not the solution we are looking for.

M-Plots avoid averaging predictions of unlikely data instances, but they mix the effect of a feature with the effects of all correlated features. ALE plots solve this problem by calculating – also based on the conditional distribution of the features – differences in predictions instead of averages.

To summarize how each type of plot (PDP, M, ALE) calculates the effect of a feature at a certain grid value v: Partial Dependence Plots: "Let me show you what the model predicts on average when each data instance has the value v for that feature. I ignore whether the value v makes sense for all data instances." M-Plots: "Let me show you what the model predicts on average for data instances that have values close to v for that feature. The effect could be due to that feature, but also due to correlated features." ALE plots: "Let me show you how the model predictions change in a small"window" of the feature around v for data instances in that window."

ALE plots can also show the interaction effect of two features. The calculation principles are the same as for a single feature, but we work with rectangular cells instead of intervals, because we have to accumulate the effects in two dimensions. In addition to adjusting for the overall mean effect, we also adjust for the main effects of both features. This means that ALE for two features estimate the second-order effect, which does not include the main effects of the features. In other words, ALE for two features only shows the additional interaction effect of the two features.

7.3.1 Advantages

ALE plots are unbiased, which means they still work when features are correlated. Partial dependence plots fail in this scenario because they marginalize over unlikely or even physically impossible combinations of feature values.

ALE plots are faster to compute than PDPs and scale with O(n), since the largest possible number of intervals is the number of instances with one interval per instance. The PDP requires n times the number of grid points estimations. For 20 grid points, PDPs require 20 times more predictions than the worst case ALE plot where as many intervals as instances are used.

The interpretation of ALE plots is clear: Conditional on a given value, the relative effect of changing the feature on the prediction can be read from the ALE plot. ALE plots are centered at zero. This makes their interpretation nice, because the value at each point of the ALE curve is the difference to the mean prediction. The 2D ALE plot only shows the interaction: If two features do not interact, the plot shows nothing.

All in all, in most situations I would prefer ALE plots over PDPs, because features are usually correlated to some extent.

7.3.2 Disadvantages

ALE plots can become a bit shaky (many small ups and downs) with a high number of intervals. In this case, reducing the number of intervals makes the estimates more stable, but also smoothes out and hides some of the true complexity of the prediction model. There is no perfect solution for setting the number of intervals. If the number is too small, the ALE plots might not be very accurate. If the number is too high, the curve can become shaky.

Unlike PDPs, ALE plots are not accompanied by ICE curves. For PDPs, ICE curves are great because they can reveal heterogeneity in the feature effect, which means that the effect of a feature looks different for subsets of the data. For ALE plots you can only check per interval whether the effect is different between the instances, but each interval has different instances so it is not the same as ICE curves.

Second-order ALE estimates have a varying stability across the feature space, which is not visualized in any way. The reason for this is that each estimation of a local effect in a cell uses a different number of data instances. As a result, all estimates have a different accuracy (but they are still the best possible estimates). The problem exists in a less severe version for main effect ALE plots. The number of instances is the same in all intervals, thanks to the use of quantiles as grid, but in some areas there will be many short intervals and the ALE curve will consist of many more estimates. But for long intervals, which can make up a big part of the entire curve, there are comparatively fewer instances. This happened in the cervical cancer prediction ALE plot for high age for example.

Second-order effect plots can be a bit annoying to interpret, as you always have to keep the main effects in mind. It is tempting to read the heat maps as the total effect of the two features, but it is only the additional effect of the interaction. The pure second-order effect is interesting for discovering and exploring interactions, but for interpreting what the effect looks like, I think it makes more sense to integrate the main effects into the plot.

The implementation of ALE plots is much more complex and less intuitive compared to partial dependence plots.

Even though ALE plots are not biased in case of correlated features, interpretation remains difficult when features are strongly correlated. Because if they have a very strong correlation, it only makes sense to analyze the effect of changing both features together and not in isolation. This disadvantage is not specific to ALE plots, but a general problem of strongly correlated features.

If the features are uncorrelated and computation time is not a problem, PDPs are slightly preferable because they are easier to understand and can be plotted along with ICE curves.

The list of disadvantages has become quite long, but do not be fooled by the number of words we use: As a rule of thumb: Use ALE instead of PDP.

7.4 Feature Interaction

When features interact with each other in a prediction model, the prediction cannot be expressed as the sum of the feature effects, because the effect of one feature depends on the value of the other feature. Aristotle's predicate "The whole is greater than the sum of its parts" applies in the presence of interactions.

If a machine learning model makes a prediction based on two features, we can decompose the prediction into four terms: a constant term, a term for the first feature, a term for the second feature and a term for the interaction between the two features. The interaction between two features is the change in the prediction that occurs by varying the features after considering the individual feature effects.

This is implemented using Friedman's H-statistic theory.

7.4.1 Advantages

The interaction H-statistic has an underlying theory through the partial dependence decomposition.

The H-statistic has a meaningful interpretation: The interaction is defined as the share of variance that is explained by the interaction.

Since the statistic is dimensionless, it is comparable across features and even across models.

The statistic detects all kinds of interactions, regardless of their particular form.

With the H-statistic it is also possible to analyze arbitrary higher interactions such as the interaction strength between 3 or more features.

