Internship Report

Explainability applied to intent classification models

Explicabilité appliquée à des modèles
de classification d'intention

Academic supervisor
David RYCKELYNCK

Enterprise supervisor
Mathieu SERIS

Author
João Paulo CASAGRANDE BERTOLDO
joao.bertoldo@mines-paristech.fr
joao-paulo.casagrande-bertoldo@dauphine.eu

MINES ParisTech

Université Paris-Dauphine

Université Paris Sciences & Lettres (PSL)

September 2020

Explainability applied to intent classification models

Abstract

The advances and democratization of Natural Language Processing (NLP) over the last few years enabled businesses to leverage new technologies to massively process textual data. Mindsay specialized in providing a customer assistance automation service with chatbots, leveraging NLP classification models to respond to customer queries. With a complex solution containing 1,000 models under the hood, the company's operators frequently face the challenge of understanding and solving issues with them. This internship focused on studying local explainability techniques - analytical tools that express a classifier's decision in simple terms, showing how the inputs impact a specific output - to enlighten the functioning of such models. We focused on using explanations not as an end, but as a means for non-experts to improve classification models. We examined, in Mindsay's context, several explainability methods proposed in the latest years, then proposed a human experiment, where we provide internal users with two different methods, SHAP and Anchors, to assist in a simulated continuous improvement process. We compared the model performance improvements with the users' modifications, and present a qualitative evaluation of the two methods based on their feedback.

Explicabilité appliquée à des modèles de classification d'intention

Résumé

Les progrès et la démocratisation du traitement automatique du langage naturel au cours des dernières années ont permis aux entreprises de tirer parti de ces technologies pour traiter des données de texte massivement. Mindsay s'est spécialisée dans la fourniture d'un service d'assistance client automatisé avec des chatbots, en exploitant les modèles de classification pour répondre aux demandes des clients. Avec une solution complexe contenant 1000 modèles sous le capot, les opérateurs de l'entreprise doivent souvent relever le défi de comprendre et de résoudre ses faiblesses. Ce stage s'est concentré sur l'étude des techniques d'explicabilité locale - des outils analytiques qui expriment la décision d'un classificateur en termes simples, montrant comment les entrées ont un impact sur une sortie spécifique - pour éclairer le fonctionnement de tels modèles. Nous nous sommes concentrés sur l'utilisation des explications non pas comme une fin, mais comme un moyen pour les non-experts d'améliorer ces modèles de classification. Nous avons examiné, dans le contexte de Mindsay, plusieurs méthodes d'explicitation proposées ces dernières années, puis nous avons proposé une expérience humaine, dans laquelle nous fournissons aux utilisateurs internes deux méthodes différentes, SHAP et Anchors, pour les aider à simuler un processus d'amélioration continue. Nous avons comparé les améliorations des performances du modèle avec les modifications apportées par les utilisateurs et nous présentons une évaluation qualitative des deux méthodes basée sur leurs commentaires.

Contents

1 Introduction

1.1 The company

Name	**Mindsay** (former Destygo)
Sector	Information Technology (IT)
Location	Paris, France
Foundation	2016
Type	Privately Held
Number of employees	∼50
Raised capital	9.9 M€
Investors	White Star Capital, Partech, Accor, Paris Aéroport

Table 1 – Mindsay's key figures.

Mindsay, a four-year old Parisian startup, provides a conversational artificial intelligence (AI) solution (also known as *chatbot*[1]) for businesses, mainly in the transport sector. Their platform allows companies to integrate chatbots to their own information systems, enabling them to automate their customer-facing processes.

"Any customer should be able to easily interact with companies, anytime, anywhere"; the company's vision is to make one-to-one customer care accessible to businesses. To achieve that goal, the Mindsay's mission today is to provide *simple* and *efficient* customer interactions; *simple* for the final users, who can easily interact with service providers, and *efficient* for the organizations, who gain productivity by leveraging Natural Language Processing (NLP) capabilities.

How it began In the Spring of 2016, when Facebook announced the introduction of Messenger Bots, Guillaume, Illias, and Pierre realized the potential of a new pathway to solve massive customer-facing challenges seen by organizations. Months later, the first customer landed, and the Mindsay's conversational platform was put to test.

Organization Mindsay is divided in five vertical teams: *HR & Finance*, responsible for employees management, recruitment, office maintenance, and financial analysis; *Business*, the promoters of the company's product, they are looking for new market opportunities; *Product & Design*, a mix of graphic designers and product managers, they are steering the wheel of the product's development, making the bridge Mindsay's customers and developers; and finally the *Delivery* and *Tech* teams.

Delivery team This team has a key role in Mindsay's business model: they structure the company's use cases using the chatbot creator platform (see Figure 1), connecting their business needs to Mindsay's technical solution. They are the ones concretely defining the chatbots' scopes (i.e.: the customer requests treated by the bot), creating the automatic answers, and configuring connections with external resources. As they operate and maintain the customers' chatbots, the members of the Delivery team are the most recurrent users of Mindsay's web application - where the chatbots are created.

[1]We use the words *chatbot* and *bot* interchangeably.

Tech team They bring the product to life. The Tech team develops the chatbot platform, manages the application's databases, and continuously improve the back-end machinery. The team is subdivided into two *feature teams* and a Site Reliability Engineering (SRE) team, who takes care of the infrastructure and security. The feature teams are dedicated to different subjects of the product. The *Resolve & Connect* team develops integrations with other technologies (like Facebook's messenger and customers' APIs) and the content generation mechanisms (like generating personalized QR codes). The *Understand & Analyse* team creates analytical tools to measure the user interactions with the chatbot, and develops the Natural Language Processing (NLP) machinery under the hood - the text processing pipeline, training data set management, model training, and model deployment.

1.2 The market

Mindsay's target market is customer service automation, and the company specialized in the transport sector (travel agencies, airports, airline companies, train operators, subway operators). Customer-facing interaction is a recurrent pain point in this sector; on one side, the consumers see, for instance, travel companies who provide essentially the same service - making the customer relation an important quality factor; on the other hand, these companies need to deal with a massive volume of customer requests, forcing them to balance cost and service quality.

Chatbots provided a way of reducing costs for the enterprises while maintaining the quality of the provided service; they are capable of processing large volumes of simple requests almost as well as human agent. This alleviates the customer care departments, allowing them to deal with complex cases. For the customers, requests are solved faster than they would if they had to wait to talk with a human operator.

It remains hard to define the size of the potential market, but as traditional companies are progressively embracing new technologies, and as artificial intelligence gets better, the opportunities in the next years tend to grow. Today's customers are mostly in Europe, with examples like RATP, SNCF, and AirFrance in France, Iberia in Spain, Kilroy in Denmark, and Thalys in Belgium. A few prospects and initial discussions have been made in the United States and in the Middle East as well.

1.3 The product

Compared to its competitors, Mindsay holds a leading position in terms of complexity of cases that its chatbots can solve, for instance providing integration with reservation systems. Their bots natively processes multiple languages, English, French, Spanish, and German, and tens of others via Google's translation service. The customers can directly plug the chatbots in their websites with a floating widget, or integrate it with multiple channels, like WhatsApp and Facebook Messenger.

Mindsay business model is based on a software as a service (SaaS), a web application (Figure 1), that provides an interface to design chatbots, monitor the their usage, analyze conversations, and manage the data fed the NLP models. Along with the platform, they also sell the service to build, adapt, and improve the bots. As the solution offers a very flexible (thus complex) technology, most clients dispatch the work of modifying the bots to the Delivery team. These operators have, therefore, a central role in how the bots are used; they analyze conversations, detect issues, and correct them.

