Interpretability is crucial for trusting AI and machine learning

We explain what exactly interpretability is and why it is so important, focusing on its use for data scientists, end users and regulators.

By Ilknur Kaynar Kabul, SAS. Source; KDNuggets

As machine learning takes its place in many recent advances in science and technology, the interpretability of machine learning models grows in importance. We are surrounded with applications powered by machine learning, and we're personally affected by the decisions made by machines more and more every day. From the mundane to the lifesaving, we ask machine learning models for answers to questions like:

- What song will I enjoy?
- Will I be able to get a loan?
- Who should I hire?
- What is the probability of me having cancer?

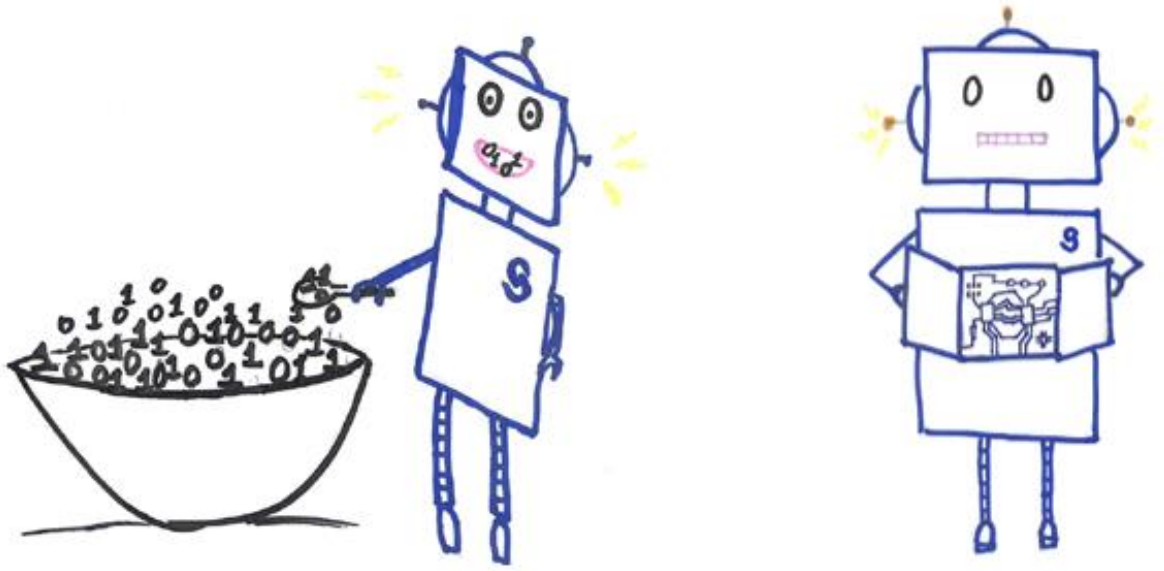
These and many other questions are answered by predictive models that most users know very little about. Data scientists often put emphasis on the prediction accuracy of their models – not on understanding *how* those predictions are actually made. With machine learning, the models just do it.

Complex models are harder to understand

Some machine learning models are simple and easy to understand. We know how changing the inputs will affect the predicted outcome and can make justification for each prediction. However, with the recent advances in machine learning and artificial intelligence, models have become very complex, including complex deep neural

networks and ensembles of different models. We refer to these complex models as black box models.

Unfortunately, the complexity that gives extraordinary predictive abilities to black box models also makes them very difficult to understand and trust. The algorithms inside the black box models do not expose their secrets. They don't, in general, provide a clear explanation of why they made a certain prediction. They just give us a probability, and they are opaque and hard to interpret. Sometimes there are thousands (even millions) of model parameters, there’s no one-to-one relationship between input features and parameters, and often combinations of multiple models using many parameters affect the prediction. Some of them are also data-hungry. They need enormous amounts of data to achieve high accuracy. It’s hard to figure out what they learned from those data sets and which of those data points have more influence on the outcome than the others.



Due to all those reasons, it’s very difficult to understand the process and the outcomes from those techniques. It’s also difficult to figure out whether we can trust the models and whether we can make fair decisions when using them.

What happens if they learn the wrong thing? What happens if they are not ready for deployment? There is a risk of misrepresentation, oversimplification or overfitting. Thus, we need to be careful when we use them, and we'd better understand how those models work.

Why accuracy is not enough

In machine learning, accuracy is measured by comparing the output of a machine learning model to the known actual values from the input data set.

A model can achieve high accuracy by memorizing the unimportant features or patterns in your data set. If there is a bias in your input data set, this can also affect your model. In addition, the data in the training environment may not be a good representation of the data in the production environment in which the model is deployed. Even if it is sufficiently representative initially, if we consider that the data in the production environment is not stationary, it can become outdated very quickly.

Thus, we cannot rely only on the prediction accuracy achieved for a specific data set. We need to know more. We need to demystify the black box machine learning models and improve transparency and interpretability to make them more trustworthy and reliable.

What is interpretability?

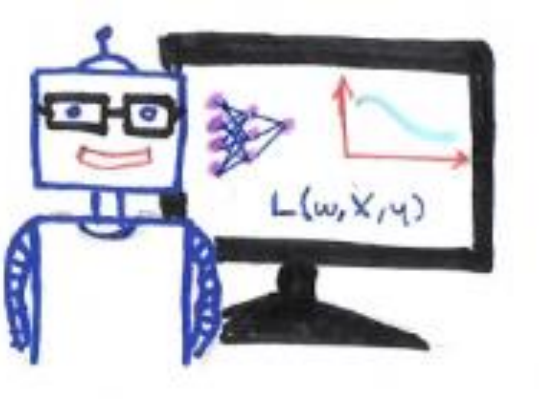
Interpretability means giving explanations to the end users for a particular decision or process. More specifically, it entails:

- Understanding the main tasks that affect the outcomes.
- Explaining the decisions that are made by an algorithm.
- Finding out the patterns/rules/features that are learned by an algorithm.
- Being critical about the results.
- Exploring the unknown unknowns for your algorithm.

It is not about understanding every detail about how a model works for each data point in the training data.

Why do we need interpretability?

Interpretability is important to different people for different reasons:



Data scientists want to build models with high accuracy. They want to understand the details to find out how they can pick the best model and improve that model. They also want to get insights from the model so that they can communicate their findings to their target audience.

End users want to know why a model gives a certain prediction. They want to know how they will be affected by those decisions. They want to know whether they are being treated fairly and whether they need to object to any decision. They want to have a certain measure of trust when they are shopping online, or clicking ads on the web.

Regulators and lawmakers want to make the system fair and transparent. They want to protect consumers. With the inevitable rise of machine learning algorithms, they are becoming more concerned about the decisions made by models.

All those users want similar things from the black box models. They want them to be transparent, trustworthy and explainable.

1. **Transparent**: The system can explain how it works and/or why it gives certain predictions

2. **Trustworthy**: The system can handle different scenarios in the real world without continuous control.
3. **Explainable**: The system can convey useful information about its inner workings, for the patterns that it learns and for the results that it gives.

In a typical machine learning pipeline, we have control over the data set used to train the model, we have control over the model that we use, and we have control over how we assess and deploy those models.

When is interpretability needed?

If you need interpretability, first you need to ask yourself why you need it. In which stage of this process do you need interpretability? It may not be necessary to understand how a model makes its predictions for every application. However, you may need to know it if those predictions are used for high-stakes decisions. After you define your purpose, you should focus on what techniques you need in which stage of the process:

1. **Interpretability in pre-modeling (interpretability of model inputs):**Understanding your data set is very important before you start building models. You can use different exploratory data analysis and visualization techniques to have a better understanding of your data set. This can include summarizing the main characteristics of your data set, finding representative or critical points in your data set, and finding the relevant features from your data set. After you have an overall understanding of your data set, you need to think about which features you are going to use in modeling. If you want to explain the input-output relationship after you do modeling, you need to start with meaningful features. While highly engineered features (such as those obtained from t-sne, random projections,

etc.) can boost the accuracy of your model, they will not be interpretable when you put the model to use.

2. **Interpretability in modeling:**We can categorize models as white box (transparent) and black box (opaque) models based on their simplicity, transparency and explainability.
    1. **White box (transparent) models**: Decision trees, rule-lists, and regression algorithms are usually considered in this category. These models are easy to understand when used with few predictors. They use interpretable transformations and give you more intuition about how things work, which helps you understand what's going on in the model. You can explain them to a technical audience. But of course, if you have hundreds of features and you build a very deep, large decision tree, things can still become complicated and uninterpretable.
    2. **Black box (opaque) models:**Deep neural networks, random forests, and gradient boosting machines can be considered in this category. They usually use many predictors and complex transformations. Some of them have many parameters. It's usually hard to visualize and understand what is going on inside these models. They're harder to communicate to a target audience. However, their prediction accuracy can be much better than other models. Recent research in this area hopes to make these models more transparent. Some of that research includes techniques that are part of the training process. Generating explanations in addition to the predictions is one way to improve transparency in these models. Another improvement is to include visualization of features after the training process.