7.4.2 Disadvantages

The first thing you will notice: The interaction H-statistic takes a long time to compute, because it is computationally expensive.

The computation involves estimating marginal distributions. These estimates also have a certain variance if we do not use all data points. This means that as we sample points, the estimates also vary from run to run and the results can be unstable. I recommend repeating the H-statistic computation a few times to see if you have enough data to get a stable result.

It is unclear whether an interaction is significantly greater than 0. We would need to conduct a statistical test, but this test is not (yet) available in a model-agnostic version.

Concerning the test problem, it is difficult to say when the H-statistic is large enough for us to consider an interaction "strong".

Also, the H-statistics can be larger than 0, which makes the interpretation difficult.

The H-statistic tells us the strength of interactions, but it does not tell us how the interactions look like. That is what partial dependence plots are for. A meaningful workflow is to measure the interaction strengths and then create 2D-partial dependence plots for the interactions you are interested in.

The H-statistic cannot be used meaningfully if the inputs are pixels. So the technique is not useful for image classifier.

The interaction statistic works under the assumption that we can shuffle features independently. If the features correlate strongly, the assumption is violated and we integrate over feature combinations that are very unlikely in reality. That is the same problem that partial dependence plots have. You cannot say in general if it leads to overestimation or underestimation.

Sometimes the results are strange and for small simulations do not yield the expected results. But this is more of an anecdotal observation.

7.5 Permutation Feature Importance

Permutation feature importance measures the increase in the prediction error of the model after we permuted the feature's values, which breaks the relationship between the feature and the true outcome.

The concept is really straightforward: We measure the importance of a feature by calculating the increase in the model's prediction error after permuting the 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 for the prediction. The permutation feature importance measurement was introduced by Breiman (2001) for random forests. Based on this idea, Fisher, Rudin, and Dominici (2018) proposed a model-agnostic version of the feature importance and called it model reliance. They also introduced more advanced ideas about feature importance, for example a (model-specific) version that takes into account that many prediction models may predict the data well. Their paper is worth reading.

7.5.1 Advantages

Nice interpretation: Feature importance is the increase in model error when the feature's information is destroyed.

Feature importance provides a highly compressed, global insight into the model's behavior.

A positive aspect of using the error ratio instead of the error difference is that the feature importance measurements are comparable across different problems.

The importance measure automatically takes into account all interactions with other features. By permuting the feature you also destroy the interaction effects with other features. This means that the permutation feature importance takes into account both the main feature effect and the interaction effects on model performance. This is also a disadvantage because the importance of the interaction between two features is included in the importance measurements of both features. This means that the feature importances do not add up to the total drop in performance, but the sum is larger. Only if there is no interaction between the features, as in a linear model, the importances add up approximately.

Permutation feature importance does not require retraining the model.

7.5.2 Disadvantages

It is very unclear whether you should use training or test data to compute the feature importance.

Permutation feature importance is linked to the error of the model. This is not inherently bad, but in some cases not what you need. In some cases, you might prefer to know how much the model's output varies for a feature without considering what it means for performance.

You need access to the true outcome. If someone only provides you with the model and unlabeled data – but not the true outcome – you cannot compute the permutation feature importance.

The permutation feature importance depends on shuffling the feature, which adds randomness to the measurement. When the permutation is repeated, the results might vary greatly. Repeating the permutation and averaging the importance measures over repetitions stabilizes the measure, but increases the time of computation.

If features are correlated, the permutation feature importance can be biased by unrealistic data instances. The problem is the same as with partial dependence plots: The permutation of features produces unlikely data instances when two or more features are correlated.

7.6 Global Surrogate

A global surrogate model is an interpretable model that is trained to approximate the predictions of a black box model. We can draw conclusions about the black box model by interpreting the surrogate model. Solving machine learning interpretability by using more machine learning!

7.6.1 Advantages

The surrogate model method is flexible: Any model from the interpretable models chapter can be used. This also means that you can exchange not only the interpretable model, but also the underlying black box model. Suppose you create some complex model and explain it to different teams in your company. One team is familiar with linear models, the other team can understand decision trees. You can train two surrogate models (linear model and decision tree) for the original black box model and offer two kinds of explanations. If you find a better performing black box model, you do not have to change your method of interpretation, because you can use the same class of surrogate models.

I would argue that the approach is very intuitive and straightforward. This means it is easy to implement, but also easy to explain to people not familiar with data science or machine learning.

With the R-squared measure, we can easily measure how good our surrogate models are in approximating the black box predictions.

7.6.2 Disadvantages

You have to be aware that you draw conclusions about the model and not about the data, since the surrogate model never sees the real outcome.

It is not clear what the best cut-off for R-squared is in order to be confident that the surrogate model is close enough to the black box model. 80% of variance explained? 50%? 99%?

We can measure how close the surrogate model is to the black box model. Let us assume we are not very close, but close enough. It could happen that the interpretable model is very close for one subset of the dataset, but widely divergent for another subset. In this case the interpretation for the simple model would not be equally good for all data points.

The interpretable model you choose as a surrogate comes with all its advantages and disadvantages.