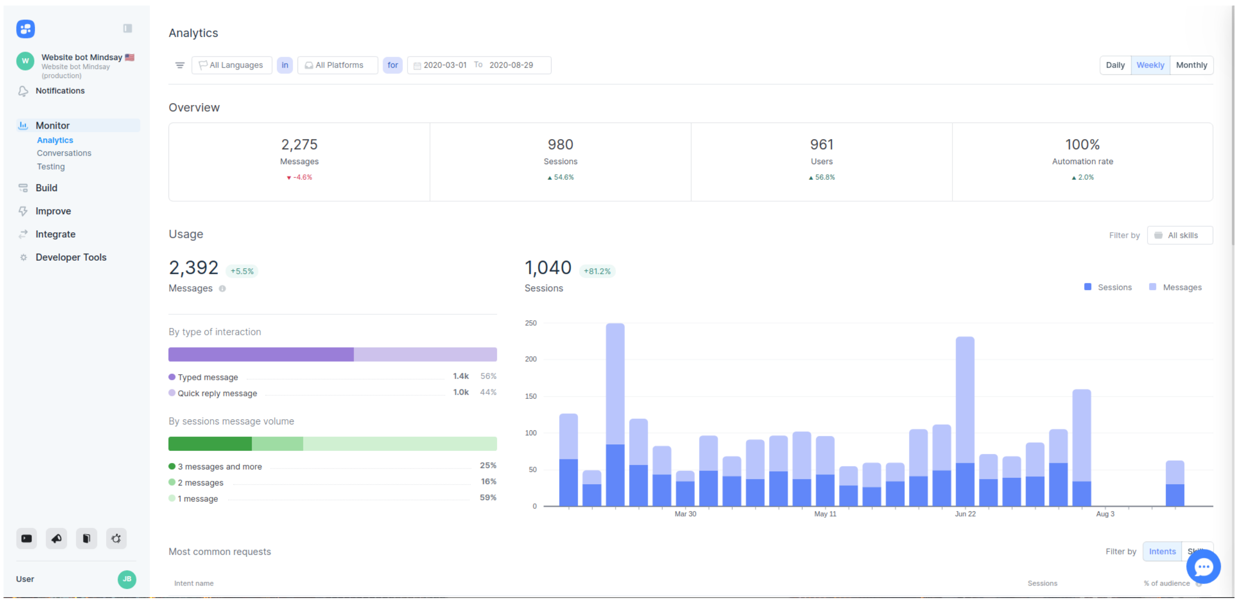


Figure 1 – The chatbot management web application

1.4 The internship

The Delivery team at Mindsay faces a complex solution, that needs careful maintenance, in particular correcting issues with the text comprehension models that power the company's solution. The goal of this internship was to explore how explainability techniques applied to NLP can help at this task. In a first moment, we reviewed a variety of methods in the literature, and analyzed their usability considering Mindsay's context (Section 3). Then, we ran a comparative experiment with human users replicating operational conditions in order to asses how two methods, SHAP and Anchors (respectively, Sections 3.3.1 and 3.3.2), can help the continuous improvement process. Finally, we compare the results showing that local explanations were not effective in this setting, although users more easily understood a text classifier's behavior.

2 Chatbots at Mindsay

In this section we explain the main mechanisms of a chatbot[2], contextualizing the text classification task that we worked with. We describe its basic components, how it processes messages, and some aspects of how the bots are managed. These elements were determinant to lead the choices of explainability methods used in our experiment, as it is detailed in Section 3.

2.1 Nodes and intentions

A bot has a graph structure where each node represents an action from the user or the bot. The actions taken by the bot, the responses sent to the user, are called *bot nodes*, and the actions taken by the user are respectively called *user nodes*. Thus, a conversation can be understood as a graph path, alternating between bot and user nodes, as it is illustrated in Figure 2.

[2]The elements explained in this section are specific to Mindsay. Other architectures and modelizations in the literature may differ significantly.

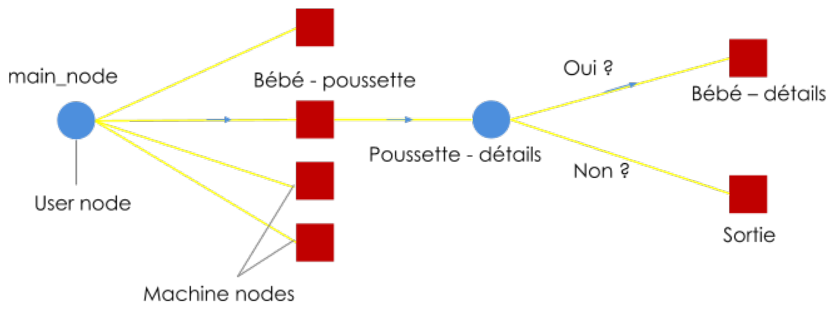


Figure 2 – Node architecture of a chatbot at Mindsay.

When the user sends a message, the bot classifies it according to a list of possible *intents* associated to that user node, which are predefined on Mindsay's platform (this is the bot's *scope*) - these intents correspond to the classes of the statistical model. As it detects an intent, the bot will send a predefined answer accordingly. Then it waits for another message, and repeats the process.

The first user node in the graph structure, the *main node*, has a central role: it is the beginning of all the use cases that a bot can execute (e.g.: provide a flight status information, or change a reservation). It also has a special behavior: every message gets a prediction from the main node, even if the conversation is in another user node, allowing users to quit a process and start another one at any time. This feature depends on a special intent called *Misunderstanding*, or *OUT* (for "out of scope"). This class always exists in any user node, in any bot, and represents everything that a bot does not know, cannot do, or is not capable of classifying - for instance, when it receives gibberish messages.

We choose to only work with main user nodes because of the its special role, and because it is usually the node with the largest number of intents (classes), making it more challenging to understand these models. From now on, I will use the terms *main node*, *model*, and *classifier* as synonyms, referring to the supervised text classification model inside a chatbot's main (user) node.

2.2 Message processing

Before a message reaches the model, it passes by two important steps. First the message is lower cased and, depending on the language, accents are removed. Second, entity tokens are added to the message. The latter considerably influences the model's inputs, so we will briefly explain what it consists of.

Entities The platform allows users to define types of entities that can be used in the bot nodes, like dates, city names, personal names, boarding card numbers, etc. Besides becoming a variable, the entities are associated to unique tokens (the words in the model's vocabulary), that are appended to the message. For instance, the message "*I want to go from Paris to Marseille tomorrow*" contains entities of city and date, then the pre-processed sentence would be "*i want to go from paris to marseille tomorrow __entity_city __entity_date*".

Model types Currently, there are two families (types) of models. The first, older, uses the framework *fastText*, and the second uses *TensorFlow*. For several reasons, the migration from the former to the latter was not completed, and a third one might eventually replace both. For this reason, it is valuable to prefer model-agnostic explainability methods - that is, methods capable of dealing with black-box models, without depending on any specific architecture.

2.3 Managing a bot

Mindsay's back-end system manages around 1,000 classification models under the hood. Analyzing and inspecting all of them is a complex task, given that the operators in the Delivery team have limited time to do this (in addition to build their scopes and create response content). As a consequence, the amount of training data available is limited because the operators (and eventually the customers) have to manually create sample sentences for every intent.

The number of samples per intent is usually imbalanced, and the quality of the distribution deteriorates as the bots are redesigned over time. The number of samples of some intents is very limited (a few tens or less) because the human cost of maintaining these data sets is high, while the *OUT* intent usually contains a lot more (several hundreds).

Splitting the data sets for model evaluation would be a source of inefficiency for the company because discarding them in the training phase would be a "wasted" of human (costly) effort. As a result, the production environment lacks a proper validation of the classification models. Mindsay has, however, developed two mechanisms to mitigate this issue: first, the customers can mark conversations with issues, so that the operators will try to solve them; and second, the platform has a data annotation tool that allows operators to manually tag messages from users.