3. **Interpretability in post-modeling (post hoc interpretability):**Interpretability in the model predictions helps us to inspect the dynamics between input features and output predictions. Some post-modeling activities are model-specific, while the others are model-agnostic. Adding interpretability at this phase can help us to understand the most important features for a model, how those features affect the predictions, how each feature contributes to the prediction and how sensitive your model is to certain features. There are model-agnostic techniques such as Partial Dependence (PD) plots, Individual Conditional Expectation (ICE) plots and Local Interpretable Model-Agnostic Explanations (LIME), in addition to the model-specific techniques, such as variable importance output from Random Forest.

Machine Learning Explainability vs Interpretability: Two concepts that could help restore trust in AI

We explain the key differences between explainability and interpretability and why they're so important for machine learning and AI, before taking a look at several techniques and methods for improving machine learning interpretability.

By Richard Gall, Packt https://www.kdnuggets.com/2018/12/machine-learning-explainability-interpretability-ai.html

It doesn't take a data scientist to work out that the machine and deep learning algorithms built into automation and artificial intelligence systems lack transparency. It also doesn't take a great deal of detective work to see that many of these systems contain an imprint of the unconscious biases of the engineers that helped to develop them.

Arguably, in the midst of what The Economist termed a *techlash*, this lack of transparency has only (ironically) become more visible. While many of the incidents that have contributed towards the techlash are as much issues caused by a mixture of corporate self-interest and an alarming absence of governance and accountability, there's no escaping the fact that the practice of data science and machine learning engineering naturally find their way hooked onto some of the year's biggest business and politics stories.

It's in this context that the concepts of explainability and interpretability have taken on new urgency. It's likely that they are only going to become more important in 2019, as discussions around the ethics of artificial intelligence continues.

But what are explainability and interpretability? And how what do they actually mean for those of us doing data mining, analysis, science in 2019?

The difference between machine learning explainability and interpretability

In the context of machine learning and artificial intelligence, explainability and interpretability are often used interchangeably. While they are very closely related, it's worth unpicking the differences, if only to see how complicated things can get once you start digging deeper into machine learning systems.

Interpretability is about the extent to which a cause and effect can be observed within a system. Or, to put it another way, it is the extent to which you are able to *predict* what is going to happen, given a change in input or algorithmic parameters. It's being able to look at an algorithm and go *yep, I can see what's happening here.*

Explainability, meanwhile, is the extent to which the internal mechanics of a machine or deep learning system can be explained in human terms. It's easy to miss the subtle difference with interpretability, but consider it like this: interpretability is about being able to discern the mechanics without necessarily knowing why. Explainability is being able to quite literally explain what is happening.

Think of it this way: say you're doing a science experiment at school. The experiment might be interpretable insofar as you can see what you're doing, but it is only really explainable once you dig into the chemistry behind what you can see happening.

That might be a little crude, but it is nevertheless a good starting point for thinking about how the two concepts relate to one another.

Why are explainability and interpretability important in artificial intelligence and machine learning?

If 2018's techlash has taught us anything, it's that although technology can certainly be put to dubious usage, there are plenty of ways in which it can produce poor - discriminatory - outcomes with no intention of causing harm.

As domains like healthcare look to deploy artificial intelligence and deep learning systems, where questions of accountability and transparency are particularly important, if we're unable to properly deliver improved interpretability, and ultimately explainability, in our algorithms, we'll seriously be limiting the potential impact of artificial intelligence. Which would be a shame.

But aside from the legal and professional considerations that need to be made, there's also an argument that improving interpretability and explainability are important even in more prosaic business scenarios. Understanding how an algorithm is actually working can help to better align the activities of data scientists and analysts with the key questions and needs of their organization.

Techniques and methods for improving machine learning interpretability

While questions of transparency and ethics may feel abstract for the data scientist on the ground, there are, in fact, a number of practical things that can be done to improve an algorithm's interpretability and explainability.

Algorithmic generalization

The first is to improve generalization. This sounds simple, but it isn't that easy. When you think most machine learning engineering is applying algorithms in a very specific way to uncover a certain desired outcome, the model itself can feel like a secondary element - it's simply a means to an end. However, by shifting this attitude to consider the overall health of the algorithm, and the data on which it is running, you can begin to set a solid foundation for improved interpretability.

Pay attention to feature importance

This should be obvious, but it's easily missed. Looking closely at the way the various features of your algorithm have been set is a practical way to actually engage with a diverse range of questions, from business alignment to ethics. Debate and discussion over how each feature should be set might be a little time-consuming, but having that tacit awareness that different features have been set in a certain way is nevertheless an important step in moving towards interpretability and explainability.

LIME: Local Interpretable Model-Agnostic Explanations

While the techniques above offer practical steps that data scientists can take, LIME is an actual method developed by researchers to gain greater transparency on what's happening inside an algorithm. The researchers explain that LIME can explain "the predictions of any classifier in an interpretable and faithful manner, by learning an interpretable model locally around the prediction."

What this means in practice is that the LIME model develops an approximation of the model by testing it out to see what happens when certain aspects within the model are changed. Essentially it's about trying to recreate the output from the same input through a process of experimentation.

DeepLIFT (Deep Learning Important Features)

DeepLIFT is a useful model in the particularly tricky area of deep learning. It works through a form of backpropagation: it takes the output, then attempts to pull it apart by 'reading' the various neurons that have gone into developing that original output.

Essentially, it's a way of digging back into the feature selection inside of the algorithm (as the name indicates).

Layer-wise relevance propagation

Layer-wise relevance propagation is similar to DeepLIFT, in that it works backwards from the output, identifying the most relevant neurons within the neural network until you return to the input (say, for example, an image). If you want to learn more about the mathematics behind the concept, this post by Dan Shiebler is a great place to begin.

Adding complexity to tackle complexity: can it improve transparency?

The central problem with both explainability and interpretability is that you're adding an additional step in the development process. Indeed, you're probably adding multiple steps. From one perspective, this looks like you're trying to tackle complexity with even greater complexity.

And to a certain extent, this is true. What this means in practice is that if we're to get really serious about interpretability and explainability, there needs to be a broader cultural change in the way in which data science and engineering is done, and how people *believe* it should be done. That, perhaps, is the really challenging part.

Model Explainability — How to choose the right tool?

Mateusz Garbacz https://medium.com/ing-blog/model-explainability-how-to-choose-the-right-tool-6c5eabd1a46a

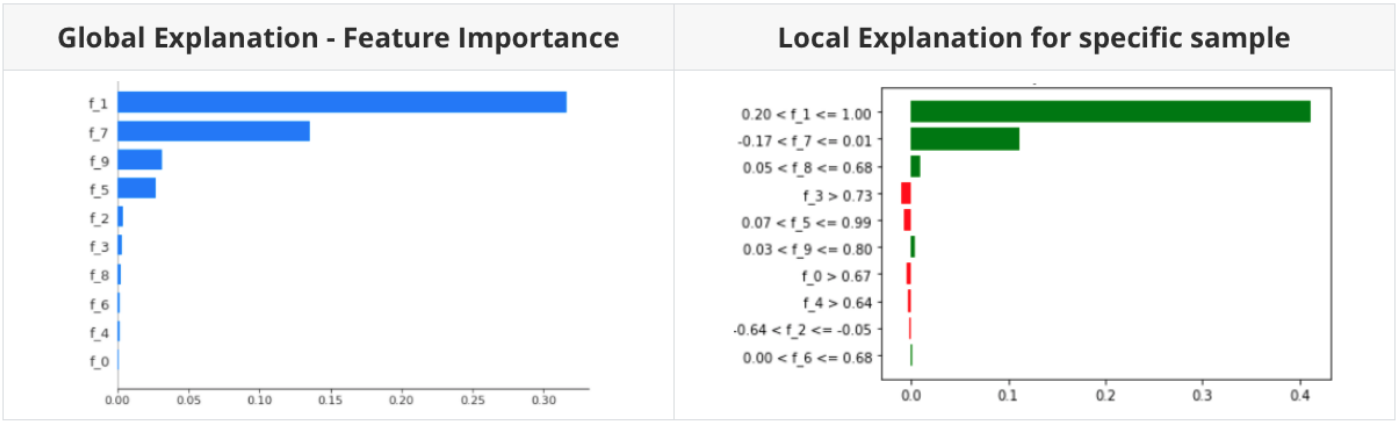
At ING we put a lot of importance on making sure that the Machine Learning (ML) models we build, are well tested and safe to use. A crucial part of it is explaining and understanding the model. However, there are many explainability tools available, and each of them uses a different methodology. One technique might lead to significantly different results than the other. **Which one should you use in your project?**

In this post, we present how we have answered that question for our team in ING, in which we mainly use tree-based models. We discuss, which aspects of explainability tools were crucial while making the decision, and compare different approaches.

Global vs. local explainability

Model Explainability is a broad concept of analyzing and understanding the results provided by ML models. It is most often used in the context of "black-box" models, for which it is difficult to demonstrate, how did the model arrive at a specific decision.

Many different tools allow us to analyze the black-box models, and interestingly, each of them looks from a slightly different angle. Some of these techniques focus on a global explanation — a holistic view of the main factors that influence the predictions of this model. An example of a global explanation is an overview of the Feature Importance (FI) of the model. Other tools focus on generating a local explanation, which means focusing on a specific prediction made by the model.