Some people argue that there are, in general, no intrinsically interpretable models (including even linear models and decision trees) and that it would even be dangerous to have an illusion of interpretability. If you share this opinion, then of course this method is not for you

7.7 Local Surrogate (LIME)

Local surrogate models are interpretable models that are used to explain individual predictions of black box machine learning models. Local interpretable model-agnostic explanations (LIME) is a paper in which the authors propose a concrete implementation of local surrogate models. Surrogate models are trained to approximate the predictions of the underlying black box model. Instead of training a global surrogate model, LIME focuses on training local surrogate models to explain individual predictions.

The idea is quite intuitive. First, forget about the training data and imagine you only have the black box model where you can input data points and get the predictions of the model. You can probe the box as often as you want. Your goal is to understand why the machine learning model made a certain prediction. LIME tests what happens to the predictions when you give variations of your data into the machine learning model. LIME generates a new dataset consisting of permuted samples and the corresponding predictions of the black box model. On this new dataset LIME then trains an interpretable model, which is weighted by the proximity of the sampled instances to the instance of interest.

The learned model should be a good approximation of the machine learning model predictions locally, but it does not have to be a good global approximation. This kind of accuracy is also called local fidelity.

Tabular data is data that comes in tables, with each row representing an instance and each column a feature. LIME samples are not taken around the instance of interest, but from the training data's mass center, which is problematic. But it increases the probability that the result for some of the sample points predictions differ from the data point of interest and that LIME can learn at least some explanation.

LIME for text differs from LIME for tabular data. Variations of the data are generated differently: Starting from the original text, new texts are created by randomly removing words from the original text. The dataset is represented with binary features for each word. A feature is 1 if the corresponding word is included and 0 if it has been removed.

LIME for images works differently than LIME for tabular data and text. Intuitively, it would not make much sense to perturb individual pixels, since many more than one pixel contribute to one class. Randomly changing individual pixels would probably not change the predictions by much. Therefore, variations of the images are created by segmenting the image into "superpixels" and turning superpixels off or on. Superpixels are interconnected pixels with similar colors and can be turned off by replacing each pixel with a user-defined color such as gray. The user can also specify a probability for turning off a superpixel in each permutation.

7.7.1 Advantages

Even if you replace the underlying machine learning model, you can still use the same local, interpretable model for explanation. Suppose the people looking at the explanations understand decision trees best. Because you use local surrogate models, you use decision trees as explanations without actually having to use a decision tree to make the predictions. For example, you can use a SVM. And if it turns out that an xgboost model works better, you can replace the SVM and still use as decision tree to explain the predictions.

Local surrogate models benefit from the literature and experience of training and interpreting interpretable models.

When using Lasso or short trees, the resulting explanations are short (= selective) and possibly contrastive. Therefore, they make human-friendly explanations. This is why LIME is seen more in applications where the recipient of the explanation is a lay person or someone with very little time. It is not sufficient for complete attributions, so we do not see LIME in compliance scenarios where you might be legally required to fully explain a prediction. Also for debugging machine learning models, it is useful to have all the reasons instead of a few.

LIME is one of the few methods that works for tabular data, text and images.

The fidelity measure (how well the interpretable model approximates the black box predictions) gives us a good idea of how reliable the interpretable model is in explaining the black box predictions in the neighborhood of the data instance of interest.

LIME is implemented in Python (lime and Skater) and R (lime package and iml package) and is very easy to use.

The explanations created with local surrogate models can use other (interpretable) features than the original model was trained on.. Of course, these interpretable features must be derived from the data instances. A text classifier can rely on abstract word embeddings as features, but the explanation can be based on the presence or absence of words in a sentence. A regression model can rely on a non-interpretable transformation of some attributes, but the explanations can be created with the original attributes. For example, the regression model could be trained on components of a principal component analysis (PCA) of answers to a survey, but LIME might be trained on the original survey questions. Using interpretable features for LIME can be a big advantage over other methods, especially when the model was trained with non-interpretable features.

7.7.2 Disadvantages

The correct definition of the neighborhood is a very big, unsolved problem when using LIME with tabular data. In my opinion it is the biggest problem with LIME and the reason why I would recommend to use LIME only with great care. For each application you have to try different kernel settings and see for yourself if the explanations make sense. Unfortunately, this is the best advice I can give to find good kernel widths.

Sampling could be improved in the current implementation of LIME. Data points are sampled from a Gaussian distribution, ignoring the correlation between features. This can lead to unlikely data points which can then be used to learn local explanation models.

The complexity of the explanation model has to be defined in advance. This is just a small complaint, because in the end the user always has to define the compromise between fidelity and sparsity.

Another really big problem is the instability of the explanations.Instability means that it is difficult to trust the explanations, and you should be very critical.

Conclusion: Local surrogate models, with LIME as a concrete implementation, are very promising. But the method is still in development phase and many problems need to be solved before it can be safely applied.

7.8 Shapley Values

A prediction can be explained by assuming that each feature value of the instance is a "player" in a game where the prediction is the payout. Shapley values – a method from coalitional game theory – tells us how to fairly distribute the "payout" among the features.

The Shapley value, coined by Shapley (1953)41, is a method for assigning payouts to players depending on their contribution to the total payout. Players cooperate in a coalition and receive a certain profit from this cooperation.

