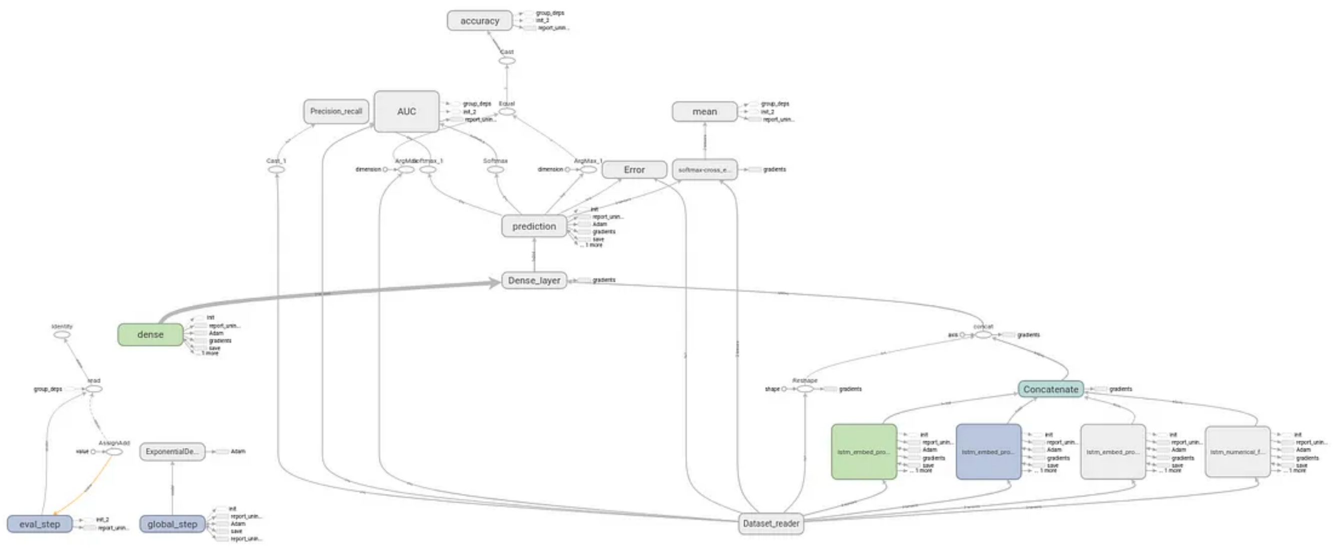
MODEL INTERPRETABILITY

Squeezing LIME in a custom network

How to interpret the interpreter?

Thomas Gorbinet · Follow
Published in Towards Data Science
17 min read · Jan 20, 2021

Rbm, Benjamin Larrousse, Thomas G.



Our custom RNN-LSTM network architecture. (Image by Author)

Machine and deep learning models are applied in a wide range of areas, spanning from fundamental research to industries and services. Their successful application to a wide diversity of problems has further promoted the expansion of the domain: it is experiencing a “golden age” whose end still cannot be seen.

While the development of algorithms for prediction and classification is studded with stories of success, there are several major challenges that must be properly addressed and that are currently the main topic of several research studies.

Model interpretability (which is often referred to as “*Explainable Artificial Intelligence* (XAI)”), is among these problems. The core of the problem is providing a way to understand and explain the rationale behind predictive algorithms that would otherwise behave as black boxes.

There are various reasons for requiring a better understanding of the results of a machine learning model, but to list a few we can think of:

- gain deeper insight into the mechanism that the model is reproducing. We can think of cases in which machine learning is applied in fundamental research, where it is not enough to have an algorithm mimicking the observed behaviour, but it is important to understand the mechanism behind it;
- increase the trust of the end user toward the results of the model. Owing to the various domains in which artificial intelligence is applied, the final user of the algorithm is probably an expert in the domain of application; however, he or she does not necessarily have a background in machine learning. By gaining an understanding of the model results through the use of algorithms for interpretability, he or she can consciously apply or refuse the information obtained from the predictive algorithm;
- legal issues. The rise of machine learning applications has seen an increase of regulations issued for guaranteeing the

proper use of data necessary for the functioning of algorithms. Respecting the legal constraints (e.g, the *right to explanation* included in the GDPR) requires a proper understanding of the rationale of an algorithm. This understanding can be acquired with an algorithm for interpretability.

In addition, the explanation of the model results can serve as an additional tool for evaluating the validity of the algorithm during its development.

In the wake of this (relatively) new trend, we decided to apply one of the different available algorithms for interpretability to a model that predicts the probability of purchase of a customer. The chosen algorithm is LIME (acronym for *Local Interpretable Model-agnostic Explanations*).

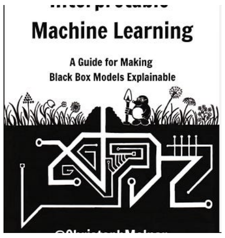
The next sections describe the functioning of the predictive algorithm and briefly introduce the LIME explainer. Finally the results of its application to our use case are presented.

Details of the theory of interpretability and available models are omitted in this article because literature provides sufficient material on this topic. Those who are interested in further exploring the subject can start with the well-known book of Christoph Molnar as a starting point.

Interpretable Machine Learning

Machine learning has great potential for improving products, processes and research. But computers usually do not…