Examples of global and local explanations.

What is essential to understand is each model explainability tool is different. Using two distinct tools for computing global explanations lead to two diverse outcomes, and thus, a different understanding of the model. More importantly, using certain explanation techniques in some cases might lead to strong bias or wrong conclusions.

Having said that, we have decided to conduct research, on which method best suits our needs.

Criteria for choosing the tool

One of the most crucial aspects of choosing the right tool for model explainability is the context, in which you operate. Think about these questions:

- Do you work with tabular, text, or image data?
- Do you want to get a global or local understanding of the model?
- Which aspects of the data, model, the business problem need to be considered?

Addressing these questions allows you to arrive at an initial set of tools to consider, as well as the aspects that might be relevant while selecting the one that suits your needs.

I work at the Risk and Pricing Advanced Analytics team at ING, and the majority of the models that we build are binary classification models. A flagship example is a binary classification model, which aims at detecting defaults, meaning if the customer will or will not manage to pay back the loan.

There are many common characteristics of these problems:

- Business and regulators impose the need for thoroughly analysing the model, both globally and locally. In case the customer challenges the automatic loan rejection in court, we need to explain model's decision,
- The explainability tool needs to be safe to use, it should not be misleading under almost any circumstances,
- Large tabular datasets used to develop the model, sometimes with millions of samples and thousands of features,
- Varying features' properties: categorical and numeric features, often having missing or correlated values
- Global explainability tools are for other purposes as well e.g. feature selection,
- Often best performing models are tree-based approaches.

Taking these into consideration, we need to find a **flexible**, **fast**, and **trustworthy** solution, which works well for **tabular data** and **tree-based** models. Let's find the candidate model explainability techniques for our target solution.

Analyzed Methods

Firstly, we will discuss the most often used model explainability tools that work with tabular data and the tree-based models.

Impurity-based Feature Importance

It is one of the most often used methods to get a global understanding of the tree-based model. You might not recognise the name, but it is applied, for example, to compute the `feature_importances_` attribute in all the tree-based models in `scikit-learn`.

Permutation Feature Importance

Another popular technique applied to get an overall understanding of the model is Permutation FI. It is computed, by randomly permuting values of a given feature, and calculating the loss of the model's performance caused by this distortion.

As an example, let's assume a model with ROC AUC equal to 90% on the validation set. If you shuffle the `requested_loan_amount` feature, getting the validation score down to 80%, the importance of this feature is 10%.

To put it simply, the technique presents how much the performance of the model relies on a given feature and its quality.

Leave One Feature Out (LOFO)

This method calculates the feature importance for any model and dataset, by iteratively removing each feature from the set, and computing the model performance.

Coming back to the example of the model with validation ROC AUC of 90%. If you remove the `requested_loan_amount` feature and retrain the model without it, getting the score down to 80%, the importance of this feature is 10%.

In simple words, it represents how much performance would be lost if a given feature was not available.

The `lofo-importance` package provides a python implementation of the technique.

SHapley Additive exPlanations (SHAP)

SHAP is a package that, as described in their documentation,

"is a game theoretic approach to explain the output of any machine learning model. It connects optimal credit allocation with local explanations using the classic Shapley values from game theory and their related extensions…."

In a nutshell, SHAP computes Shapley values, which represent the contribution of a given feature towards the prediction of the model for a given sample. The explanation presents how strongly, and which way a given feature affects prediction both locally or globally.

Local Interpretable Model-agnostic Explanations (LIME)

LIME is a package, which focuses on explaining the prediction of a model locally. The explanation is computed, by first, generating random points around the explained sample, computing the model's output for these points, and training a surrogate model on top of this output.

The surrogate models are simple and explainable (e.g. linear) ML models that are trained to approximate the predictions of the underlying black-box model. By analyzing the surrogate model, you can get insights into the explained model.

For example, let's assume you want to explain the prediction of a complex tree-based using a Logistic Regression surrogate model. If the `Beta` coefficient for the feature `requested_loan_amount` is above 0, the higher values of that feature would result in higher confidence of the positive class.

Lime is especially effective for text and image-based models, however, is also applicable for tabular data.

Analysis

In this section, we will compare the selected model explainability tools, in terms of the most relevant criteria. The table below presents an overview of the results of our analysis. The colors indicate the strengths (green) and weaknesses (red) of the compared tools. We will go over the different comparison dimensions and discuss each of them separately.

						Issue Types method prone to		
Name	Explanation type	Data used	Model types supported	Explanation potential	Speed	Correlated features	High cardinality features	Based on unrealistic data points
Impurity FI	Global	Trained model	Only Trees	Single feature				
Permutation FI	Global	Hold-out / CV	Any	Single feature				
LOFO	Global	Hold-out / CV	Any	Single feature				
SHAP	Global and local	Any datapoint	Different explainers	Single feature or interactions	Tree Explainer only			
LIME	Local	Any datapoint	Any	Single feature or interactions				

Comparison of selected model explainability tools.

Criteria 1: Explanation type

One of the main aspects to consider, when selecting the model explainability tool is, whether it allows you to get local or global explanations. Out of the compared tools, SHAP has the widest range of applications, because it allows computing various plots on the global, as well as the local level.

Criteria 2: Data used

Another crucial aspect is the data used to compute the explanation. In general, it is preferable to compute model explanation on the data held out from the model training. Thanks to this, one can assess how the model acts on the unseen samples. Using only training data may have a severe and misleading effect on the explanation, especially if the model overfits it.

From the compared methods, only the Impurity FI is computed based on the model structure, formed based on the train data. Other tools make use of the data given by the user, or Cross-Validation (CV). SHAP and LIME provide the highest flexibility since the explanation can be computed on any data point.

Criteria 3: Model types supported

In many cases, it is crucial to ensure that the model explainability tool works for any model. In case the tree-based classifiers are outperformed by others, the target solution should be able to explain it as well.

While Impurity FI can only be calculated for tree-based models, the Permutation FI, LOFO, and LIME are model-agnostic tools, which means that they work for any classifier.

SHAP uses various explainers, which focus on analyzing specific types of models. For instance, the `TreeExplainer` can be used for tree-based models and

the `KernelExplainer` for any model. However, the properties of these explainers significantly differ, for instance, the `KernelExplainer` is significantly slower than `TreeExplainer`. Therefore, SHAP works especially well for tree-based models and can be used for other models, but it needs to be done with caution.

Criteria 4: Explanation potential

Explanation potential indicates how complex patterns can be explained, using a given technique.

The first 3 of the compared tools compute the explanations for each feature separately. Therefore, they provide a limited understanding of non-linear patterns that the model has learned.

When it comes to SHAP, Shapley values are computed, based on the individual contribution of a given feature, as well as the interactions between features. Different plots provided by the package, allow to get a different level of insights about the model.

Finally, the complexity of LIME explanation depends on the type of surrogate model used. For linear surrogate models, each feature is analyzed separately, while, for instance, simple tree-based surrogate models explain more complex feature interactions.

Criteria 5: Speed

Another crucial aspect to consider is, how long does it take to compute the explanation. This may affect your project, especially if you work with a relatively large dataset.

To test the speed of the tools, we have set up a simple experiment that aims at measuring the mean runtime of the analyzed tools. We have constructed two simple numerical datasets:

- 2000 samples with 10 features, split into 50/50 train/validation sets,
- 20000 samples with 100 features, split into 50/50 train/validation sets,

Later, we have fitted a Random Forest classifier (100 estimators, and max depth of 5) on the train set, and used the validation splits to compute global and local explanations. Thus, the global explanations have been computed on 1000x10 and 10000x100 sets of samples, and local explanations on 10x10 and 10x100 samples. The mean and standard deviation of the run time has been measured over multiple runs. The table below presents the runtime results:

Explainability tool	Global - 1000x10	Global - 10000x100	Local - 10x10	Local 10x100
Impurity FI	< 0.1 s	< 0.1 s	-	-
Permutation FI (50 iterations)	6.3 s ± 0.7 s	266 s ± 11.5 s	-	-
LOFO	2.1 s ± 0.2 s	280 s ± 20.9 s	-	-
SHAP	0.1 s ± 0.0 s	3.3 s ± 0.3 s	< 0.1 s	< 0.1 s
LIME	-	-	4.4 s ± 0.4 s	32.2 s ± 2.1 s

Speed comparison of the selected tools.

Before we deep dive into the results, let's keep in mind that the runtime might differ, depending on the machine used, model complexity, parallelization applied, and tool's settings (e.g. `iterations` parameter in Permutation FI).

The experiment shows that Impurity FI and SHAP have the best performance in terms of time. For SHAP it only applies to TreeExplainer, if a non-tree-based model was used, the run time would significantly increase.

Using Permutation FI and LOFO is time expensive, and may significantly slow down your project in some cases.

Finally, the run time of LIME is higher than SHAP for local explanations of tree-based models. However, it should not affect the project negatively, because such explanations are typically computed for several samples only.