Players? Game? Payout? What is the connection to machine learning predictions and interpretability? The "game" is the prediction task for a single instance of the dataset. 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 receive the gain (= predict a certain value).

The Shapley value is the average marginal contribution of a feature value across all possible coalitions.

7.8.1 Advantages

The difference between the prediction and the average prediction is fairly distributed among the feature values of the instance – the Efficiency property of Shapley values. This property distinguishes the Shapley value from other methods such as LIME. LIME does not guarantee that the prediction is fairly distributed among the features. The Shapley value might be the only method to deliver a full explanation. In situations where the law requires explainability – like EU's "right to explanations" – the Shapley value might be the only legally compliant method, because it is based on a solid theory and distributes the effects fairly. I am not a lawyer, so this reflects only my intuition about the requirements.

The Shapley value allows contrastive explanations. Instead of comparing a prediction to the average prediction of the entire dataset, you could compare it to a subset or even to a single data point. This contrastiveness is also something that local models like LIME do not have.

The Shapley value is the only explanation method with a solid theory. The axioms – efficiency, symmetry, dummy, additivity – give the explanation a reasonable foundation. Methods like LIME assume linear behavior of the machine learning model locally, but there is no theory as to why this should work.

It is mind-blowing to explain a prediction as a game played by the feature values.

7.8.2 Disadvantages

The Shapley value requires a lot of computing time. In 99.9% of real-world problems, only the approximate solution is feasible. An exact computation of the Shapley value is computationally expensive because there are 2k possible coalitions of the feature values and the "absence" of a feature has to be simulated by drawing random instances, which increases the variance for the estimate of the Shapley values estimation. The exponential number of the coalitions is dealt with by sampling coalitions and limiting the number of iterations M. Decreasing M reduces computation time, but increases the variance of the Shapley value. There is no good rule of thumb for the number of iterations M. M should be large enough to accurately estimate the Shapley values, but small enough to complete the computation in a reasonable time. It should be possible to choose M based on Chernoff bounds, but I have not seen any paper on doing this for Shapley values for machine learning predictions.

The Shapley value can be misinterpreted. The Shapley value of a feature value is not the difference of the predicted value after removing the feature from the model training. The interpretation of the Shapley value is: Given the current set of feature values, the contribution of a feature value to the difference between the actual prediction and the mean prediction is the estimated Shapley value.

The Shapley value is the wrong explanation method if you seek sparse explanations (explanations that contain few features). Explanations created with the Shapley value method always use all the features. Humans prefer selective explanations, such as those produced by LIME. LIME might be the better choice for explanations lay-persons have to deal with. Another solution is SHAP introduced by Lundberg and Lee (2016), which is based on the Shapley value, but can also provide explanations with few features.

The Shapley value returns a simple value per feature, but no prediction model like LIME. This means it cannot be used to make statements about changes in prediction for changes in the input, such as: "If I were to earn 300 more a year, my credit score would increase by 5 points."

Another disadvantage is that you need access to the data if you want to calculate the Shapley value for a new data instance. It is not sufficient to access the prediction function because you need the data to replace parts of the instance of interest with values from randomly drawn instances of the data. This can only be avoided if you can create data instances that look like real data instances but are not actual instances from the training data.

Like many other permutation-based interpretation methods, the Shapley value method suffers from inclusion of unrealistic data instances when features are correlated. To simulate that a feature value is missing from a coalition, we marginalize the feature. This is achieved by sampling values from the feature's marginal distribution. This is fine as long as the features are independent.

When features are dependent, then we might sample feature values that do not make sense for this instance. But we would use those to compute the feature's Shapley value.

7.9 SHAP (SHapley Additive exPlanations)

SHAP (SHapley Additive exPlanations) by Lundberg and Lee (2016) is a method to explain individual predictions. SHAP is based on the game theoretically optimal Shapley Values.

There are two reasons why SHAP got its own chapter and is not a subchapter of Shapley values. First, the SHAP authors proposed KernelSHAP, an alternative, kernel-based estimation approach for Shapley values inspired by local surrogate models. And they proposed TreeSHAP, an efficient estimation approach for tree-based models. Second, SHAP comes with many global interpretation methods based on aggregations of Shapley values. This chapter explains both the new estimation approaches and the global interpretation methods.

The goal of SHAP is to explain the prediction of an instance x by computing the contribution of each feature to the prediction. The SHAP explanation method computes Shapley values from coalitional game theory. The feature values of a data instance act as players in a coalition. Shapley values tell us how to fairly distribute the "payout" (= the prediction) among the features. A player can be an individual feature value, e.g. for tabular data. A player can also be a group of feature values. For example to explain an image, pixels can be grouped to super pixels and the prediction distributed among them. One innovation that SHAP brings to the table is that the Shapley value explanation is represented as an additive feature attribution method, a linear model. That view connects LIME and Shapley Values.

Since SHAP computes Shapley values, the interpretation is the same as in the Shapley value section. But with the Python shap package comes a different visualization: You can visualize feature attributions such as Shapley values as "forces". Each feature value is a force that either increases or decreases the prediction. The prediction starts from the baseline. The baseline for Shapley values is the average of all predictions. In the plot, each Shapley value is an arrow that pushes to increase (positive value) or decrease (negative value) the prediction. These forces balance each other out at the actual prediction of the data instance.