In these circumstances, looking under the hood of a classifier and making sure it works well is a constant pain for the Delivery team. Besides the bots' parameters, it remains a challenge to understand why a model cannot understand a specific phrase, especially considering that most operators have limited knowledge about NLP. Furthermore, it is hard to know where to look up for issues; the myriad of messages make it hard to find relevant ones [3].

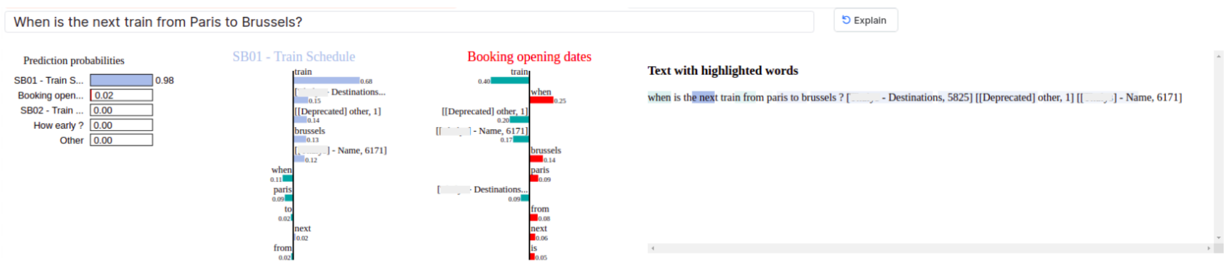
In this context, Mindsay started investigating the potential of explainability methods to help diagnosing misclassifications, improving their classifiers, and, more generally, understand the models' behavior.

3 Explainability

3.1 What is it?

There has been great improvements in NLP in the last few years, and the complexity of the models have quickly grown [4]. Models using, for instance, the Transformers architecture [15] are achieving a new level of natural language processing performance, but it also becomes harder and harder to fully understand a model's decision process as it grows in complexity. In this context, the term *Explainable AI* (or XAI) emerged to name a sub-domain of research dedicated to mitigate this issue. Sometimes the term is *interpretability* is also used with a similar meaning, so for the sake of clarity, let's briefly discuss the difference between these two terms.

[3] A large portion of the messages are only keywords or simple "hello"s


When is the next train from Paris to Brussels?

Text with highlighted words

when is the next train from paris to brussels ? [[redacted] - Destinations, 5825] [[Deprecated] other, 1] [[[redacted]] - Name, 6171]

Figure 3 – A screenshot from Mindsay's application on the Explainer page. An example of explanation on a requested sentence. The probabilities on the left correspond to the model's intents. The plots in the middle correspond to the importances given to each word for two most likely intents.

Interpretability As proposed in [1], *interpretability* is the capacity to understand the meaning of something in human-understandable terms. For instance, a decision tree model's decision is interpretable because one can understand each of the steps on the graph, then recompose the final decision by following them. On the other hand, a multilayer perceptron's decision is not interpretable because the learned features are combinations of the previous layers, so a human cannot keep track of multiple transformations of real-valued vectors.

Explainability Even if a model is not interpretable, it can be explainable. The term *explainability* refers to the concept of making a bridge between a decision maker (say, an LSTM model) and a human (say, an operator at Mindsay) [1]. Explaining a model, in this example, means being capable of representing the LSTM's decision accurately with simple terms can be understood by the an operator, who supposedly does not even know about the model's architecture. These simple terms could be a linear model under certain conditions - as we will see in Section 3.3, despite the linear model being *weaker* than an LSTM, we can restrain the input space to make the classification "easier".

3.2 Explainability at Mindsay

Mindsay previously adopted an explainability tool in their bot management application; the so called *Explainer* (Figure 3) generates explanations for a query sentence using the popular method LIME (Local Interpretable Model-agnostic Explanations). The operators in the Delivery team are the main users of this tool, which allows them to better understand why a model makes certain decisions.

The goal of this internship was to explore other techniques proposed in the literature in order to improve and complement the existing tool. With this in mind, some aspects of Mindsay's context guided our approach choices; the next paragraphs discuss these considerations.

Utility The ability of explaining a model's behavior is a mean, not a finality. Some authors focus on explainability, for instance, to prevent AI from deepening gender, racial, or sexual discrimination originated in human behavior [3]. Other authors, like Ribeiro et al. [9], explore how explainability can be used for model selection, which can be useful in cases where the amount of data is limited. In the case of Mindsay, the main users of such a tool - the operators of the Delivery team - ultimately wish to improve a chatbot. Therefore, we focused on the usability of explainability methods for the search of issues and bad generalizations.

Available data As it was mentioned in Section 2.3, there is no validation data available in Mindsay's production data sets, therefore making it impractical to measure a bot's performance properly. Besides, the messages received from the final users can have a distribution considerably different from those in the training set for two reasons. First, the messages are generated differently; while the training messages are handcrafted by the operators and third-party collaborators, the production messages are from real users who ignore the intent definitions. Second, the user's intentions are naturally imbalanced because some use cases are more relevant/demanded than others.

For these reasons, we gave priority to methods that enable a proactively processing the users' messages (e.g.: explanations that can be generated in advance, eliminating the need to query them) because, due to the classification task's complexity, it is hard to even know where it is relevant to query. Thus we focused on *local post-hoc* explanations - so let's clarify what those terms mean.

Post-hoc The term *post-hoc* refers to the explanation being generated after a model is already trained and considering it as an immutable black-box. These methods usually query the model many times to detect behavior changes. By contrast, methods that are not post-hoc make the model itself more transparent, for instance summarizing its behavior with a rule list, such as proposed in [8]. Others have proposed hybrid architectures that make the model partially more interpretable, such as [7], who generates decision tree with a neural network in each node.

Local A local explanation (as opposed to a global one) only makes sense in a sub-region of the input space. In the case of text classification, this means that, instead of considering all the possible texts, an explanation only considers variations of a certain sentence, which can be generated by exchanging words with similar meaning or masking them. A local explanation takes a sentence as a reference of a point in the input space so that the classification task becomes simpler, making it possible to use a simpler proxy model (like a logistic regressor or a decision tree) in that region while maintaining accuracy.

3.3 Literature review

While reviewing recent literature on explainability, we considered several methods that could be useful for Mindsay. In this section we present, first, a brief introduction to linear explanations, then the two explanation techniques selected for our study (Section 4), SHAP [6] and Anchors [10].

In the Appendix A I shortly comment several papers reviewed during our research. The Appendix B summarizes other interesting alternatives that were discarded and why they were not retained. Finally, after reading the definitions of SHAP and Anchors (respectively Sections 3.3.1 and 3.3.2), the reader is encouraged to read the Appendix C, where we discuss practical matters with the usage of these methods that we encountered.

Linear explanations (LIME and SHAP) Given an input sentence, these methods assign a real value, called *importance*, to each token in the sentence (for each class of a multi-class model). Each probability predicted by the classifier is, in a way, decomposed by these values. In other words, they tell the user how much each token is important towards that probability. If you

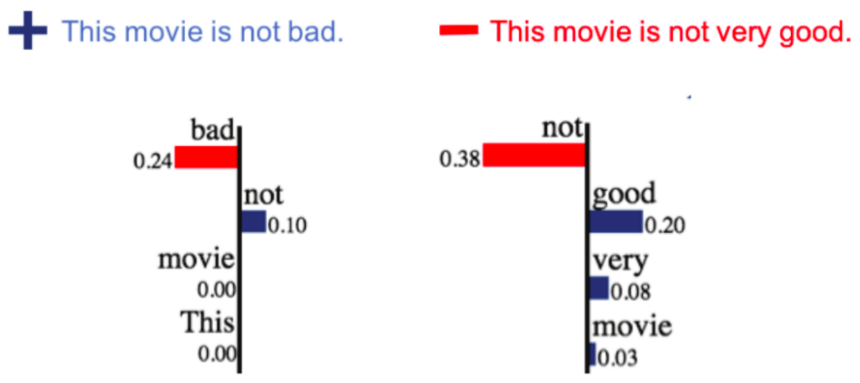


Figure 4 – Examples of linear explanations using LIME with a sentiment analysis model used with movie reviews. In these examples the problem is binary, and the positive class means *positive sentiment*.

remove a word and the probability decreases, it means that the token is important for that class; contrarily, if the probability increases, then the token has a negative importance.