christophm.github.io



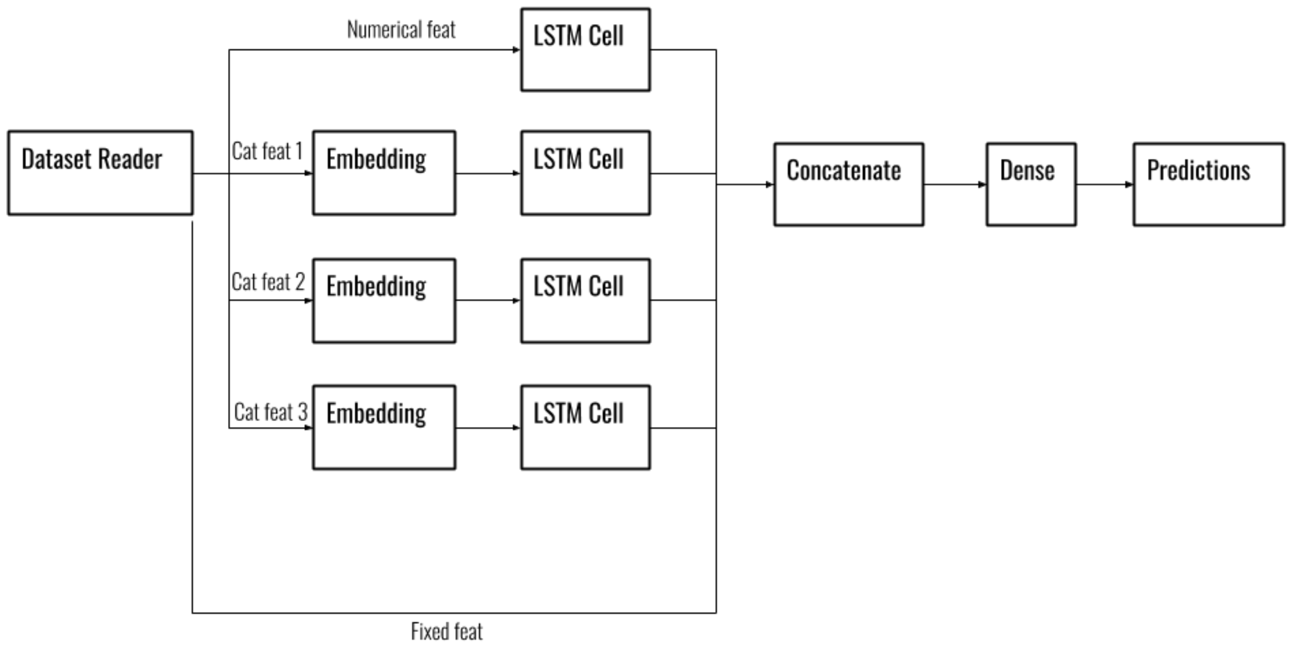
Our algorithm of interest

The model we will try to interpret is a deep learning algorithm which predicts the probability of a client purchasing an item in the next *X* days (where *X* is typically 7 or 28), based on the navigation history of this client: page views, baskets, purchases both online and in physical stores. It gives us sequences of events that characterize users' behavior. It is thus possible to train an algorithm for estimating the purchase probability based on this data.

The input length (i.e., the number of events in the client's history) is not fixed and can be arbitrarily large. In addition, each event has a number of characteristics (features), which make the dataset quite big. For these reasons, we chose to use a deep learning algorithm, and specifically a Long Short-Term Memory (LSTM) which is a type of Recurrent Neural Network (RNN). LSTM networks are well suited for inputs of arbitrarily large lengths and can also keep into memory information from relatively old events.

To use the proposed LSTM effectively, we started with data preprocessing in order to shape the raw data.. Not all features are numerical; thus, their specificities (e.g., the types and range of values) must be considered. This step is very important and can reduce memory usage and increase the quality of predictions.

The LSTM network is designed in *Tensorflow 1.15*. It is not at all a vanilla network: given the specificities of our data inputs, we constructed a custom architecture with separated LSTM cells for numerical features and categorical features The following figure presents an illustration of the architecture.



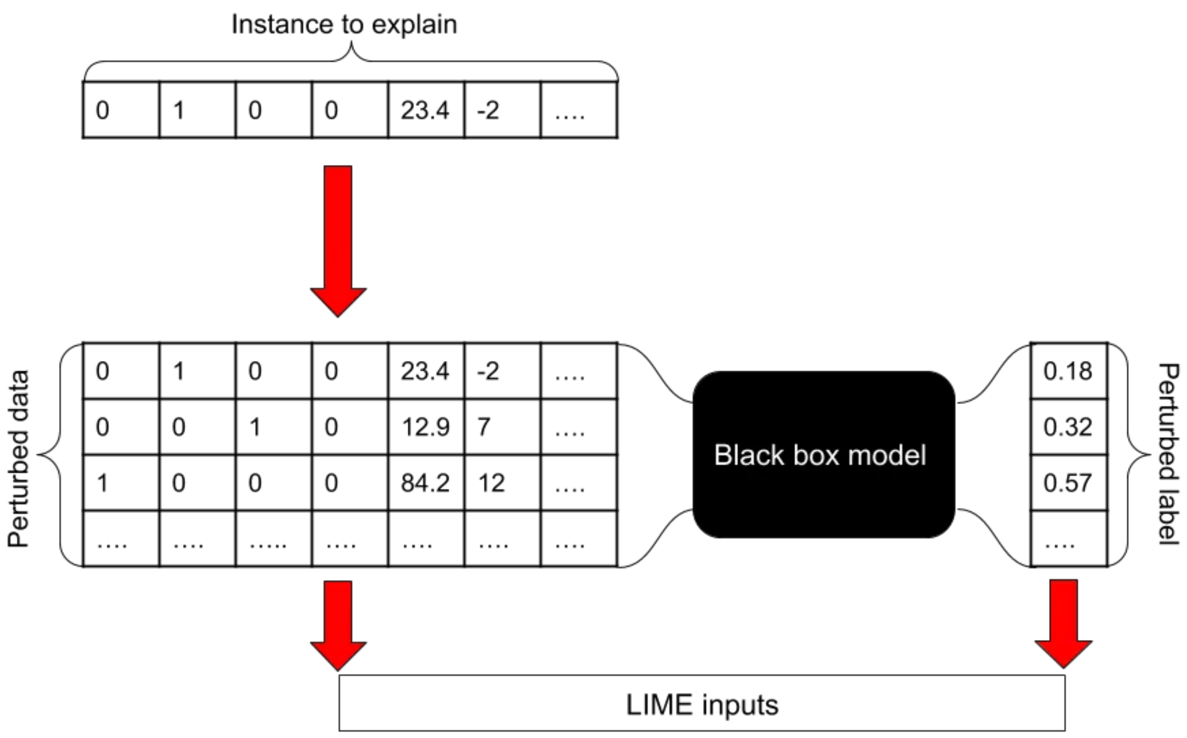
Schematic of the model for purchase probability. (Image by Author)

The custom nature of the architecture alongside the high number of features (~50) for each event make the interpretation complex in many ways: many dimensions must be considered, and as our architecture is not standard, *off-the-shelf* interpretable algorithms can not be used directly. Nonetheless, these algorithms are a starting point to what we want to achieve in this study.