SHAP Feature Importance.The idea behind SHAP feature importance is simple: Features with large absolute Shapley values are important. Since we want the global importance, we average the absolute Shapley values per feature across the data.

SHAP Summary Plot. The 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 position on the y-axis is determined by the feature and on the x-axis by the Shapley value. The color represents the value of the feature from low to high. Overlapping points are jittered in y-axis direction, so we get a sense of the distribution of the Shapley values per feature. The features are ordered according to their importance.

SHAP Dependence Plot SHAP feature dependence might be the simplest global interpretation plot: 1) Pick a feature. 2) For each data instance, plot a point with the feature value on the x-axis and the corresponding Shapley value on the y-axis. 3) Done.

Clustering SHAP values You can cluster your data with the help of Shapley values. The goal of clustering is to find groups of similar instances. Normally, clustering is based on features. Features are often on different scales. For example, height might be measured in meters, color intensity from 0 to 100 and some sensor output between -1 and 1. The difficulty is to compute distances between instances with such different, non-comparable features.

SHAP clustering works by clustering on Shapley values of each instance. This means that you cluster instances by explanation similarity. All SHAP values have the same unit – the unit of the prediction space. You can use any clustering method. The following example uses hierarchical agglomerative clustering to order the instances.

7.9.1 Advantages

Since SHAP computes Shapley values, all the advantages of Shapley values apply: SHAP has a solid theoretical foundation in game theory. The prediction is fairly distributed among the feature values. We get contrastive explanations that compare the prediction with the average prediction.

SHAP connects LIME and Shapley values. This is very useful to better understand both methods. It also helps to unify the field of interpretable machine learning.

SHAP has a fast implementation for tree-based models. I believe this was key to the popularity of SHAP, because the biggest barrier for adoption of Shapley values is the slow computation.

The fast computation makes it possible to compute the many Shapley values needed for the global model interpretations. The global interpretation methods include feature importance, feature dependence, interactions, clustering and summary plots. With SHAP, global interpretations are consistent with the local explanations, since the Shapley values are the "atomic unit" of the global interpretations. If you use LIME for local explanations and partial dependence plots plus permutation feature importance for global explanations, you lack a common foundation.

7.9.2 Disadvantages

The disadvantages of Shapley values also apply to SHAP: Shapley values can be misinterpreted and access to data is needed to compute them for new data (except for TreeSHAP).

8 Example-Based Explanations

Example-based explanation methods select particular instances of the dataset to explain the behavior of machine learning models or to explain the underlying data distribution.

Example-based explanations are mostly model-agnostic, because they make any machine learning model more interpretable. The difference to model-agnostic methods is that the example-based methods explain a model by selecting instances of the dataset and not by creating summaries of features (such as feature importance or partial dependence). Example-based explanations only make sense if we can represent an instance of the data in a humanly understandable way. This works well for images, because we can view them directly.

In general, example-based methods work well if the feature values of an instance carry more context, meaning the data has a structure, like images or texts do. It is more challenging to represent tabular data in a meaningful way, because an instance can consist of hundreds or thousands of (less structured) features. Listing all feature values to describe an instance is usually not useful. It works well if there are only a handful of features or if we have a way to summarize an instance.

8.1 Counterfactual Explanations

A counterfactual explanation describes a causal situation in the form: "If X had not occurred, Y would not have occurred". Thinking in counterfactuals requires imagining a hypothetical reality that contradicts the observed facts, hence the name "counterfactual". The ability to think in counterfactuals makes us humans so smart compared to other animals.

In interpretable machine learning, counterfactual explanations can be used to explain predictions of individual instances. The "event" is the predicted outcome of an instance, the "causes" are the particular feature values of this instance that were input to the model and "caused" a certain prediction. Displayed as a graph, the relationship between the inputs and the prediction is very simple: The feature values cause the prediction.

Even if in reality the relationship between the inputs and the outcome to be predicted might not be causal, we can see the inputs of a model as the cause of the prediction.

Given this simple graph, it is easy to see how we can simulate counterfactuals for predictions of machine learning models: We simply change the feature values of an instance before making the predictions and we analyze how the prediction changes. We are interested in scenarios in which the prediction changes in a relevant way, like a flip in predicted class (e.g. credit application accepted or rejected) or in which the prediction reaches a certain threshold (e.g. the probability for cancer reaches 10%). A counterfactual explanation of a prediction describes the smallest change to the feature values that changes the prediction to a predefined output.

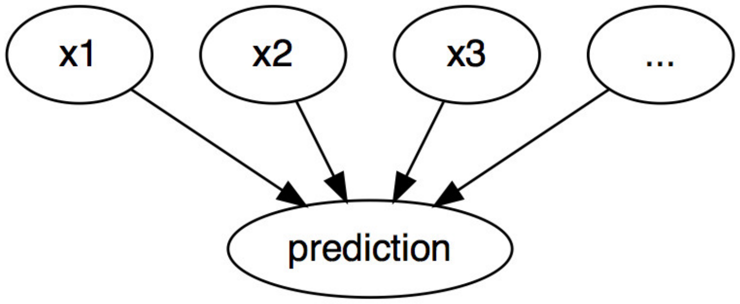

Figure 3: The causal relationships between inputs of a machine learning model and the predictions, when the model is merely seen as a black box. The inputs cause the prediction (not necessarily reflecting the real causal relation of the data).