Criteria 6: Correlated Features

In almost any dataset, there are pairs of highly correlated features. Such pairs of features act in a very similar way and one does not bring much new information over the other one. An excellent example of such would be `requested_loan_amount_in_usd` and `requested_loan_amount_in_euro` with Pearson's correlation of 1.

Most of the ML models can efficiently deal with correlated features during training the model. However, many of the explanation tools do not. Let's assume a situation that in the model, we only use one of the two correlated features mentioned above, and all explainability tools agree that it is the most important feature.

If the model also uses the `requested_loan_amount_in_usd`, Impurity FI, LOFO, SHAP, and LIME will distribute the importance over these two features for some models, possibly causing that none of them is in the top 5 most important features. Therefore, if you do not remove highly correlated features, these explanations might be negatively affected by them.

This issue has the most severe effect on LOFO since the FI is computed by iteratively removing features. If at one iteration it removes one of the two correlated features, the model still has the other feature to train on. In such a case,

for instance, a Decision tree has no loss of performance. Therefore, the feature importance of correlated pairs of features might be driven to 0 in LOFO.

The table below presents a simplified effect of a pair of highly correlated features on the explanations:

Explainability tool	feature_1 Importance	correlated_feature Importance
Optimal Explanation	1	0
Impurity FI	0.5	0.5
Permutation FI	0.5	0.5
LOFO	0	0
SHAP	0.5	0.5
LIME	0.5	0.5

Comparison of behaviour of selected tools in case of highly correlated features.

Overall, the correlated features may have a strong effect on the quality of the model explanation. Therefore, it is best to prevent that issue, by removing correlated features, or at least be aware of them, while explaining models.

Criteria 7: High Cardinality Features

High cardinality features are variables that have many distinct values. A categorical feature `loan_type` has only a couple of possible values, thus it has a low cardinality. In contrast, the numerical feature `requested_loan_amount` might have thousands of different ones in your dataset, thus we call it high cardinality.

When you train a tree-based model, high-cardinality features have a significantly higher chance of being selected higher in the tree, only because there are many more places for placing the feature split in the tree node. This causes that Impurity metrics calculation might be significantly affected by it.

This post presents an experiment, in which a completely random feature scores a high value of Impurity FI.

Thus, the tool is not trustworthy for numerical datasets and should be avoided. The other analyzed tools, are safe to use with high cardinality features in the dataset.

Criteria 8: Unrealistic Data Points

The final dimension that we have considered, is whether the method uses unrealistic data points to compute model explanation.

Let's assume that we a sample with two features `has_credit_card = True` and `credit_card_saldo=1000`. If you apply Permutation FI or LIME, these methods may perturb the original sample to the following state `has_credit_card = False` and `credit_card_saldo=1000`, which is contradictory. Using such a sample to make any conclusions, might lead to bias. Thus, in Permutation FI, if permuting `has_credit_card` leads to a dramatic loss of performance, this might mean that either the feature is important or the model is sensitive to unrealistic situations, caused by low data quality.

In some domains and for some datasets, the issue described above has low severity. However, when building credit risk-related models, it is better to stick to methods that use the original dataset (SHAP and LOFO) and analyze situations that may materialize.

The Verdict

Having considered the context, in which our team explains the models, and the above analysis, we have decided to use **SHAP** as the best practice in our projects. It is characterized by high flexibility, speed, and safety of use in our projects.

We use SHAP in various ways and have built tools on top of it in probatus, an open-source package developed by ING. See examples on how we use SHAP to:

- Generate all the important model explanation plots faster,
- Perform feature selection process,
- Analyse the difference between two samples of data, for instance, data shift caused by Covid-19.

An overview of model explainability in modern machine learning

Rui Aguiar https://towardsdatascience.com/an-overview-of-model-explainability-in-modern-machine-learning-fc0f22c8c29a

Model explainability is one of the most important problems in machine learning today. It's often the case that certain "black box" models such as deep neural networks are deployed to production and are running critical systems from everything in your workplace security cameras to your smartphone. It's a scary thought that not even the developers of these algorithms understand why exactly the algorithms make the decisions they do — or even worse, how to prevent an adversary from exploiting them.

While there are many challenges facing the designer of a "black box" algorithm, it's not completely hopeless. There are actually many different ways to illuminate the decisions a model makes. It's even possible to understand which features are the most salient in a model's predictions.

In this article, I give a comprehensive overview of model explainability for deeper models in machine learning. I hope to explain how deeper models more traditionally considered “black boxes” can actually be surprisingly explainable. We use model-agnostic methods to apply interpretability to all different kinds of black box models.

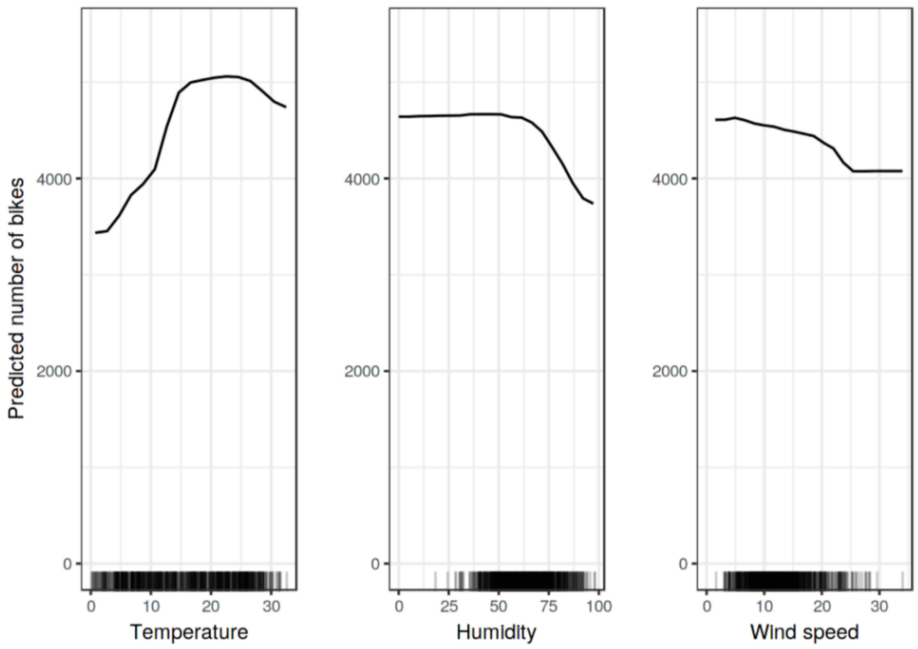
Partial Dependence Plots

A partial dependence plot shows the effect of a feature on the outcome of a ML model.

$$\hat{f}_{x_S}(x_S) = E_{x_C}\left[\hat{f}(x_S, x_C)\right] = \int \hat{f}(x_S, x_C) d\mathbb{P}(x_C)$$

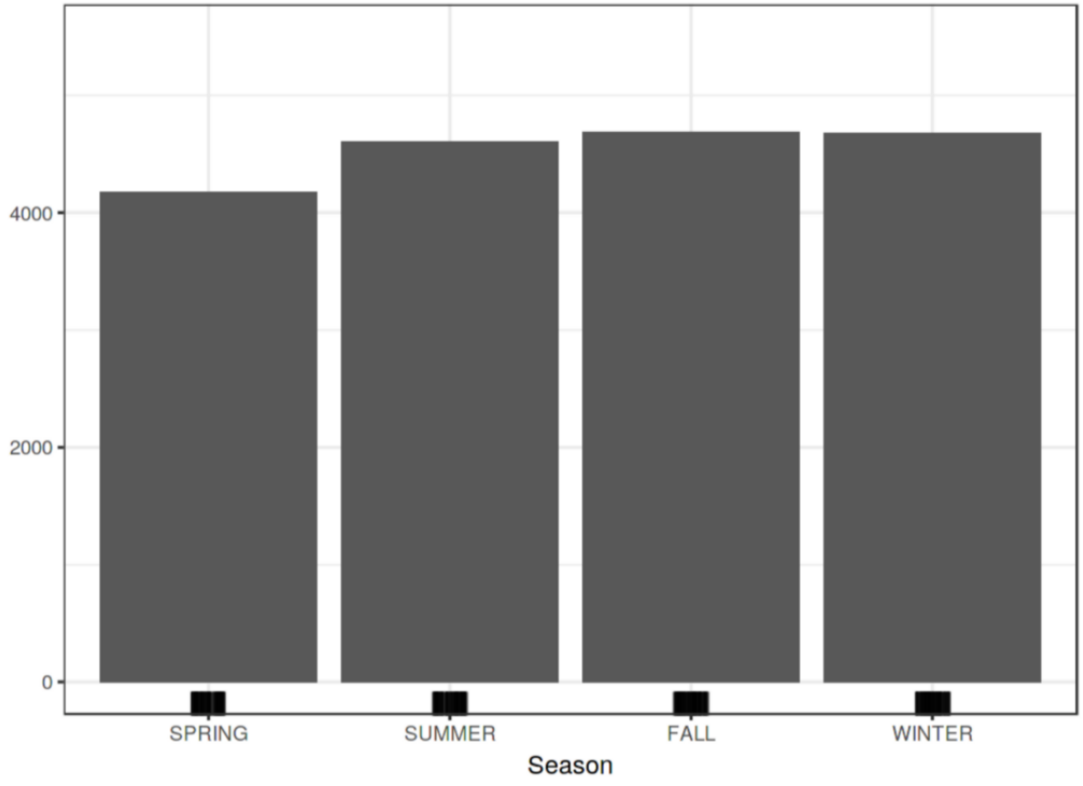
Partial dependence equation for regression

Partial dependence works by marginalizing the machine learning model output over the distribution of the features we are not interested in (denoted by features in set C). This makes it such that the partial dependence function shows the relationship between the features we do care about (which we denote by buying in set S) and the predicted outcome. By marginalizing over the other features, we get a function that depends only on features in S. This makes it easy to understand how varying a specific feature influences model predictions. For example, here are 3 PDP plots for Temperature, Humidity and Wind Speed as relating to predicted bike sales by a linear model.