(Yet another) primer on LIME

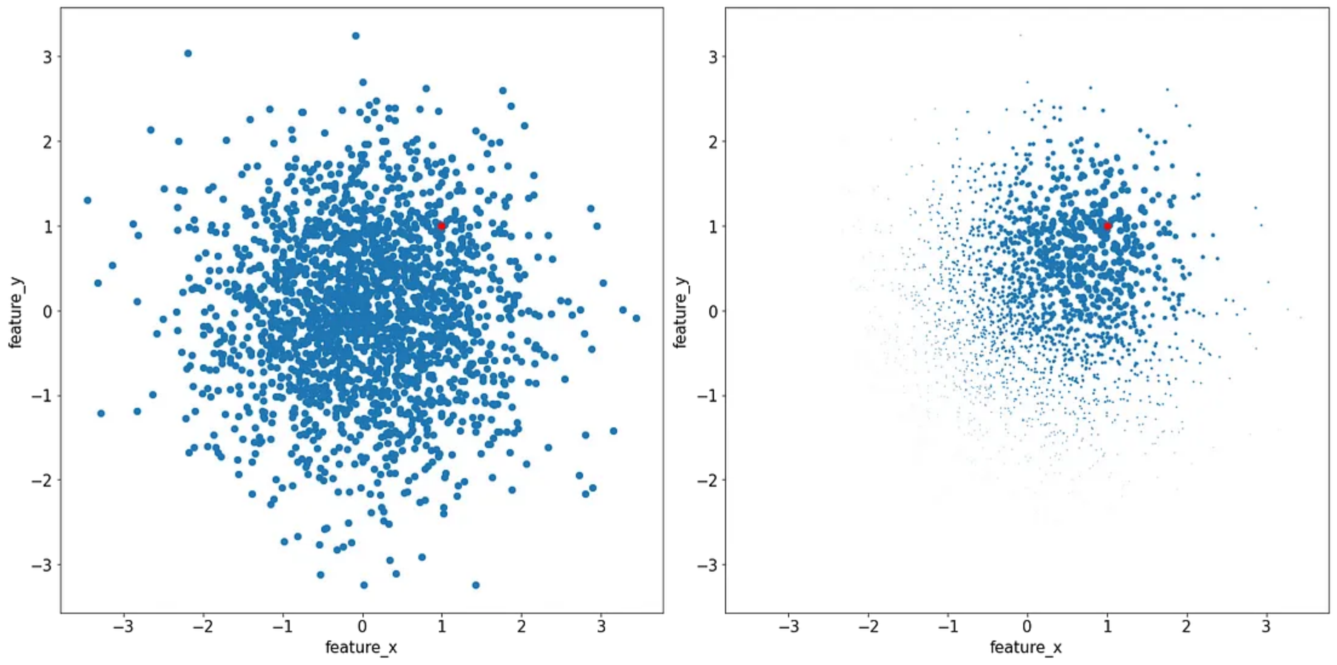
LIME is a local surrogate model; thus, the interpretable model, which is usually a linear one or a tree, is fitted in a neighborhood of the instance to explain. Being model-agnostic, LIME can be applied to any kind of desired algorithm, including deep neural networks.

The set of training data necessary for fitting the local explainer is generated by LIME by perturbing the instance to explain. The corresponding labels are obtained by passing the perturbed data to the model to explain.



Schema of the generation of input data and labels for LIME. (Image by Author)

The interpretable model is trained by minimizing the loss function (which is a weighted RMSE). The weight is defined according to the distance of the perturbed data from the instance to explain; closer data have higher weights.



Example of using synthetic data of the impact of the weights. The red point is the instance to explain, and the blue points represent the perturbed data. In the right plot, the sizes of the points are scaled according to the weights. (Image by Author)

To enforce the interpretability of the model, the minimization of the loss function is constrained by a complexity measure, namely the number of features to be applied in the interpretable model. The procedure aims to ensure local fidelity (i.e., the interpretable model must be a sufficiently good local approximation of the original one) and interpretability (i.e., the interpretable model must have a low complexity).

Despite its simplicity, the LIME algorithm relies on several parameters and definitions that can be tuned by the users and that can affect the final result. The most important one is the definition of the kernel function at the core of the distance-based weights within the loss function. By default, LIME uses an exponential kernel with an Euclidean distance and a default kernel width of 0.75 √(nf) where nf are the number of features.

In LIME, the impact of each feature on the prediction is defined by its weights in the local explainer. This way, an estimate of the relative importance of the different features AND the “direction” of the influence can be obtained: features with positive weights push the prediction toward the selected label, and features with negative weights push the prediction far away from the selected class.

When model and LIME unite

Although LIME is model-agnostic, a direct application of the algorithm to our model is like trying to match two unfitted pieces of a puzzle. It is not possible without first finding the missing pieces between the two, which correspond to the pre- and postprocessing steps in our case.. This is required in order to match the constraints of the predictive model with those of the explainer.

LIME requires:

- a set of training data in the form of a two-dimensional array; these data should be identical to the training data used for the predictive model. LIME evaluates the mean values and variances for each continuous feature and the pairs (value, frequency) for each categorical feature. As an alternative, this step can be avoided by providing LIME with a dictionary containing this information; this is a useful solution when the set of training data is very large;
- the instance to explain must be a two-dimensional array with an identical number of features as the training data;
- the original model must have a predict function.

These last two requirements do not pose any particular constraint on models that resemble those available in, for example, the scikit-learn package. In our case, these requirements can be strict.

Our model for estimating the probability of purchase does not fulfill these requirements. The input of the model is composed of three-dimensional arrays for each LSTM cell and of additional two-dimensional arrays required for the dense layer. The generation of perturbed data poses another problem in our application. When perturbing the instance to explain, LIME has no constraints but the statistical properties of the features obtained from the training data. Moreover, some of the features are related to each other; these relations must be addressed when perturbed data are generated. This is the case for example for the *day of the week* variable in our model, represented by a one-hot-encoded vector. The default perturbation in LIME does not take into account this constraint, thereby eventually generating vectors containing more than just one element different from zero.

To address all these issues we designed a series of preprocessing steps to map the inputs of the original model in a form suitable for LIME that ensures the consistency of the perturbed inputs. Evidently the transformation must be inverted when the perturbed data generated from LIME are fed as inputs into the original model. This kind of approach is not only related to LIME and has been observed as a general scheme when applying a model-agnostic explainer.

The preprocessing part comprises the following transformations:

- mapping of one-hot-encoded numerical features onto label-encoded categorical features;
- reshaping three-dimensional inputs of LSTM cells to two-dimensional arrays;
- concatenating the different inputs in a single array.

The postprocessing part comprises the following transformations:

- resampling (if necessary) the interdependent categorical features;
- converting (when required by the features) the sampled data to positive values;