Take, for example, the explanations in the Figure 4, generated with LIME for a sentiment analysis model. We can clearly see the effect of locality on the explanations; on the left, the importance of the word *not* is positive, meaning that its presence makes the model "think" that the whole sentence is positive (in other words, without the word "not", the sentence would become more negative); on the right, the same word has a negative importance (i.e.: towards the opposite direction of the positive class).

Both LIME and SHAP are designed to associate importance values that could simulate the model's behavior linearly with variations of the original sentence. The difference between the two methods is the strategy adopted to find those importance values. They both solve a similar problem considering the presence or absence of the tokens, but the formulations (and, therefore, the optimization problems under the hood) are essentially different.

In the next section we present SHAP's formulation; LIME's is explained in Section B.1, and the reader is strongly encouraged to first read it, as some parts are reused.

3.3.1 SHAP (SHapley Additive exPlanations)

Similar to LIME, SHAP (SHapley Additive exPlanations), proposed by [6], is a model-agnostic framework based on the presence/absence of features in an example to generate local explanations. However, while the former defines heuristic penalization and weighting schemes, SHAP imposes certain characteristics to the explainer model $g \in G$ and demonstrate that there is a unique solution.

We use auxiliary inputs from a simplified space $z' \in \{0,1\}^d$ to represent variations $z \in X$ of the original sentence $x \in X$, where each entry z'_j in z' represents the presence/absence of a token z_j in z, and d is the number of tokens in the original sentence x. We denote $h_x : X \to \{0,1\}^d$ the transformation from simplified inputs to variations of the sentence x. We then define an additive proxy model $g \in G$ for a probabilistic model $f : X \to [0,1]$ as:

$$g\left(z'\right) = \phi_0 + \sum_{j=1}^{d} \phi_j\, z'_j \quad \text{with} \;\; \phi_j \in \mathbb{R} \tag{1}$$

Note that ϕ_j depends on the model f and the input x, so we will sometimes denote $\phi_j = \phi_j(f,x)$. Let's now define three desirable properties for this model: *local accuracy*, *missingness*, and *consistency*.

Property 1 (local accuracy) The proxy's prediction for the original sentence must match the explained model's probability:

$$g(x') = f(h_x(x')) \tag{2}$$

where $x' = [1]_{j=1}^{d}$, therefore $h_x(x') = x$.

Property 2 (missingness) A token absent in the original input must have a null importance:

$$x'_i = 0 \Longrightarrow \phi_i = 0 \tag{3}$$

Property 3 (consistency) If a simplified input z'_j contributes more to a model f' than to another f, then the the former's importance $\phi_j(f',x)$ must be greater than the latter's, $\phi_j(f,x)$. Let $z' \backslash j$ denote setting $z'_j = 0$, and $f_x\left(z'\right) = f\left(h_x\left(z'\right)\right)$. For any two models f and f':

$$f'_x\left(z'\right) - f'_x\left(z' \backslash j\right) \;\geq\; f_x\left(z'\right) - f_x\left(z' \backslash j\right) \;\Longrightarrow\; \phi_j\left(f',x\right) \;\geq\; \phi_j(f,x) \tag{4}$$

Solution Lundberg, Allen, and Lee [6] demonstrate that there is an unique additive model $g \in G$ that respects the three properties above is, and its weights can be analytically computed with

$$\phi_j(f,x) = \sum_{z' \in \{0,1\}^d} \frac{|z'|!\,(d - |z'| - 1)!}{d!} \left[f_x\left(z'\right) - f_x\left(z' \backslash j\right)\right] \tag{5}$$

where $|z'|$ is the number of non-zero elements in the vector z'. The values ϕ_j of this solution are called *Shapley values*, a result from combined cooperative game theory found by Shapley [13].

Approximation Notice that the sum in the analytical solution above iterates over all the simplified space $\{0,1\}^d$, thus making 2×2^d calls to the model f, for each token. As the sentence length increases, the number of evaluations necessary make it prohibitive to compute the Shapley values. Lundberg et al. [6] propose an approximation, called *Kernel SHAP*, using the formulation created by Ribeiro et al. [9] (see Section B.1) by setting LIME's framework's components as follows:

$$
\begin{aligned}
\Omega(g) &= 0 \\
\pi_x\left(z'\right) &= \frac{(d-1)}{(d\,\text{choose}\,|z'|)\,|z'|\,(d-|z'|)} \\
\mathcal{L}\left(f,g,\pi_x\right) &= \sum_{z'\in Z}\left[f\left(h_x\left(z'\right)\right)-g\left(z'\right)\right]^2\pi_x\left(z'\right)
\end{aligned}
\tag{6}
$$

Why SHAP? We picked SHAP explanations to be a candidate for our experiment because it offers a model-agnostic framework, therefore being adaptable to any existing or eventually new model at Mindsay. As mentioned by Lundberg et al. [6], *"[As LIME uses heuristic functions (Ω and π_x),] One consequence is that local accuracy and/or consistency are violated, which in turn leads to unintuitive behavior in certain circumstances ..."*. For this reason we preferred to use SHAP instead of LIME. Besides, linear explanations are relatively easy to understand, as we confirmed with our experiment.

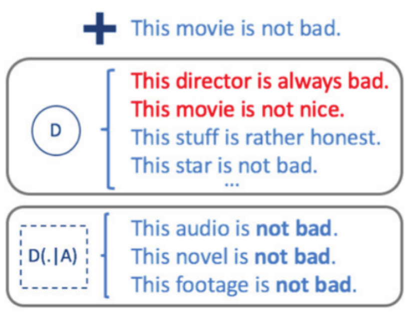
3.3.2 Anchors

The second method we selected provides a completely different type of explanation. Anchors [10] finds a subset of tokens that *guarantees* (with a certain precision) that the model's prediction does not change.

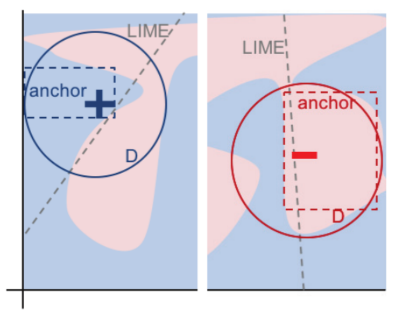
Before formally defining an *anchor*, let's consider an example with the sentence "I would like to buy a ticket to Paris". Suppose the model predicts the intention "Buy ticket". While an additive explanation would probably give a high importance to words like "buy" and "ticket", an anchor tells you that "buy a ticket" *anchors* (affixes) this prediction. In other words, if you mask (hide) any set of the other words in the sentence, the prediction will still be "Buy ticket". In this mental example, we (of course) supposed a model with a reasonable behavior, but they can as well reveal weird behaviors like the example in Figure 8b, where the model seems to give too much attention the token "les".