PDP's for temperature, humidity and wind speed for a regression problem where the number of bikes is the outcome. From these plots, it's clear to see that temperature is an important determiner of how many bikes arre rented, and the hotter it is, the more bikes are rented.

PDP's can even be used for categorical features. Here's one for the effect of season on bike rental.



Partial dependence plot for the effect of seasons on bike rentals

For classification, the partial dependence plot displays the probability for a certain class given different values for features. A good way to deal with multi-class problems is to have one PDP per class.

The partial dependence plot method is useful because it is global. It makes a point about the global relationship between a certain feature and a target outcome across all values of that feature.

Advantages

Partial Dependence Plots are highly intuitive. The partial dependence function for a feature at a value represents the average prediction if we have all data points assume that feature value.

Disadvantages

You can really only model a maximum of two features using the partial dependence function.

Assumption of independence: You are assuming the features that you are plotting are not correlated with any other features. For example, if you are predicting blood pressure off of height and weight, you have to assume that height is not correlated with weight. The reason this is the case is that you have to average over the marginal distribution of weight if you are plotting height (or vice-versa). This means, for example, that you can have very small weights for someone who is quite tall, which you probably not see in your actual dataset.

Great! I want to implement a PDP for my model. Where do I start?

Here's an implementation with with scikit-learn.

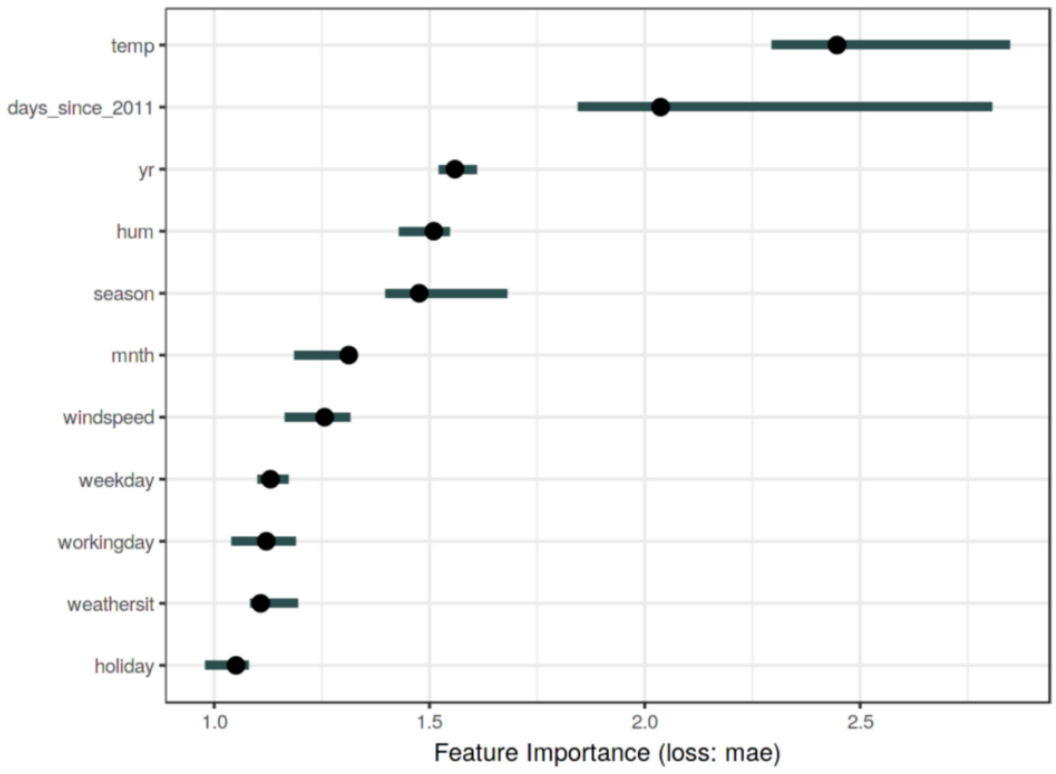
Permutation Feature Importance

Permutation feature importance is a way to measure the importance of a feature by calculating change in a model's prediction error after permuting the feature. A feature is "important" if permuting its values increases the model error, and "unimportant" if permuting the values leaves the model error unchanged.

The algorithm works as follows:

```
input: a model f, feature matrix X, target vector Y and error measure L(y, f)1. Estimate the original model error e^0 = L(Y, f(x))2. For each feature j:- Generate feature matrix X' by permuting feature j in the original feature matrix X.- Estimate the new error e^1=L(Y, f(X')) based off the model's predictions for the new data X'- Calculate the permutation feature importance FI=e^1/e^0. You can also use e^1-e^0.3. Sort the features by descending FI.
```

After you have sorted the features by descending FI, you can plot the results. Here is the permutation feature importance plot for the bike rentals problem.



Permutation feature importance plot for bike rentals. You can clearly see that the model sees temperature and days since 2011 as the most important features.

Advantages

Interpretability: Feature importance is just how much the error increases when a feature is distorted. This is easy to explain and visualize.

Permutation feature importance provides global insight into the model's behavior.

Permutation feature importance does not require training a new model or retraining an existing model, simply shuffling features around.

Disadvantages

It's not clear whether you should use training or test data for your plot.

If features are correlated, you can get unrealistic samples after permuting features, biasing the outcome.

Adding a correlated feature to your model can decrease the importance of another feature.

Great! I want to implement Permutation Feature Importance for my model. Where do I start?

Here's an implementation with the eli5 model in Python.

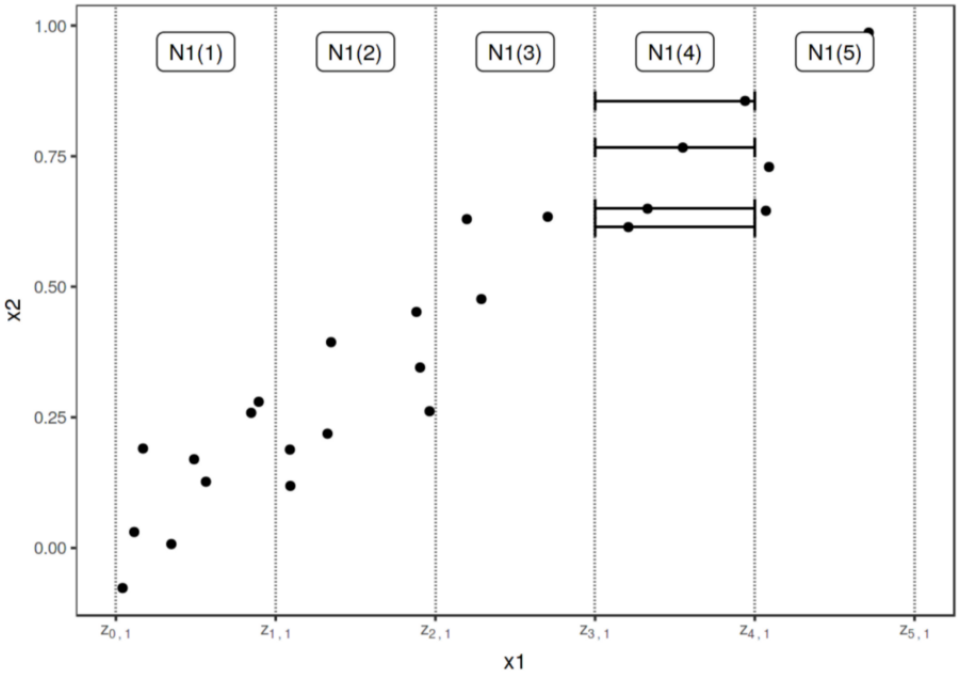
Accumulated Local Effects Plot

ALE plots are a faster and unbiased alternative to partial dependence plots. They measure how features influence the prediction of a model. Because they are unbiased, they handle correlated features much better than PDP's do.