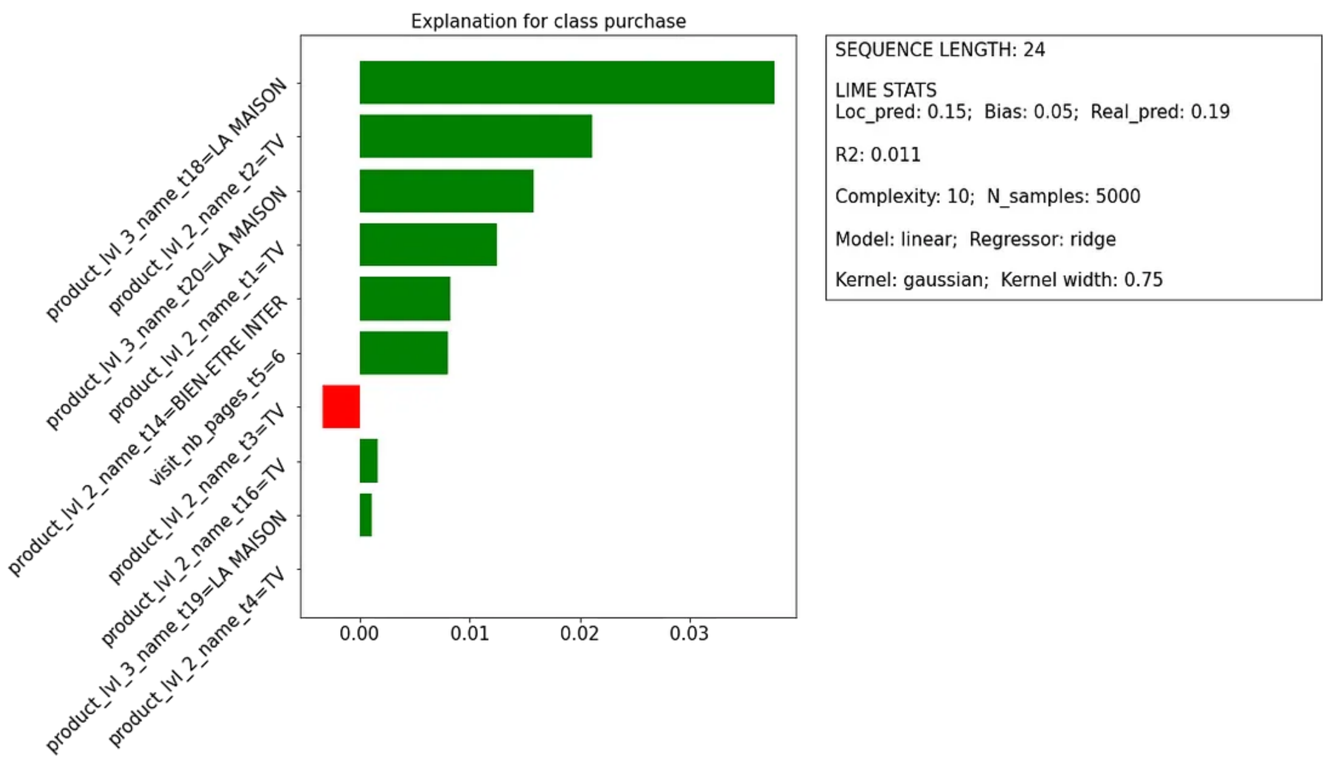
- inverting the transformations in the preprocessing phase.

Check your instruments before using them: parametric analysis for LIME

The algorithm for purchase predictions and LIME were applied to a set of real world data belonging to a french retailer specialized in consumer electronics and household appliances.

To provide an example, we applied LIME to a single instance, which is represented in the context of our algorithm of purchase predictions by a single person with a specific navigation history that can eventually lead to a purchase.

The explanation provided by LIME using the default values of the parameters is presented in the following figure.



SEQUENCE LENGTH: 24

LIME STATS
Loc_pred: 0.15; Bias: 0.05; Real_pred: 0.19

R2: 0.011

Complexity: 10; N_samples: 5000

Model: linear; Regressor: ridge

Kernel: gaussian; Kernel width: 0.75

Example of explanation for a single instance with LIME. (Image by Author)

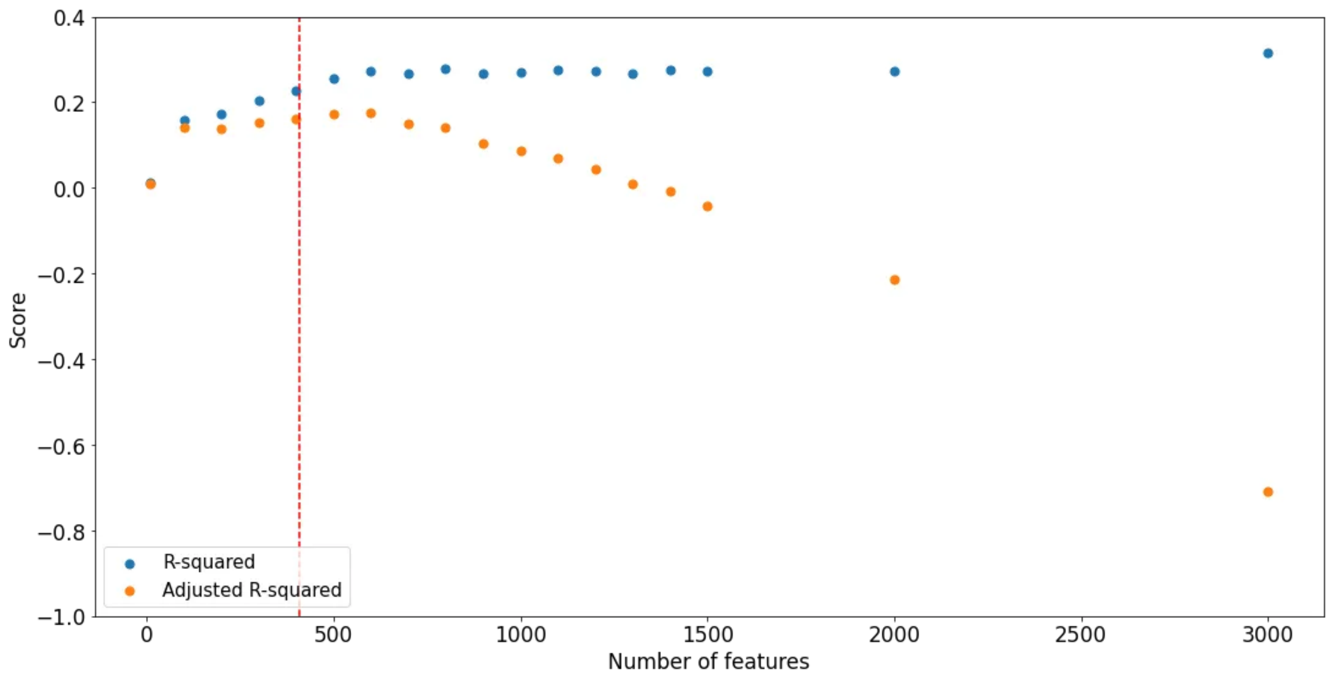
The left panel presents the weights of each feature, which were evaluated by fitting the explainer. The importance of each feature is based on the absolute value of its weights. Owing to the sequential nature of the problem, the weights refer to the pair (feature, step). Therefore, the same feature can have a different effect at different positions in the sequence; for instance, the case "product_lvl_2_name=TV", has a positive impact at t=2, a negative one at t=3 and no impact at t=4.

The right panel lists information on the sequence length of the instance and the total number of sequences composing the explained instance; in addition, some information on the explainer is provided. These figures show that although the local prediction provided by LIME is indeed close to the real value obtained by the original model (15% and 19% probabilities of purchase with LIME and original algorithm respectively), the coefficient of determination (the R-squared value) is extremely low, which suggests that the local model is rather unreliable.

Before throwing LIME in the trash, we can take a minute to think about the reason for this low fidelity...

The first guess is that the complexity of the local model is too low. The original model requires approximately 50 variables as input. When applying the preprocessing transformations mentioned in the previous section, the number of features increases to approximately 400 features for this instance, which are much more than the ten features used to train the local explainer.