Using (almost) the same notation from the Section 3.3.1, let's consider a sentence $x \in X$ and a model $f : X \to \{1,\ldots,K\}$, where K is the number of intents (classes)[4]. Let $\mathcal{D}_x$ denote the

[4]Notice that, in the previous section, the model's output was a probability; here, the image is the set of classes.



(a) $\mathcal{D}$ and $\mathcal{D}(.|A)$



(b) Two toy visualizations

Figure 5 – Anchors distributions: on the right, examples of a perturbation distribution $\mathcal{D}$ and a conditional perturbation distribution $\mathcal{D}(\cdot \mid A)$ with the anchor $A = \{\text{"not"}, \text{"bad"}\}$.

perturbation distribution of the sentence x - sentences with modifications based on the reference x. We will use the rule set $A : X \to \{0,1\}$ (a set of predicates) to represent an explanation such that $A(z) = 1$ if all the tokens in A are present in the sentence z. In the previous example, the anchor would be $A = \{\text{buy}, \text{a}, \text{ticket}\}$, and a perturbed sentence could be $z =$ "I want to purchase a train ticket to Paris" (remember that $z \sim \mathcal{D}_x$). Finally, let $\mathcal{D}_x(\cdot \mid A)$ denote the perturbation distribution conditioned to $A(z) = 1$ (see Figure 5).

We say that the predicate A is an anchor if

$$\text{precision}(A) = \mathbb{E}_{\mathcal{D}_x(z|A)}\left[\mathbb{1}_{f(x)=f(z)}\right] \geq \tau \tag{7}$$

where τ is a (fixed and pre-defined by the user) precision threshold. Notice, however, that this definition leaves us with the computation of a conditional expectation, which we can only estimate. To efficiently compute an anchor, we use, instead, a probabilistic version of this definition:

$$P(\text{precision}(A) \geq \tau) \geq 1 - \delta \tag{8}$$

where δ is a user-defined confidence bound. This formulation gives an easier task as long as we can evaluate the probability associated to the average precision of a set of samples from $\mathcal{D}_x(z \mid A)$. Yet, we would not know how to pick a candidate anchor if two or more predicates respect the condition 8. So we introduce the concept of *coverage*, which represents how much of the perturbed distribution is covered by a predicate, and formulate the anchor search as a constrained maximization problem:

$$\begin{aligned} A^* &= \underset{A \text{ s.t. } P(\text{precision}(A) \geq \tau) \geq 1-\delta}{\text{argmax}} \text{coverage}(A) \\ \text{coverage}(A) &= \mathbb{E}_{\mathcal{D}_x(z)}[A(z)] \end{aligned} \tag{9}$$

Again, this reformulation leaves us with an expectation and, in addition, with an intractable optimization problem because the number of candidates grows exponentially with the number of tokens in the sentence x. To solve this, we make the assumption that shorter anchors have higher coverage, since they put less constraints on the perturbations. Therefore, we compute anchors using a bottom-up strategy, starting with an empty predicate and adding up tokens one by one. At each step, we pick the anchor candidate with the highest precision and increment it with another token from the sentence - in reality, the author's implementation[5]uses a beam search algorithm with B candidates.

As for the precision computation, we use a multi-armed bandit (MAB) strategy at each step because otherwise the necessary number of samples (and model predictions) would be prohibitive. Using the *explore-m* setting, in a given step of the beam search, the candidate predicates are the arms, the latent rewards are their precisions, and each pull of the arm is an evaluation of $\mathbb{1}_{f(x)=f(z)}$. The MAB problem returns a set $\mathcal{A}$ of size B, which is an approximation of the optimal set $\mathcal{A}^*$. We use the KL-LUCB algorithm [5], which guarantees with a tolerance $\varepsilon \in [0,1]$ that

[5]Link to the author's implementation repository.

$$P\left(\min_{A \in \mathcal{A}} \operatorname{precision}(A) \geq \min_{A' \in \mathcal{A}^*} \operatorname{precision}\left(A'\right) - \varepsilon\right) \geq 1 - \delta \tag{10}$$

Figure 15 illustrates this result with $B = 1$. Notice that, by definition, Anchors explanations provide a clear definition of its perimeter (i.e.: where it is valid), an aspect that other methods often neglect.

4 Experiment

In order to evaluate the selected methods (SHAP and Anchors) in Mindsay's setup, we elaborated an experiment with potential users in the company to evaluate how these methods would affect their way of tweaking the classification models. The goal was to test if an explanation method can help Mindsay's operator understand why a model makes mistakes, and make it easier to improve a classifier. Comparing explainability methods is less straightforward than evaluating models because there is no ground truth to compare with. As we focus on using explanations to debug and improve models, our test uses the model performance as a proxy to measure how well the users are capable of correcting the issues found.

Figure 6 illustrates the current continuous improvement process at Mindsay. The explanations (yellow), currently generated on demand with LIME, influence how an operator, whose goal is to identify issues (red), sees the model; then the operator creates new samples sentences to retrain the model (purple)[6].

We essentially replicate this process fixing the operators' handles to modify the model (adding samples and blacklisting tokens), and compare the effects of their actions on retrained models. In other words, the experiment is designed to measure if using SHAP or Anchors is more effective then simply not using explanations. Unlike today's process, however, we proactively generate the explanations in advance, and select them with a strategy to optimize their global comprehension of the model. Our experiment setup is encouraged by positive results reported by Ribeiro et al. [9],

[6]The dashed arrow represents samples that were created by the operators in the past, supposedly when the bot was first created.

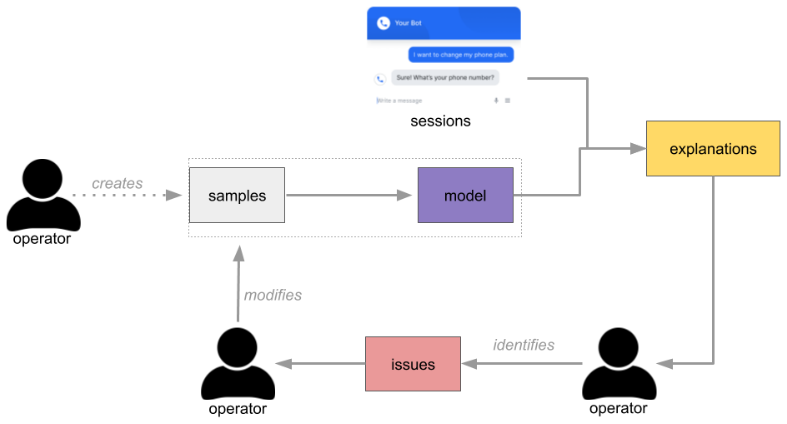


Figure 6 – Experiment diagram.

where the authors say: "The insights given by explanations are particularly helpful in identifying what must be done to convert an untrustworthy model into a trustworthy one – for example, removing leaked data or changing the training data to avoid data set shift."

In order to reproduce a realistic scenario, we considered the following conditions:

1. There is no validation data set available[7].
2. The models used in production often overfit the training data set[8].
3. The classifiers have tens or hundreds of classes.
4. The main leverage to correct a model is by feeding it with more samples and retraining.
5. The messages from the final users (i.e.: from chatbots' conversations) are not labeled.

Besides these conditions, there is an extra hypothesis that we considered: improving a bot by detecting issues is more effective using bad classification decisions than correct ones. Considering the condition 2 above, it would be hard to pick bad decisions from the training set, as there are very few mistakes. Thus, we used production messages, which, however, are unlabeled - making it impossible to search for misclassifications. Therefore we used the entropy of the output probabilities as a metric of "uncertainty", expecting to spot misclassifications (Section 4.1).