The counterfactual explanation method is model-agnostic, since it only works with the model inputs and output. But a counterfactual explanation is itself a new instance, so it lives in this chapter ("starting from instance X, change A and B to get a counterfactual instance"). Unlike prototypes, counterfactuals do not have to be actual instances from the training data, but can be a new combination of feature values.

A simple and naive approach to generating counterfactual explanations is searching by trial and error. This approach involves randomly changing feature values of the instance of interest and stopping when the desired output is predicted. In this section, we will present the approach suggested by Wachter et. al (2017). They suggest minimizing the following loss.

$$L(x, x', y', \lambda) = \lambda \cdot (\hat{f}(x') - y')^2 + d(x, x')$$

The first term is the quadratic distance between the model prediction for the counterfactual x' and the desired outcome y', which the user must define in advance. The second term is the distance d between the instance x to be explained and the counterfactual x', but more about this later. The parameter λ balances the distance in prediction (first term) against the distance in feature values (second term). The loss is solved for a given λ and returns a counterfactual x'. A higher value of λ means that we prefer counterfactuals that come close to the desired outcome y', a lower value means that we prefer counterfactuals x' that are very similar to x in the feature values. If λ is very large, the instance with the prediction that comes closest to y' will be selected, regardless how far it is away from x. Ultimately, the user must decide how to balance the requirement that the prediction for the counterfactual matches the desired outcome with the requirement that the counterfactual is similar to x. The authors of the method suggest instead of selecting a value for λ to select a tolerance ϵ for how far away the prediction of the counterfactual instance is allowed to be from y'. This constraint can be written as:

$$|\hat{f}(x') - y'| \leq \epsilon$$

To minimize this loss function, any suitable optimization algorithm can be used, e.g. Nelder-Mead. If you have access to the gradients of the machine learning model, you can use gradient-based methods like ADAM. The instance x to be explained, the desired output y' and the tolerance parameter ϵ must be set in advance. The loss function is minimized for x' and the (locally) optimal counterfactual x' returned while increasing λ until a sufficiently close solution is found (= within the tolerance parameter).

The function d for measuring the distance between instance x and counterfactual x' is the Manhattan distance weighted feature-wise with the inverse median absolute deviation (MAD). The total distance is the sum of all p feature-wise distances, that is, the absolute differences of feature values between instance x and counterfactual x'. The feature-wise distances are scaled by the inverse of the median absolute deviation of feature j over the dataset defined as:

$$MAD_j = \text{median}_{i \in \{1,\dots,n\}}(|x_{i,j} - \text{median}_{l \in \{1,\dots,n\}}(x_{l,j})|) \tag{1}$$

The median of a vector is the value at which half of the vector values are greater and the other half smaller. The MAD is the equivalent of the variance of a feature, but instead of using the mean as the center and summing over the square distances, we use the median as the center and sum over the absolute distances. The proposed distance function has the advantage over the Euclidean distance that it introduces sparsity. This means that two points are closer to each other when fewer features are different. And it is more robust to outliers. Scaling with the MAD is necessary to bring all the features to the same scale – it should not matter whether you measure the size of an apartment in square meters or square feet.

8.1.1 Advantages

The interpretation of counterfactual explanations is very clear. If the feature values of an instance are changed according to the counterfactual, the prediction changes to the predefined prediction. There are no additional assumptions and no magic in the background. This also means it is not as dangerous as methods like LIME, where it is unclear how far we can extrapolate the local model for the interpretation.

The counterfactual method creates a new instance, but we can also summarize a counterfactual by reporting which feature values have changed. This gives us two options for reporting our results. You can either report the counterfactual instance or highlight which features have been changed between the instance of interest and the counterfactual instance.

The counterfactual method does not require access to the data or the model. It only requires access to the model's prediction function, which would also work via a web API, for example. This is attractive for companies which are audited by third parties or which are offering explanations for users without disclosing the model or data. A company has an interest in protecting model and data because of trade secrets or data protection reasons. Counterfactual explanations offer a balance between explaining model predictions and protecting the interests of the model owner.

The method works also with systems that do not use machine learning. We can create counterfactuals for any system that receives inputs and returns outputs. The system that predicts apartment rents could also consist of handwritten rules, and counterfactual explanations would still work.

The counterfactual explanation method is relatively easy to implement, since it is essentially a loss function that can be optimized with standard optimizer libraries. Some additional details must be taken into account, such as limiting feature values to meaningful ranges (e.g. only positive apartment sizes).

8.1.2 Disadvantages

For each instance you will usually find multiple counterfactual explanations (Rashomon effect). This is inconvenient – most people prefer simple explanations over the complexity of the real world. It is also a practical challenge. Let us say we generated 23 counterfactual explanations for one instance. Are we reporting them all? Only the best? What if they are all relatively "good", but very different? These questions must be answered anew for each project. It can also be advantageous to have multiple counterfactual explanations, because then humans can select the ones that correspond to their previous knowledge.

There is no guarantee that for a given tolerance ϵ a counterfactual instance is found. That is not necessarily the fault of the method, but rather depends on the data.