To validate this hypothesis, we realized several explanations for the same instance based on exactly the same explainer but varying the complexity. The effects of the number of features on the resulting R-squared value can be seen in the following Figure.



Evolution of the R-squared coefficient and adjusted R-squared as functions of the number of features in the explainer. The vertical dashed line represents the number of real features applied in the original model for the explained instance. (Image by Author)

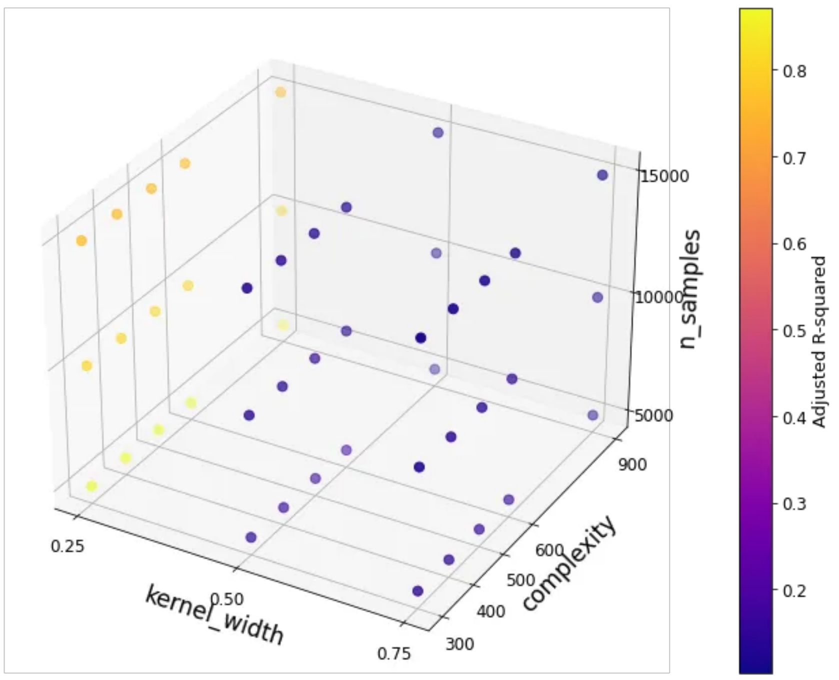
Increasing the number of variables leads indeed to a small increase in the R-squared value; however the adjusted R-squared value reaches its maximum for approximately 600 features. This suggests that the increase in the R-squared is a spurious one induced by the simple addition of new features. The maximum of the adjusted R-squared value is approximately 0.16; this rather small value does not ensure the fidelity of the local explainer.

Interestingly, the maximum is obtained for approximately 600 features, which is higher than the expected approximately 400 features corresponding to the real variables for this instance and represented by the vertical dashed line in the figure. This high number is probably caused by LIME, which generates perturbed data for the entire set of (features, steps) pairs, therefore involving also sequence steps greater than the maxima sequence length of the instance and that should be considered as dummy values. *(We need some context here. The original LSTM model involves inputs with variable sequence lengths, a specificity addressed by the combined use of bucketing and padding. When training the LIME explainer on the other hand, we need to consider the maximum sequence length among all the training data, in order to be able to apply the explainer to any desired instance, no matter their sequence length ; the alternative would be to have an explainer for each possible sequence length in the dataset. Therefore when generating the perturbation, the explainer can include variables belonging to sequence lengths greater than the one of the instance to explain.)*

This simple analysis shows how sensitive the obtained explanation obtained can be to the LIME parameters. The complexity of the explainer is just one of the parameters that can be tuned. The other relevant ones are the number of samples (which represent the number of training data for the explainer) and the kernel width (for sake of simplicity we did not consider the kernel function itself and the definition of distance within the kernel function, while keeping the default definitions unchanged).

To apply LIME properly one must determine the best parameters that provide an understandable explanation (which is represented by a low complexity) without severely affecting the fidelity of the explainer. With this goal in mind, we performed a parametric analysis of the adjusted R-squared score as a function of the complexity of the model, number generated samples, and kernel width. The objective is to find a combination of the three parameters that provides the optimal score. A full scale parametric analysis is computationally expensive. Therefore, we chose a small set of values that are supposedly close to the optimal ones.

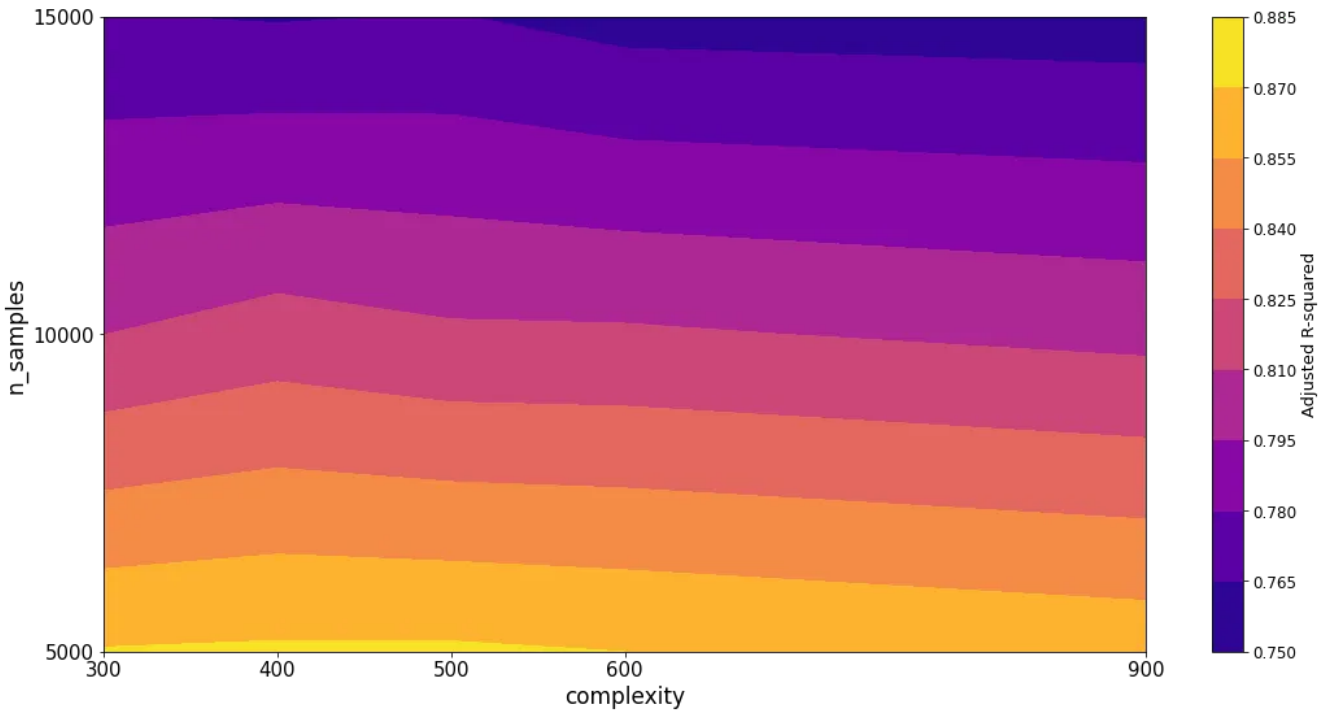
The following Figure clearly shows that the kernel width is the main parameter affecting the adjusted R-squared coefficient; the scores are higher for shorter kernel widths.