Experimental procedure We ran the experiment on the main user node [9] of a chatbot in French accessed by 4,000 users every month in average. This node contains 106 intents, and we used a train/test split of 50/50% from the production model's training set (9,514 sample sentences). The steps of the experiment were the following:

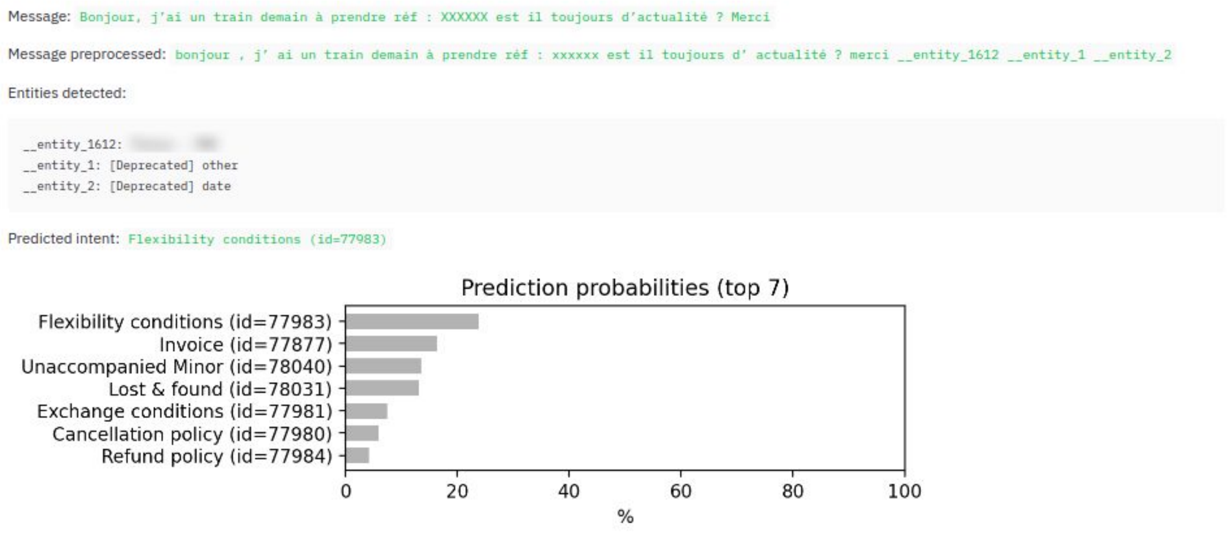
1. Split the sample sentences into train/test sets, and train a model.
2. Select a sample of production messages
3. Generate explanations with SHAP and Anchors.
4. Show explanations to a human user.
5. Ask the user to create new samples to be added to the training data set, and potentially blacklist tokens.
6. Retrain the model with the modifications recommended by the users.
7. Compare the performances of the retrained models.

User sessions The experiment sessions (steps 4 and 5) were limited to roughly 1 hour of work. Each user only used one type of explanation at a time: SHAP, Anchors, or NoExp - the control group, without any explanation.

[7]As it is mentioned in the Section 3.2, there is no train/test split for production models because it is costly to maintain a data set big enough. We made a copy of a production model using reduced training set in order to have a test split. However, the explainability method itself does not make use of it. It would not be the case, for instance, if we used L2X (see the Section B.3).

[8]Classification F1-scores are commonly between 98% and 100%

[9]See Section Nodes and intentions.



Message: Bonjour, j'ai un train demain à prendre réf : XXXXXX est il toujours d'actualité ? Merci

Message preprocessed: bonjour , j' ai un train demain à prendre réf : xxxxxx est il toujours d' actualité ? merci __entity_1612 __entity_1 __entity_2

Entities detected:

```
__entity_1612: [redacted]
__entity_1: [Deprecated] other
__entity_2: [Deprecated] date
```

Predicted intent: Flexibility conditions (id=77983)

Prediction probabilities (top 7)
Flexibility conditions (id=77983)
Invoice (id=77877)
Unaccompanied Minor (id=78040)
Lost & found (id=78031)
Exchange conditions (id=77981)
Cancellation policy (id=77980)
Refund policy (id=77984)
0
20
40
60
80
100
%

Figure 7 – A screen capture from a user session using no explanations, only the basic information of the text itself and its prediction.

Displayed information This was the information available to the users according to the method used:

NoExp The original message, the preprocessed message with entity tokens (see Section 2.2), a dictionary of the entity tokens, and the predicted probabilities. Figure 7 shows an example.

SHAP All the information from NoExp, a textual and a graphical visualizations of the SHAP values of the predicted intent (see an example on Figure 8). Note that each intent (class) has different SHAP values, and the users only see the values for the predicted intent by default. In addition to the default intent, they could also request the values for any intent.

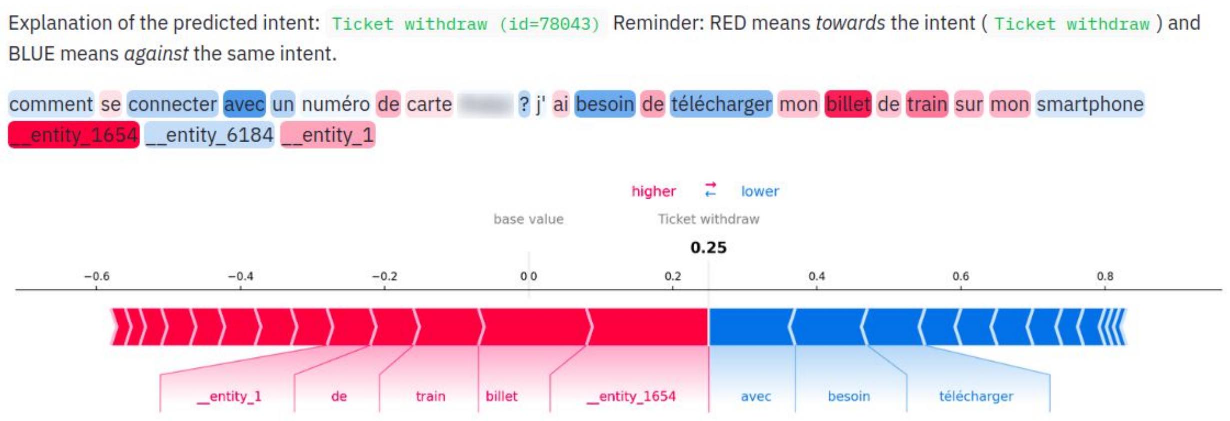
Anchors All the information from NoExp, the text with the anchor highlighted, the anchor tokens, and its associated precision.

4.1 Production messages sample

The user requests from the production environment naturally have an imbalanced distribution of intents, and many of them are simply not treated by the chatbot. Thus, generating explanations from a sample would give the users very little variation of intents, which we wish to avoid. As one cannot know in advance what are the ground truth intents of the production messages, we adopted a strategy based on the model's predictions.

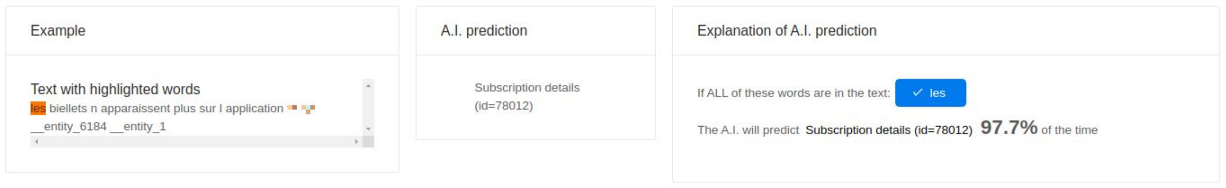
Consider a sentence $x \in X$ and a model $f : X \to (0,1)^K$, where $K = 106$ is the number of classes. We denote $\widetilde{y} = f(x)$ the model's output probabilities, $\widetilde{y}_k \in (0,1)$ the probability assigned to the class k, and $\hat{y} \in [K] = \{1,...,K\}$ the predicted class - naturally, $\hat{y}$ is the class with highest probability: $\hat{y} = \hat{f}(x) = \text{argmax}_{k \in [K]} \widetilde{y}_k$. Finally, let $h = H(\widetilde{y}) = -\sum_{k \in [K]} \widetilde{y}_k \log \widetilde{y}_k$ be the entropy of a prediction.