If features of a machine learning model are correlated, the partial dependence plot cannot be trusted, because you can generate samples that are very unlikely in reality by varying a single feature. ALE plots solve this problem by calculating – also based on the conditional distribution of the features – differences in predictions instead of averages. One way to interpret this is by thinking of the ALE as saying

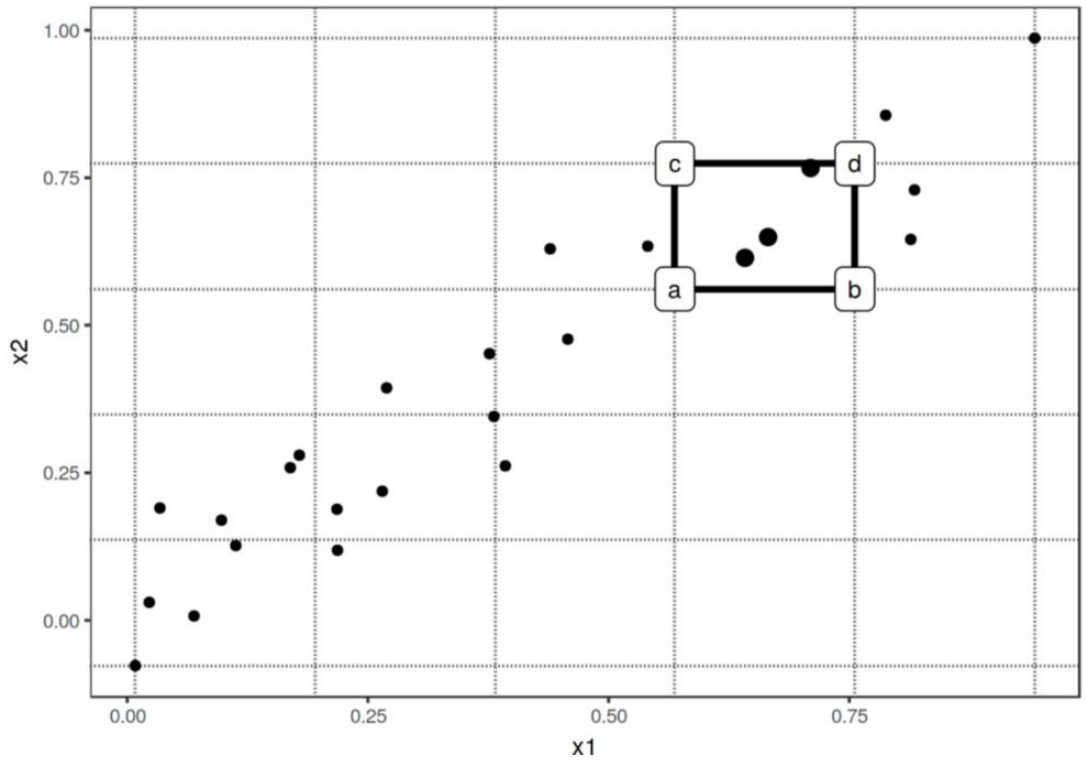
"Let me show you how the model predictions change in a small "window" of the feature."

Here's a visual interpretation of what is going on in an ALE plot.



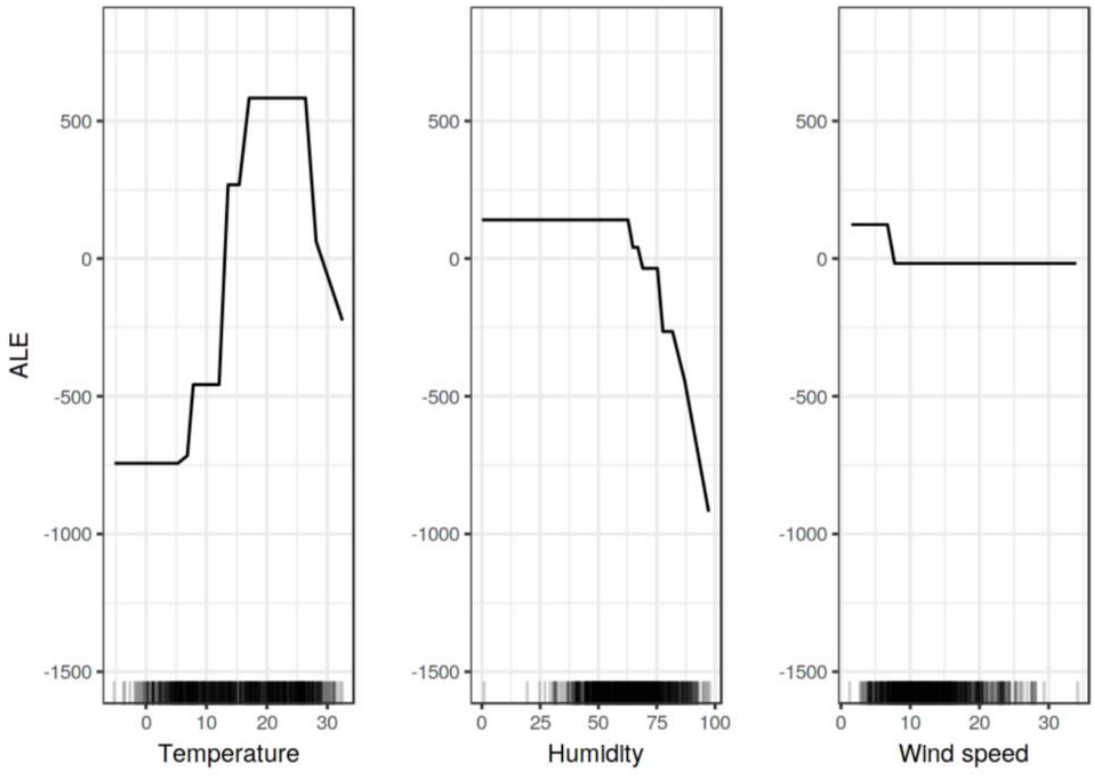
ALE plot calculation. Divide x1 into "windows". For all the points in each window, calculate the difference in prediction when we replace each point with the upper and lower bounds of the window.

This can also be done with two features.



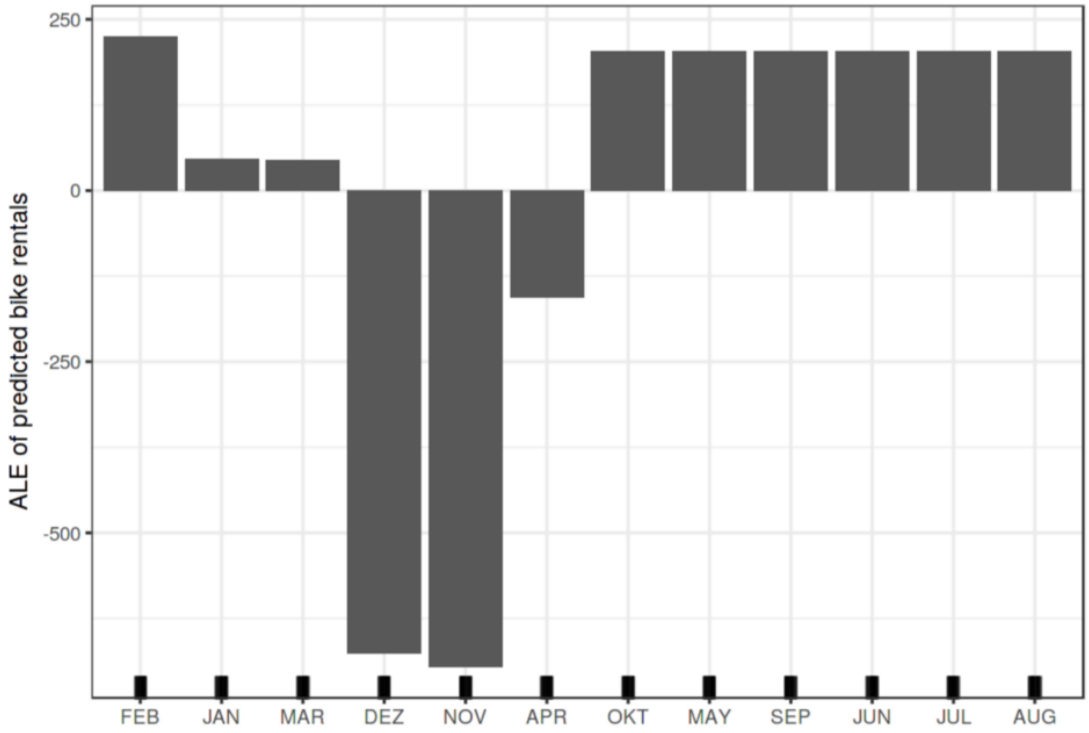
2D-ALE plot. Same general idea as the 1D plot, but instead of using the upper and lower “window” bounds, you calculate the difference in prediction over the four corners of the square in the grid.

Once you have computed the differences in predictions over each window, you can generate an ALE plot.



ALE plot for bike rentals. Clearly, temperature has a strong effect on bike rental prediction, but you can also see that if humidity is above a certain point, it has a strong effect on bike rental prediction as well.

ALE plots can also be done for categorical features.



ALE plot for month and bike rentals. January and March seem to have little effect on the number of bikes rented, but December and November seem to have a large negative effect.

Advantages

ALE plots are unbiased, meaning they work with correlated features.

ALE plots are computationally fast to compute.

The interpretation of the ALE plot is clear.

Disadvantages

The implementation of ALE plots is complicated and difficult to understand.