Dependence of the adjusted R-squared coefficients on the parameters of the explainer. (Image by Author)

This behavior is reasonable. Reducing the kernel width emphasizes the importance of perturbed data closer to the instance when minimizing the loss function. The smaller the neighborhood of the instance relevant to the explanation, the more likely the original model can be approximated with a linear function because nonlinear effects are less evident. The roles of the remaining two parameters, complexity, and number of samples can be better understood by focusing only on the case in which the kernel width is 0.25 (i.e., the case with the best results).

The Figure below presents an approximate evolution of the score as a function of the two variables. The adjusted R-squared value increases with decreasing the number of samples and complexity (within the tested range tested of this analysis). The optimal score is achieved with 5000 samples and a complexity between 300 and 500.



Dependence of the adjusted R-squared coefficient on the number of features in the explainer and number of generated samples. (Image by Author)

This small but important parametric analysis emphasizes the importance of tuning the LIME explainer for proper use; otherwise, misleading conclusions can be drawn.

According to these results, the best explainer for our model has a kernel width of 0.25, complexity of 500, and 5000 samples.

In the following section, the application of the explainer is presented to provide insight into the results of the original model.

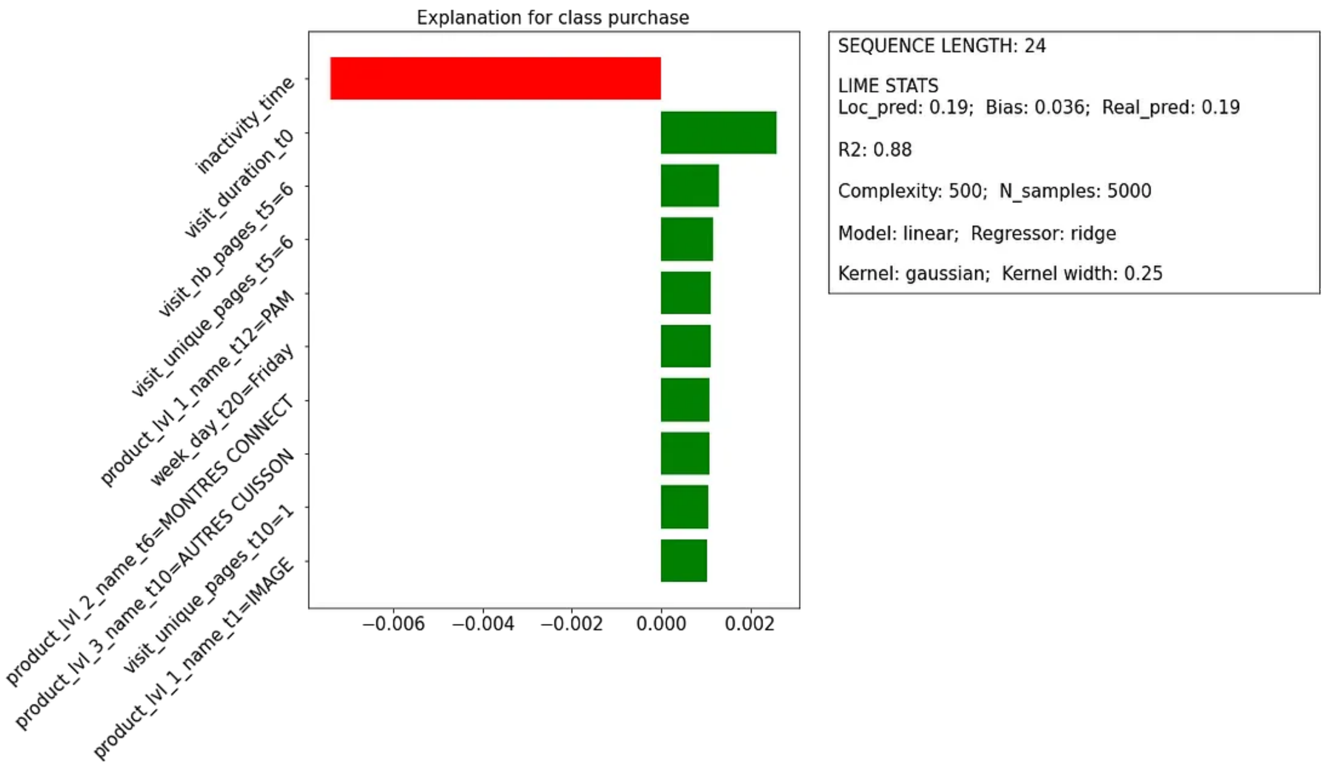
. . .

🥁... explanation is served

Single explanation

Having found a set of parameters ensuring sufficiently good enough fidelity, we can now take a thorough look at the explanation obtained for the instance of interest.

The following Figure presents an analogous explanation to that presented in the previous section; this one was obtained with the new explainer. With a complexity of 500 features, the ten features presented are the top ten features regarding the importance (defined as the absolute value of the weights).



SEQUENCE LENGTH: 24

LIME STATS
Loc_pred: 0.19; Bias: 0.036; Real_pred: 0.19

R2: 0.88

Complexity: 500; N_samples: 5000

Model: linear; Regressor: ridge

Kernel: gaussian; Kernel width: 0.25

Explanation for a single instance with the optimized LIME explainer. (Image by Author)