We get a set of sentences $S_{prod} = \{x^{(1)}, \ldots, x^{(i)}, \ldots, x^{(N)}\}$ (with $N = 49.000$ approximately), and split it into K sets by predicted intent $S_k = \{x^{(i)} \in S \mid \hat{y}^{(i)} = k\}$. Then, from each set S_k, we select the top $n = 11$ highest entropy samples, giving S_k^*. Let's recall that a prediction with probability


Explanation of the predicted intent: `Ticket withdraw (id=78043)` Reminder: RED means *towards* the intent (`Ticket withdraw`) and BLUE means *against* the same intent.

comment se connecter avec un numéro de carte [redacted] ? j' ai besoin de télécharger mon billet de train sur mon smartphone __entity_1654 __entity_6184 __entity_1

(a) SHAP explanation.


Example	A.I. prediction	Explanation of A.I. prediction
Text with highlighted words les biellets n apparaissent plus sur l application __entity_6184 __entity_1	Subscription details (id=78012)	If ALL of these words are in the text: ✓ les The A.I. will predict Subscription details (id=78012) 97.7% of the time

(b) Anchor explanation.

Figure 8 – Explanation visualizations.

of 1 for a single intent has entropy 0, and, contrarily, one with little probability over many intents will have a high entropy; thus we use the entropy as a measure of *certainty* of the model as a way to look up for "hard" decisions. Finally, we join them into the set $S = \bigcup_{k=1}^{K} S_k^*$, with $|S| = n \times K = 1166$ sentences.

Each message gets an explanation from each of the 2 methods (SHAP and Anchors), ending the step 2 of the procedure described above.

4.2 Selecting explanations (pick step)

As the users only had 1 hour, they clearly lacked time to look at more than 1000 explanations, so we used a strategy to pick which explanation should be presented - the so called *pick step*. We designed a strategy inspired by the "Submodular Pick (SP)" proposed by Ribeiro et al. [9], where the authors shows that this strategy is significantly more efficient than randomly picking examples in a similar task.

4.2.1 Original Submodular Pick

Let's recall what the original SP strategy does, then present our adapted version[10]. Let $\mathbf{V}$ be the vocabulary, a list of all the tokens found in S, and $\Phi \in \mathbb{R}^{|S| \times |\mathbf{V}|}$ a matrix with local importances of the $|S|$ sentences, each relative to their predicted classes (maximum probability).

The sentence $x^{(i)}$ gets an explanation $\phi^{(i)} \in \mathbb{R}^{K \times |x|}$, where $\phi_{k,j'}^{(i)} \in \mathbb{R}$ is the importance of the token j (from the vocabulary) for the class k[11]. Then, its respective row Φ_i in the importances matrix is

[10]We use a modified formulation of the original algorithm so that the notation is more natural for our own.

such that

$$\Phi_{i,j} = |\phi^{(i)}_{\hat{y},j'}| \times \mathbb{1}_{[\mathbf{V}_j \in x^{(i)}]}$$

Finally, let's define a token's global importance as $I_j = \sqrt{\sum_{i=1}^{|S|} \Phi_{i,j}}$.

The submodular pick algorithm then greedily selects explanations one by one such that the words seen are as varied as possible, giving priority to the tokens with high global importance. Let S_{seen} be the set of sentences already selected by the algorithm (at the first iteration $S_{seen} = \emptyset$), and $\mathbf{s} \in \mathbb{N}^{|V|}$ the vector of token usage counter, where $s_j = \sum_{x \in S_{seen}} \mathbb{1}_{[j \in x]}$ is the number of times the token $\mathbf{V}_j$ is used in the sentences from S_{seen}. Then, the algorithm picks the unseen sentence with the highest sum of token importances only considering unseen tokens ($s_j = 0$):

$$x^* = \underset{x \in S \backslash S_{seen}}{\text{argmax}} \; \alpha_{\mathbf{s}}(x) = \underset{x \in S \backslash S_{seen}}{\text{argmax}} \sum_{j \in x} I_j \times \mathbb{1}_{[s_j = 0]} \tag{11}$$

We can interpret the quantity inside the argmax operator as the added importance of a new sentence. We aim to get the highest sum of global importances with the tokens seen (i.e.: the tokens used in the sentences seen), but minimize the number of tokens seen in order to demand less effort from the user by showing less text. From this point of view, the algorithm grows the set S_{seen} by maximizing the global importance added by each candidate sentence (the equation above).

4.2.2 Our pick algorithm (with SHAP)

For several reasons, the algorithm presented above in Section 4.2.1 was unfit for our use case, so we adapted it to mitigate some of the issues. We gave the users the option to refuse proposed examples, introduced the an entropy term to prioritize "hard" classifications, and generalized the formulation to avoid ignoring recurrent keywords.

Class variety and sentence size The original submodular pick ignores the variety of classes in multi-class settings, which is also valuable to understand a multi-class model. Besides, as it uses a sum of importances over the tokens of a sentence, it tends to pick long sentences. In a context with hundreds classes like ours, this is problematic because a single example contains a considerable amount of information, so short messages are preferable.

We have tried a variations of this formulation that take these factors into account, but the simplest and most efficient way of mitigating these issues was to give the users the choice of using or not a proposed sentence. This way, they could avoid long sentences and repeated subjects. Before showing an explanation, our experimetn application prompted a confirmation (see Figure 16); if the user says "yes", then the sentence is added to S_{seen}, otherwise it is added to the set of rejected sentences $S_{rejected}$. The Equation 11 then becomes:

[11] Notice that there are two different word indexes, j and j', because they represent the indexes of different sequences. The first, j, represents the position of a token in the vocabulary; the second, j', represents the position of that token in the sentence being considered.

$$x^* = \underset{x \in S \backslash S_{seen} \backslash S_{rejected}}{\operatorname{argmax}} \quad \alpha_{\mathbf{s}}(x) \tag{12}$$

Entropy Even though the set of selected production messages S are already the entropy highest entropy examples per class, we found it useful to integrate it to the pick step in order to make sure that the users were presented "hard" classification examples. Giving the following formulation:

$$x^* = \underset{x \in S \backslash S_{seen} \backslash S_{rejected}}{\operatorname{argmax}} \quad H(f(x)) \times \alpha_{\mathbf{s}}(x) \tag{13}$$

Binary added importances Notice that, in Equation 11, a new candidate sentence's token only adds up to the added global importance if it has never been seen. However, Mindsay's bots often have several intents (classes) with similar definitions. For instance, "Ticket purchase" and "Ticket exchange" have a common factor - even in the name: they both concern subjects about tickets. Thus, the sentences from these two classes naturally share words like "ticket", "reservation", "pay", etc.

This all-or-none strategy (the condition $s_j = 0$) ignores phrases from similar intents because if, for instance, an example from "Ticket purchase" has been seen, the ones in "Ticket exchange" will not add much importance. Curiously, this mechanism also caused the effect of forcing intents; for instance, after seeing a few examples in French, the algorithm starts to return sentences only in English[12].