The proposed method does not handle categorical features with many different levels well. The authors of the method suggested running the method separately for each combination of feature values of the categorical features, but this will lead to a combinatorial explosion if you have multiple categorical features with many values. For example, 6 categorical features with 10 unique levels would mean 1 million runs. A solution for only categorical features was proposed by Martens et. al (2014)50. A solution that handles both numerical and categorical variables with a principled way of generating perturbations for categorical variables is implemented in the Python package Alibi.

8.2 Adversarial Examples

An adversarial example is an instance with small, intentional feature perturbations that cause a machine learning model to make a false prediction.Adversarial examples are counterfactual examples with the aim to deceive the model, not interpret it.

Why are we interested in adversarial examples? Are they not just curious by-products of machine learning models without practical relevance? The answer is a clear “no”. Adversarial examples make machine learning models vulnerable to attacks, as in the following scenarios.

A self-driving car crashes into another car because it ignores a stop sign. Someone had placed a picture over the sign, which looks like a stop sign with a little dirt for humans, but was designed to look like a parking prohibition sign for the sign recognition software of the car.

A spam detector fails to classify an email as spam. The spam mail has been designed to resemble a normal email, but with the intention of cheating the recipient.

A machine-learning powered scanner scans suitcases for weapons at the airport. A knife was developed to avoid detection by making the system think it is an umbrella.

There are many techniques to create adversarial examples. Most approaches suggest minimizing the distance between the adversarial example and the instance to be manipulated, while shifting the prediction to the desired (adversarial) outcome. Some methods require access to the gradients of the model, which of course only works with gradient based models such as neural networks, other methods only require access to the prediction function, which makes these methods model-agnostic. Adversarial examples for images are images with intentionally perturbed pixels with the aim to deceive the model during application time.

8.3 Prototypes and Criticisms

A prototype is a data instance that is representative of all the data. A criticism is a data instance that is not well represented by the set of prototypes. The purpose of criticisms is to provide insights together with prototypes, especially for data points which the prototypes do not represent well. Prototypes and criticisms can be used independently from a machine learning model to describe the data, but they can also be used to create an interpretable model or to make a black box model interpretable.

There are many approaches to find prototypes in the data. One of these is k-medoids, a clustering algorithm related to the k-means algorithm. Any clustering algorithm that returns actual data points as cluster centers would qualify for selecting prototypes. But most of these methods find only prototypes, but no criticisms. MMD-critic is an approach that combines prototypes and criticisms in a single framework.

MMD-critic compares the distribution of the data and the distribution of the selected prototypes. This is the central concept for understanding the MMD-critic method. MMD-critic selects prototypes that minimize the discrepancy between the two distributions. Data points in areas with high density are good prototypes, especially when points are selected from different “data clusters”. Data points from regions that are not well explained by the prototypes are selected as criticisms.

8.3.1 Advantages

In a user study the authors of MMD-critic gave images to the participants, which they had to visually match to one of two sets of images, each representing one of two classes (e.g. two dog breeds). The participants performed best when the sets showed prototypes and criticisms instead of random images of a class.

We are free to choose the number of prototypes and criticisms.

MMD-critic works with density estimates of the data. This works with any type of data and any type of machine learning model.

The algorithm is easy to implement.

MMD-critic is very flexible in the way it is used to increase interpretability. It can be used to understand complex data distributions. It can be used to build an interpretable machine learning model. Or it can shed light on the decision making of a black box machine learning model.

Finding criticisms is independent of the selection process of the prototypes. But it makes sense to select prototypes according to MMD-critic, because then both prototypes and criticisms are created using the same method of comparing prototypes and data densities.

8.3.2 Disadvantages

While, mathematically, prototypes and criticisms are defined differently, their distinction is based on a cut-off value (the number of prototypes). Suppose you choose a too low number of prototypes to cover the data distribution. The criticisms would end up in the areas that are not that well explained. But if you were to add more prototypes they would also end up in the same areas. Any interpretation has to take into account that criticisms strongly depend on the existing prototypes and the (arbitrary) cut-off value for the number of prototypes.

You have to choose the number of prototypes and criticisms. As much as this can be nice-to-have, it is also a disadvantage. How many prototypes and criticisms do we actually need? The more the better? The less the better? One solution is to select the number of prototypes and criticisms by measuring how much time humans have for the task of looking at the images, which depends on the particular application. Only when using MMD-critic to build a classifier do we have a way to optimize it directly. One solution could be a screeplot showing the number of prototypes on the x-axis and the MMD2 measure on the y-axis. We would choose the number of prototypes where the MMD2 curve flattens.

The other parameters are the choice of the kernel and the kernel scaling parameter. We have the same problem as with the number of prototypes and criticisms: How do we select a kernel and its scaling parameter? Again, when we use MMD-critic as a nearest prototype classifier, we can tune the kernel parameters. For the unsupervised use cases of MMD-critic, however, it is unclear. (Maybe I am a bit harsh here, since all unsupervised methods have this problem.)

It takes all the features as input, disregarding the fact that some features might not be relevant for predicting the outcome of interest. One solution is to use only relevant features, for example image embeddings instead of raw pixels. This works as long as we have a way to project the original instance onto a representation that contains only relevant information.