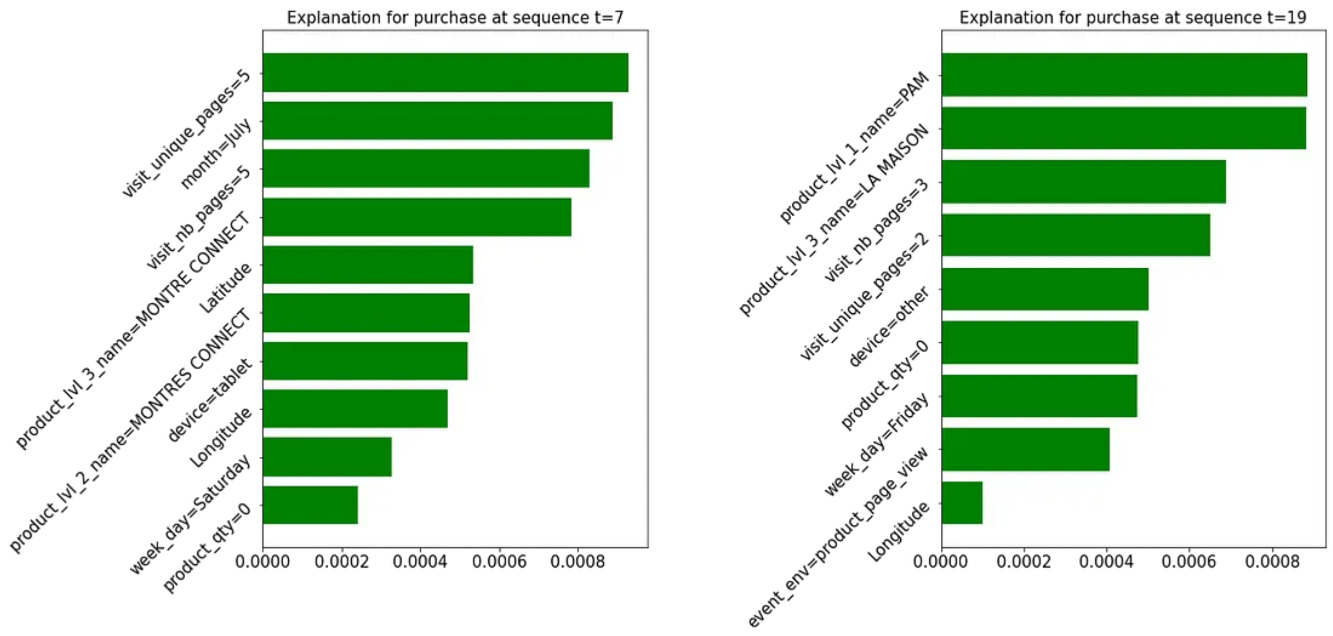
Evidently, the new explanation is substantially different from the previous one. The most important feature (which has a negative impact) is the *inactivity time*; it is defined as the period between the days of the prediction and last interaction of the user. The feature was not present in the previous explanation. All the other features have positive impacts on the purchase. Another difference with respect to the previous case are the small weights of the features. They are probably due to the high number of features used for the explanation.

The presented explanation can indeed provide some insight into the rationale of the original model. The importance of the *inactivity time* appears reasonable when the scope of the model is considered. A long *inactivity time* can suggest a poor interest of the person toward the viewed products, which translates to a weak purchase probability. By contrast, a short *inactivity time* can indicate a still active purchase interest. Therefore, the variable can be an important driver when modeling purchase intentions; the explanation confirms this hypothesis.

The impacts of the other features are more difficult to extract and understand based on the presented explanation. The mixed information including variables and sequence steps does not provide a clear picture of the total impact of each feature. In addition, drawing conclusions from the more important variables for each step is impossible.

To obtain a more thorough insight into these matters, representations of the same explanation were generated by focusing specifically on these aspects.

Explanation per sequence step

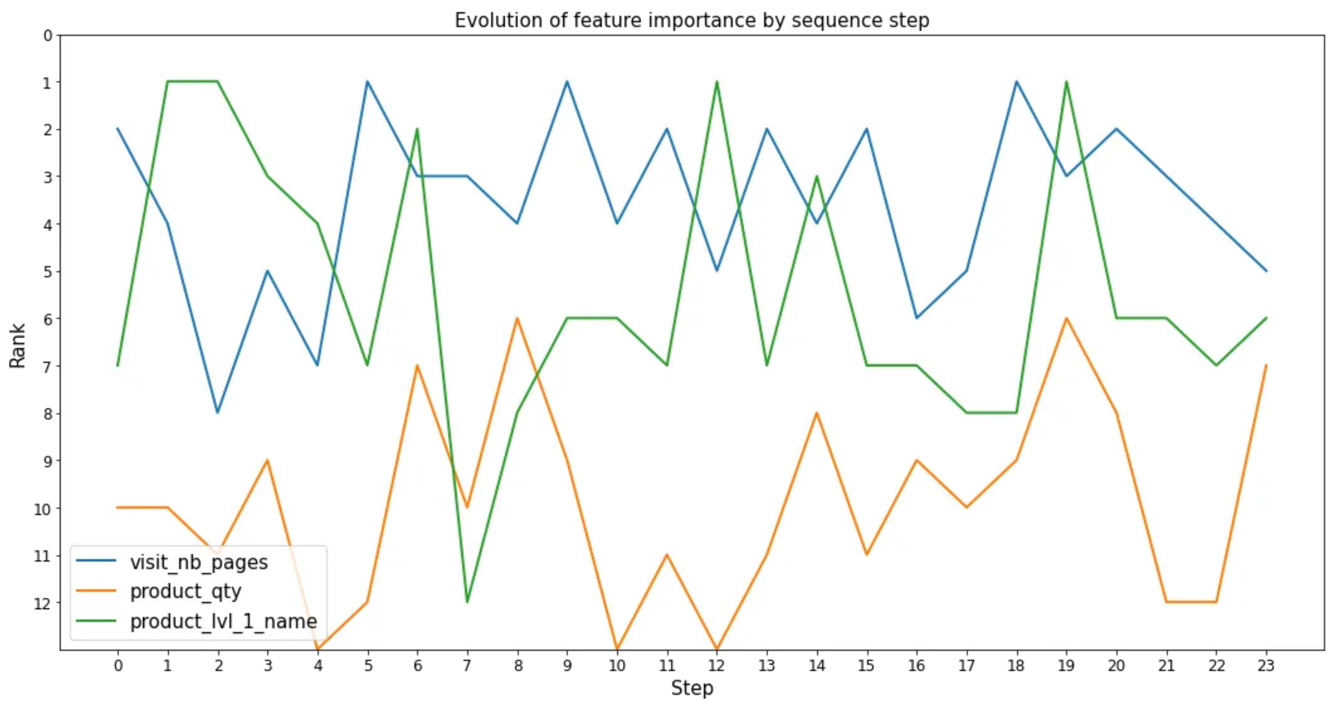


Examples of explanations for two different steps in instance history. (Image by Author)

The Figure above presents the problem of explaining the impact of each feature in each step. The representation is similar to that in the previous section; however, the feature weights refer to a specific step (in this case, t = 7 on the left and t = 19 on the right). In this explanation, features that are independent of the step number (e.g., the *inactivity time*) are not presented.

This description provides additional information that would have been difficult to obtain from the previous one. More specifically, it can be more easily determined which features have a positive or negative impact at a specific step. Moreover, the evolution of the ranking of the features can be observed while moving through the different sessions. For example, at t = 7 the variable *month* ranks second with a positive impact, whereas it is not included in the top ten for t = 19. Similarly, *product_lvl_1_name* is not present at t = 7, whereas it ranks first at t = 19. Of course, this information can be interpreted differently according to the goal of the user. An end user who applies the model to increase the sales can see the importance of the variable *month* at t = 7 as a link to the observed product if it is usually sold in summer. The developers of the model can assume that the variable is important in the first steps of the history to set up an “environment” for the model (e.g., the time of a year or location).