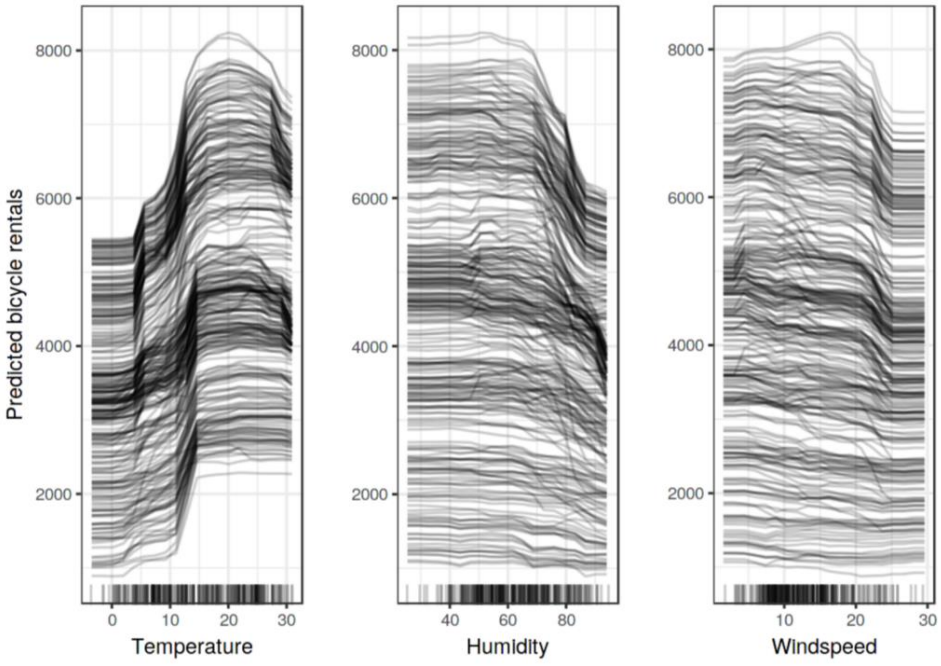
Interpretation still remains difficult if features are strongly correlated.

Second-order or 2D ALE plots can be hard to interpret.

Generally, it is better to use ALE's over PDP's, especially if you expect correlated features. [Here's a library that provides an ALE implementation.]

Individual Conditional Expectation

Individual Conditional Expectation (ICE) plots display one line per data point. It produces a plot that shows how the model's prediction for a data point changes as a feature varies across all data points in a set. For the plot below, you can see the ICE plots for varying temperature, humidity and wind speed across all instances in the training set bike rental data.



ICE plot for bike-sharing

Looking at this plot, you may ask yourself: what is the point of looking at an ICE plot instead of a PDP? It seems much less interpretable.

PDPs can only show you what the average relationship between what a feature and a prediction looks like. This only works well if the interactions between the features for which the PDP is calculated and the other features are uncorrelated, but in the case of strong, correlated interactions, the ICE plot will be more insightful.

Advantages

Like PDP plots, ICE plots are very intuitive to understand.

ICE plots can uncover heterogeneous relationships better than PDP plots can.

Disadvantages

ICE curves can only display one feature at a time.

The plots generated by this method can be hard to read and overcrowded.

[Here's an overview of interpretability with an ICE implementation.]

Surrogate Models

A surrogate model is an interpretable model (such as a decision tree or linear model) that is trained to approximate the predictions of a black box. We can understand the black box better by interpreting the surrogate model's decisions.

The algorithm for generating a surrogate model is straightforward.

```
1. Select a dataset X that you can run your black box model on.2. For the selected dataset X, get the predictions of your black box model. 3. Select an interpretable model type (e.g. linear model or decision tree)4. Train the interpretable model on the dataset X and the black box's predictions. 5. Measure how well the surrogate model replicates the predictions of the black box model. 6. Interpret the surrogate model.
```

One way to measure how well the surrogate replicates the black box through the R-squared metric:

$$R^2 = 1 - \frac{SSE}{SST} = 1 - \frac{\sum_{i=1}^{n}(\hat{y}_*^{(i)} - \hat{y}^{(i)})^2}{\sum_{i=1}^{n}(\hat{y}^{(i)} - \bar{\hat{y}})^2}$$

The R-squared metric is a way to measure the variance captured by the surrogate model. An R-squared value close to 1 implies the surrogate model captures the variance well, and close to 0 implies that it is capturing very little variance, and not explaining the black box model well.

Advantages

This approach is intuitive: you are learning what the black box model thinks is important by approximating it.

Easy to measure: It's clear how well the interpretable model performs in approximating the black box through the R-squared metric.

Disadvantages

The linear model may not approximate the black box model well.

You are drawing conclusions about the black box model and not the actual data, as you are using the black box's model predictions as labels without seeing the ground truth.

Even if you do approximate the black box model well, the explainability of the "interpretable" model may not actually represent what the black box model has learned.

It may be difficult to explain the interpretable model.

Here's an overview of interpretability with a surrogate model implementation.

The future of interpretability

As machine learning becomes more prominent in daily life, the future of interpretability is more important than ever before. I believe a few trends will categorize the future of interpretability, and this will shape how we interact with AI models in the future.

Model agnostic interpretability focus

All the trends in deep learning research point to the fact that deep networks are not saturating with our current computing and data limits. It's important to realize that as our models get deeper and deeper in everything from image recognition to text generation, there is a need for methods that can provide interpretability across all types of models. The generalizability aspect will become more and more useful the more machine learning takes a hold in different fields. The methods I discussed in this blog post are a start, but we need to take interpretability more seriously as a whole to better understand why the machine learning systems powering our day-to-day are making the decisions they do.

Models that explain themselves

One trend that I have not seen take hold in most ML systems that I believe will exist in the future is the idea of a self-explainable model. Most systems today simply make a decision with reasons that are opaque to the user. In the future, I believe that will change. If a self-driving car makes a decision to stop, we will know why. If Alexa cannot understand our sentence, it will tell us in specific detail

what went wrong and how we can phrase our query more clearly. With models that explain themselves, we can better understand how the ML systems in our lives work.

Increased model scrutiny

Finally, I believe that we as a society have pushed black-box model scrutiny under the rug. We do not understand the decisions that our models are making, and that doesn't seem to be bothering anyone in particular. This will have to change in the future. Engineers and Data Scientists will be held accountable as models start to make mistakes, and this will lead to a trend where we examine the decisions our model makes with the same rigor we would a dataset that the model is trained on.

Works Cited: Most of the examples here inspired from the excellent Interpretable Machine Learning book. (Molnar, Christoph. "Interpretable machine learning. A Guide for Making Black Box Models Explainable", 2019. https://christophm.github.io/interpretable-ml-book/)

Beginner's Guide to Machine Learning Explainability

Deepak Moonat — June 17, 2021 https://www.analyticsvidhya.com/blog/2021/06/beginners-guide-to-machine-learning-explainability/

What does Interpretability/Explainability mean in AI?

The following points encompass the explainability-

- How the internal working of the so-called black box(machine/deep learning models) can be interpreted/explained in human terms
- In addition to the outcome from our model if we know why it has arrived at such output i.e output backed by proper explanation, will be good to drive the business
- How can we build trust in such black box models?
- Scope:Local(per sample) or Global(overall) interpretation

Algorithms such as linear/logistic regression are easy to interpret based on model coefficients and Tree-based algorithms which can help us understand how it's making decisions, with built-in support for feature importance and visualization.

What about complex Machine learning algorithms?
Permutation Feature Importance

One simple method is Permutation Feature Importance, It is a model inspection technique that can be used for any fitted estimator when the data is tabular. The permutation feature importance is defined to be the decrease in a model score when a single feature value is randomly shuffled. This procedure breaks the relationship between the feature and the target, thus the drop in the model score is indicative of how much the model depends on the feature.

Algorithm

- Train a model on a dataset
- Calculate the error metrics or score(s) for the trained model for reference
- Shuffled the validation data columns values one column(features) at a time and can be repeated K times
- Using the shuffled data to evaluate the trained model using the same error or score metrics for each iteration
- The features that affect the error metrics most are the important ones as it indicates a model dependency on that features
- We can calculate the importance score for each feature as

$$i_j = s - \frac{1}{K}\sum_{k=1}^{K} s_{k,j}$$

source

Where,

$\mathbf{i}_j$ is the feature importance of feature j

s is the reference score calculated from the trained model

K is the number of iterations(shuffling operations) performed for a feature

$s_{k,j}$ is the score for k^{th} iteration on feature **j**

We can use the python sklearn package for build-in **permutation_importance** function

```
from sklearn.inspection import permutation_importance
```

We will be using the Xgboost algorithm with default parameters on the Boston dataset

```
xg = xgboost.XGBRegressor()
xg.fit(Xtrain, ytrain)
```