There is some code available, but it is not yet implemented as nicely packaged and documented software.

8.4 Influential Instances

Machine learning models are ultimately a product of training data and deleting one of the training instances can affect the resulting model. We call a training instance "influential" when its deletion from the training data considerably changes the parameters or predictions of the model. By identifying influential training instances, we can "debug" machine learning models and better explain their behaviors and predictions.

There are two approaches for identifying influential instances, namely deletion diagnostics and influence functions. Both approaches are based on robust statistics, which provides statistical methods that are less affected by outliers or violations of model assumptions. Robust statistics also provides methods to measure how robust estimates from data are (such as a mean estimate or the weights of a prediction model).

Outliers. An outlier is an instance that is far away from the other instances in the dataset. "Far away" means that the distance, for example the Euclidean distance, to all the other instances is very large. Outliers can be interesting data points (e.g. criticisms). When an outlier influences the model it is also an influential instance.

Influential instance. An influential instance is a data instance whose removal has a strong effect on the trained model. The more the model parameters or predictions change when the model is retrained with a particular instance removed from the training data, the more influential that instance is. Whether an instance is influential for a trained model also depends on its value for the target y.

8.4.1 Deletion Diagnostics

Statisticians have already done a lot of research in the area of influential instances, especially for (generalized) linear regression models. When you search for "influential observations", the first search results are about measures like DFBETA and Cook's distance. DFBETA measures the effect of deleting an instance on the model parameters. Cook's distance (Cook, 197763) measures the effect of deleting an instance on model predictions. For both measures we have to retrain the model repeatedly, omitting individual instances each time. The parameters or predictions of the model with all instances is compared with the parameters or predictions of the model with one of the instances deleted from the training data.

8.4.2 Influence Functions

As with deletion diagnostics, the influence functions trace the model parameters and predictions back to the responsible training instance. However, instead of deleting training instances, the method approximates how much the model changes when the instance is upweighted in the empirical risk (sum of the loss over the training data).

The method of influence functions requires access to the loss gradient with respect to the model parameters, which only works for a subset of machine learning models. Logistic regression, neural networks and support vector machines qualify, tree-based methods like random forests do not. Influence functions help to understand the model behavior, debug the model and detect errors in the dataset.

8.4.3 Advantages of Identifying Influential Instances

A look at influential instances emphasizes the role of training data in the learning process. This makes influence functions and deletion diagnostics one of the best debugging tools for machine learning models.

Deletion diagnostics are model-agnostic, meaning the approach can be applied to any model. Also influence functions based on the derivatives can be applied to a broad class of models.

We can use these methods to compare different machine learning models and better understand their different behaviors, going beyond comparing only the predictive performance.

Influence functions via derivatives can also be used to create adversarial training data. These are instances that are manipulated in such a way that the model cannot predict certain test instances correctly when the model is trained on those manipulated instances.

For deletion diagnostics and influence functions, we considered the difference in the prediction and for the influence function the increase of the loss. But, really, the approach is generalizable to any question of the form: "What happens to ... when we delete or upweight instance z?", where you can fill "..." with any function of your model of your desire. You can analyze how much a training instance influences the overall loss of the model. You can analyze how much a training instance influences the feature importance. You can analyze how much a training instance influences which feature is selected for the first split when training a decision tree.

8.4.4 Disadvantages of Identifying Influential Instances

Deletion diagnostics are very expensive to calculate because they require retraining. But history has shown that computer resources are constantly increasing. It is therefore not a big leap to assume that deletion diagnostics will work without problems even with large neural networks in 10 years.

Influence functions are a good alternative to deletion diagnostics, but only for models with differentiable parameters, such as neural networks. They do not work for tree-based methods like random forests, boosted trees or decision trees. Even if you have models with parameters and a loss function, the loss may not be differentiable. But for the last problem, there is a trick: Use a differentiable loss as substitute for calculating the influence when, for example, the underlying model uses the Hinge loss instead of some differentiable loss. The loss is replaced by a smoothed version of the problematic loss for the influence functions, but the model can still be trained with the non-smooth loss.

Influence functions are only approximate, because the approach forms a quadratic expansion around the parameters. The approximation can be wrong and the influence of an instance is actually higher or lower when removed.

There is no clear cutoff of the influence measure at which we call an instance influential or non-influential. It is useful to sort the instances by influence, but it would be great to have the means not only to sort the instances, but actually to distinguish between influential and non-influential.

The influence measures only take into account the deletion of individual instances and not the deletion of several instances at once. Larger groups of data instances may have some interactions that strongly influence model training and prediction. But the problem lies in combinatorics: There are n possibilities to delete an individual instance from the data. There are n times (n-1) possibilities to delete two instances from the training data. There are n times (n-1) times (n-2) possibilities to delete three ... we can see where this is going, there are just too many combinations.

9 Conclusion

The interpretability is an important part of anyone that uses Machine Learning, that needs to be analysed in order to be able to trust the results of the produced models. There is an insane amount of methods and techniques that achieve that, with each one of them having its own advantages and disadvantages. Selecting the most suitable ones for each application is not an easy task, but most of this trouble is now gone.

References

[1] Christoph Molnar. *Interpretable Machine Learning.* 2019. https://christophm.github.io/interpretable-ml-book/.