Therefore, the variation of the relative position of a variable across the sequence can be important information that should be extracted and that is not clearly provided by the previous representation.



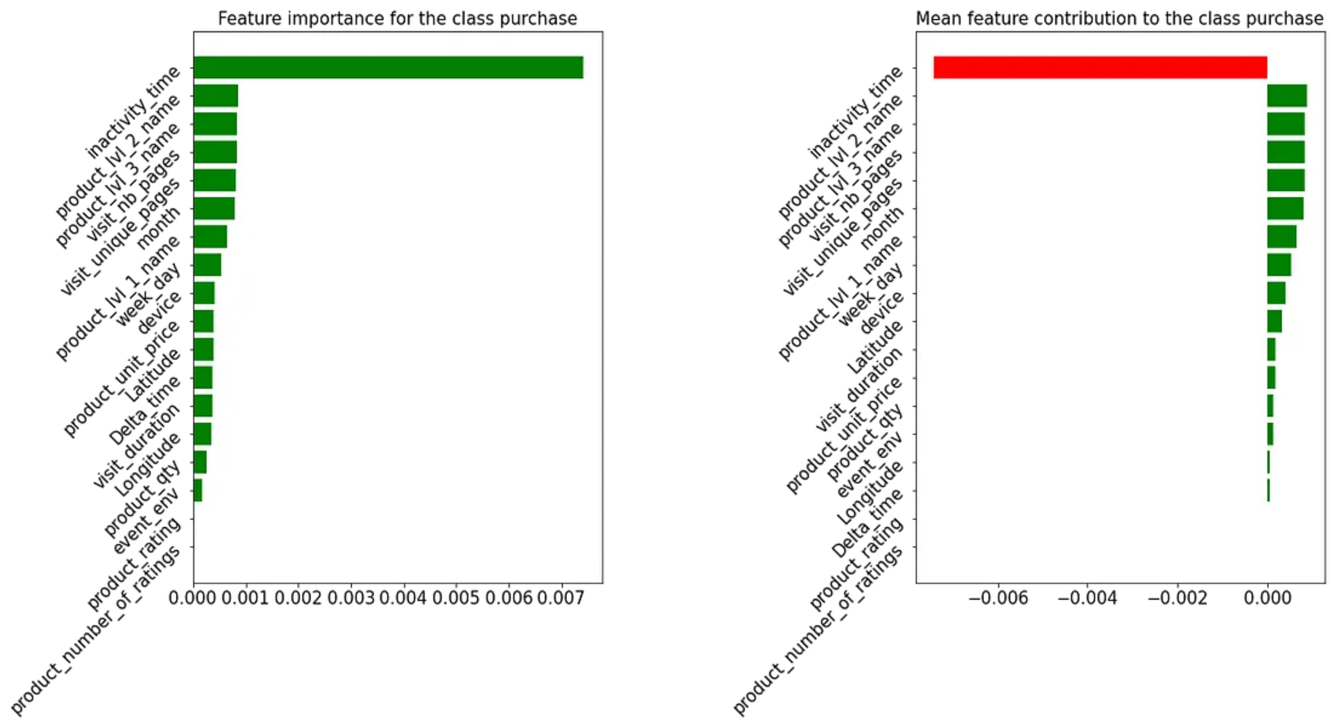
Evolution of the feature importance as a function of the step in the history of the user. (Image by Author)

To address this problem, we prepared a graph (see the Figure above) that presents the evolutions of the ranks of the features with respect to the sequence steps. evidently, the rank information is more relevant than the weights to this type of plot, because the latter is normalised. Nevertheless, this plot does not contain the information on the way the feature influences the prediction. The evolution of the rank provides additional insight into the model. For example, the feature *visit_nb_pages* is almost always in the top five, which underlines its importance. The feature *product_lvl_1_name* acquires importance only in specific steps, which may indicate the start of a new session or the visualization of a new product. Once again, the type of insight information that can be extracted from the figure also depends on the end user.

Explanation per feature

Although the previous plots focus on extracting information related to the single steps of the history from the explanation, an overall understanding of the global importance of the variables for the entire history is still lacking.

The following Figure addresses this problem. Both results were obtained by aggregating the weights of the features for the entire sequence. On the left, the variables are ranked based on the absolute importance, which is defined by the average of the absolute value of the weights for each feature across the path. On the right, the direction of influence of the feature is maintained by evaluating the actual mean weight across the path. Evidently, in this figure, the weights are not anymore related to the fitted linear model (except the variable *inactivity time*); instead, they serve as simple measures of importance of the features.



Feature importance (left) and feature impact (right) averaged over the entire history of the explained instance. (Image by Author)

The first explanation already demonstrated that the variable *inactivity_time* is the most important one and that it has a negative effect on the probability of purchase. Moreover, the plots provide additional information on the global importance of the features. On average, it appears that all the features (except *inactivity_time*), have positive effects by pushing the prediction toward the purchase. By looking at the lower parts of the plots, one can notice the uselessness of the features *product_rating* and *product_number_of_ratings*. If this behavior is reproduced across different instances, the explanation suggests that one can simply omit these features from the model without affecting the final result. (We would think these features should have importance — *rather buy a 5-starred item rather than a 0-starred item* — suggesting the data behind may be bad or missing…). A similar conclusion can be drawn for other variables such as *Longitude* or *Delta_time*.

Thus, a developer trying to improve the performance of the model can set a threshold value for the weights of the variables and check if a simpler model based on only the features that exceed the threshold provides similar predictions with fewer variables. An end user might profit from this information by improving the content of the visited pages of the products defined by the variables *product lvl 2–3 name* to increase the *visit duration* and tune the *visit nb pages* and *visit unique pages*.

. . .

Providing an explanation of the results of a model is not anymore an additional feature of a product; it is crucial for ensuring transparency, improving the trust in the product, and meeting the most recent regulations regarding data utilization.

LIME is a rather flexible tool that provides these possibilities. By using LIME, we gained insight into the rationale behind a prediction and drew conclusions that can be fed back into the model to improve it further or employed by the end user to set up a sales strategy.

The flexibility of the explainer has the drawback of the complexity of its first usage. If the predictive model is rather complex (as in our case), intense pre- and postprocessing must be conducted to match the model with the explainer without obtaining misleading explanations. Moreover, the sensitivity to the parameters requires an accurate analysis to ensure that the best possible explainer is applied to the problem; this task is computationally expensive.

The use of recurrent networks in the predictive model makes the direct use of the explainer difficult; more effort is required to obtain a more thorough insight into the result of the explanation.

Overall LIME is a useful tool but that requires careful preparation before its application.

· · ·

In a future post, we will test the capabilities of **SHAP** for the same model to check whether we encounter the same problems as with LIME and whether it provides a better bouquet of explanations suited to a recurrent network.

This work has been done within easyence datalab.