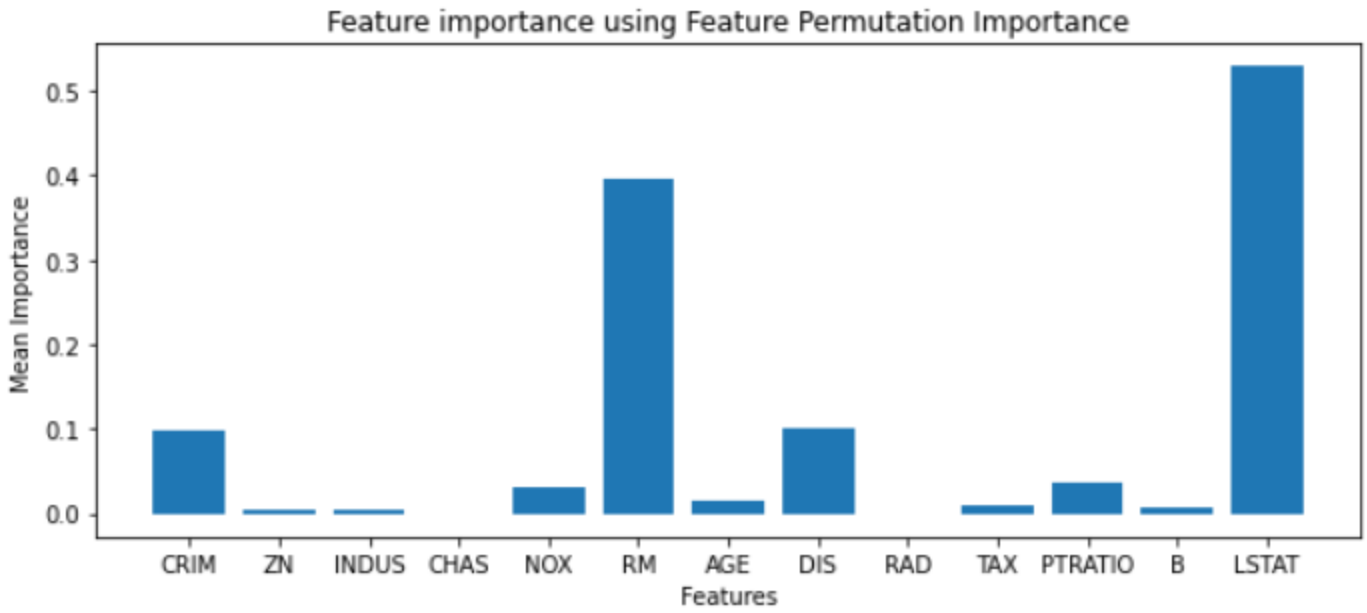
Now we will use the **permutation_importance** function on the test set to calculate the feature importance, we need to provide a trained model and number of shuffling iteration to perform (n_repeats parameter)

```
r = permutation_importance(xg, Xtest, ytest,
                           n_repeats=30,
                           random_state=0)
```

Here, we have set *n_repeats=30*

The returned object, **r** will contain feature importance values for each feature which we can visualize using the **matplotlib** python library

```
plt.figure(figsize=(10,4))
plt.bar(boston.feature_names,r.importances_mean)
plt.xlabel('Features')
plt.ylabel('Mean Importance')
plt.title('Feature importance using Feature Permutation Importance');
```



Source: Github

Caveats

If two features are correlated and one of them is permuted then

- The model still has the other correlated feature and in that case, both features will have lower importance value even if those are actually important
- It can be biased by unrealistic data instances formed by permutation

Computationally expensive for a large number of features

A good practice is to drop one of the correlated features based on domain understanding and try to apply the Permutation Feature Importance algorithm which will provide better feature understanding.

Let's discuss another method to interpret the black box models.

Global surrogate models

- An interpretation model trained to approximate the predictions of the black-box models
- It's like solving black box Interpretability task using simpler and explainable models (such as linear regression, decision tree,..) i.e explaining machine learning using more machine learning

How does it work?

- Get the predictions(yhat) from the black-box model
- Select any simple and explainable model(linear reg., decision tree..) as per the use case
- Train the selected model on the same dataset used for training the black-box model, using predictions(yhat) as the target
- Measure the performance, as to how well the surrogate model approximates the behavior of the black-box model
- Finally, we can interpret the global surrogate model

Advantages:

Flexible, as a selection of surrogate model, does not depend on the black-box model. If at some time we have a better performing black-box model in place of an existing black box, we do not have to change the method of interpretation

Disadvantages

How much confidence is enough(% of variance explained) in deciding the surrogate model is close enough to the black-box model

Interpretation becomes irrelevant if the black-box model is not performing well

LIME (Local Interpretation Model agnostic Explanation)

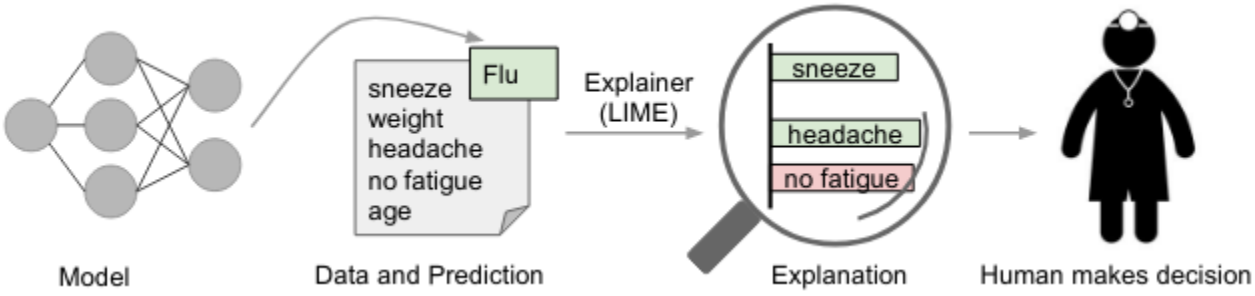
We have seen few model interpretation techniques for Global interpretation, what about Local Interpretation i.e when we want to understand how a model prediction was made for a particular observation.

Consider a loan approval model, what if a user request is declined, then the user has the right to question WHY? and authorities should know why the model has declined the user request and communicate the same to the user as they just can't say that their system has rejected it instead they need to explain on what factors(features) the request is rejected.

LIME can explain the predictions of any classifier or regressor in a faithful way, by approximating it locally with an interpretable model(linear reg., decision tree..)

It tests what happens to the predictions when we feed variations of the data into the machine learning model

Can be used on tabular, text, and image data



Source: LIME Paper

We can use python **lime** library to interpret models

```
import lime
import lime.lime_tabular
```

We will be using the Xgboost algorithm with default parameters on the Boston dataset

```
xg = xgboost.XGBRegressor()
xg.fit(Xtrain, ytrain)
```

Now we will create a lime explainer object, for which we have to specify the target column, features names, categorical features, and mode of the algorithm(regression or classification)

```
explainer = lime.lime_tabular.LimeTabularExplainer(Xtrain,

feature_names=boston.feature_names,
                                                   class_names=['price'],

categorical_features=categorical_features,
```

```
                                                    verbose=True,
mode='regression')
```

We can use **the explain_instance** method of **the explainer** object to interpret a particular instance of data

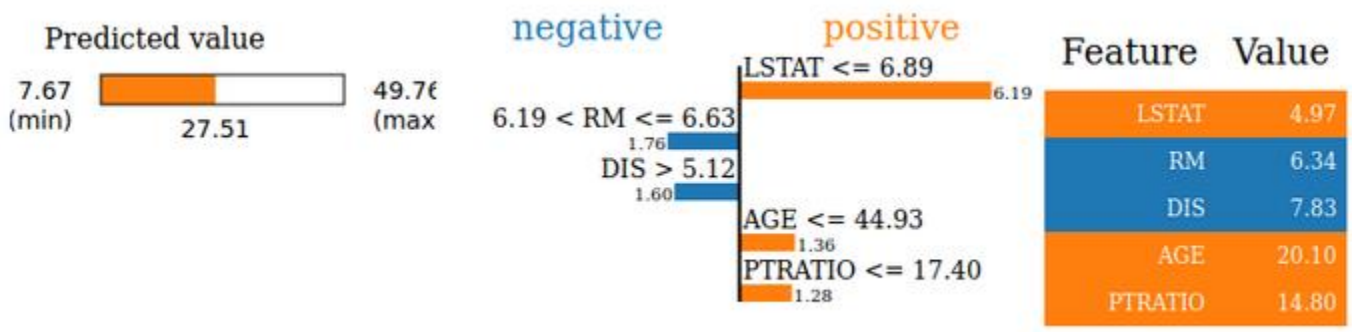
```
exp = explainer.explain_instance(Xtest[i], xg.predict, num_features=5)
```

i is the index in test data that we need to interpret

we can visualize the interpretation output using **the show_in_notebook** method

```
exp.show_in_notebook(show_table=True)
```

We will get similar to below output



Source: Github

For Complete code refer to Github Repository

References

https://christophm.github.io/interpretable-ml-book/feature-importance.html
https://christophm.github.io/interpretable-ml-book/global.html
https://scikit-learn.org/stable/modules/permutation_importance.html
https://scikit-learn.org/stable/auto_examples/inspection/plot_permutation_importance_multicollinear.html#sphx-glr-auto-examples-inspection-plot-permutation-importance-multicollinear-py
LIME Research Paper: https://arxiv.org/pdf/1602.04938.pdf
https://github.com/marcotcr/lime