In order to mitigate these issues, we changed the $\alpha_{\mathbf{s}}$ function to:

$$\alpha_{\mathbf{s}}(x) = \sum_{j \in x} \frac{I_j}{\sqrt{1 + s_j}} \tag{14}$$

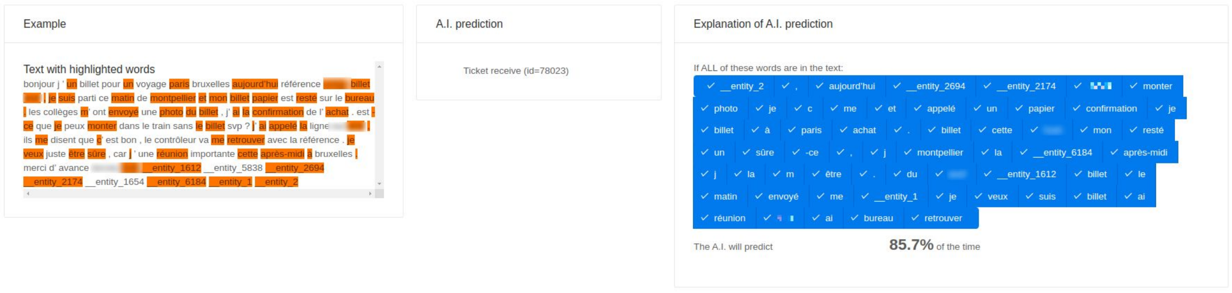
4.2.3 Anchors pick algorithm

Although the algorithm above worked well, the Anchor explanations required yet another change. As the users quickly realised, Anchors with long messages are barely useful at all (see an example in Figure 9). When presented with too much information, it is hard interpret an explanation, and, after all, we figured out that Anchors are rather useful when it reveals a "strong bias", when only a few words in the sentence are enough to make a decision.

Anchors was developed to work with a generic classifier, and it can deal with any type of input (categorical, numerical, textual, etc). In the case of text inputs, each token is seen as an individual feature, thus, when masking the tokens (see Section 3.3.2), it completely ignores the sequential nature of the text. As a result, we often get anchors with disconnected words in the explanation - especially when they are long.

Therefore we adopted a different approach to select anchor explanations, based on the length of the explanation. Assuming that long sentences with short anchors are more useful, we changed

[12]Although the model under test had been trained in French, the final users sometimes try to talk with the bot in other languages. Usually Mindsay's bots have a few intents to detect other languages and redirect the user to the right bot.


Example

Text with highlighted words

bonjour j ' un billet pour un voyage paris bruxelles aujourd'hui référence [redacted] billet [redacted] , je suis parti ce matin de montpellier et mon billet papier est resté sur le bureau . les collèges m' ont envoyé une photo du billet , j' ai la confirmation de l' achat . est - ce que je peux monter dans le train sans le billet svp ? j' ai appelé la ligne [redacted] , ils me disent que c' est bon , le contrôleur va me retrouver avec la référence . je veux juste être sûre , car j ' une réunion importante cette après-midi à bruxelles . merci d' avance [redacted] __entity_1612 __entity_5838 __entity_2694 __entity_2174 __entity_1654 __entity_6184 __entity_1 __entity_2

A.I. prediction

Ticket receive (id=78023)

Explanation of A.I. prediction

If ALL of these words are in the text:

✓ __entity_2 ✓ , ✓ aujourd'hui ✓ __entity_2694 ✓ __entity_2174 ✓ [redacted] ✓ monter ✓ photo ✓ je ✓ c ✓ me ✓ et ✓ appelé ✓ un ✓ papier ✓ confirmation ✓ je ✓ billet ✓ à ✓ paris ✓ achat ✓ . ✓ billet ✓ cette ✓ [redacted] ✓ mon ✓ resté ✓ un ✓ sûre ✓ -ce ✓ , ✓ j ✓ montpellier ✓ la ✓ __entity_6184 ✓ après-midi ✓ j ✓ la ✓ m ✓ être ✓ . ✓ du ✓ [redacted] ✓ __entity_1612 ✓ billet ✓ le ✓ matin ✓ envoyé ✓ me ✓ __entity_1 ✓ je ✓ veux ✓ suis ✓ billet ✓ ai ✓ réunion ✓ [redacted] ✓ ai ✓ bureau ✓ retrouver

The A.I. will predict 85.7% of the time

Figure 9 – A long anchor with many disconnected tokens.

the pick step to select the smallest ratio of tokens in the anchor and in the sentence[13]:

$$x^* = \underset{x \in S \backslash S_{seen} \backslash S_{rejected}}{\operatorname{argmin}} \frac{|\phi_{\hat{y}}|}{|x|} \tag{15}$$

4.2.4 NoExp pick algorithm

As for the control method, we first considered adapting the original pick algorithm with a constant value for every token. However, this strategy encourages the selection of long messages, making the task more time consuming for the users. Therefore we used a random selection without replacement with NoExp.

4.3 User modifications

Users had two ways of modifying the model: adding samples and blacklisting tokens. The Delivery team adds samples to the training set in a daily basis while inspecting issues with the chatbots, and it is the most accessible way of maintaining a model in the platform, thus we kept this in order to imitate the real process. We added the possibility of blacklisting tokens inspired by positive results found by Ribeiro et al. [9] using this strategy in a similar experiment. Each user separately wrote new samples, annotated tokens, and wrote comments/rationales about the behaviors of the model on a text file, which was later manually processed and analyzed.

5 Results

We ran the experiment with eight subjects; one of them was discarded because we changed the test conditions after its participation - we solved some issues with the user interface (Figures 7 and 8), and adapted the pick method (Section 4.2). We had two experiments with NoExp, two with Anchors, and three with SHAP. In this section we present the users' outputs from the experiment (Section 4.3), a qualitative analysis, and the performances of the retrained models.

5.1 User outputs

Seen, rejected, and accepted examples As explained in Section 4.2.2, users had the choice to accept or reject examples presented to them. Table 2 shows the number of explanations rejected

[13]The notation here considers that $\phi_{\hat{y},j'} = 1$ if the token at the position j' is in the anchor, and 0 otherwise.

method	**user**	**rejected**	**accepted**	**total**	**n. of new sentences**
noexp	A	7	11	18	16
	B	41	16	57	27
	together	*48*	*27*	*75*	43 ($\approx$ **22**/user)
shap	C	6	7	13	35
	D	8	8	16	20
	E	8	6	14	53
	together	*12*	*15*	*27*	108 ($\approx$ **36**/user)
anchor	F	1	14	15	15
	G	3	18	21	0
	together	*4*	*19*	*23*	15 ($\approx$ **8**/user)

Table 2 – Explanations rejected and accepted by each users during the experiment sessions. The lines *together* show the number of *distinct* samples seen and by the group of users. The last column (on the right) shows the number of sentences created by the users during the experiment, used to increment the training set of the model.

and accepted by each user and grouped by method.

We see that users using SHAP inspected less examples than others, probably because they had more information to analyze at each one - besides the explanation on the predicted intent, users could also inspect the SHAP values of other intents. Users using no explanations did not have an overlap of samples due to the random selection, giving them a chance to see more intents.

We also see that SHAP users proportionally rejected sentences more than Anchors users. According to feedback from the participants, the selected sentences were often too long in the beginning of the sessions, which overwhelms the users. This happens because the pick method maximizes a metric that sums positive values associated to the tokens (see Equation 15), so longer sentences will naturally have a higher chance of being selected.

These results encourage the hypothesis that the pick methods did make sense for this task because the NoExp examples (picked at random) were rejected more often than the others. However, it would be unjustified to infer it for sure because we only had a few experiments, and most of the rejected examples from NoExp come from a single user - *B* rejected 41 examples.

New samples The last column on Table 2 shows the number of sentences created by each user in order to increment the training test. We see that those who used SHAP tended to create more sentences than the others, perhaps because it was easier to see which parts of the sentences were problematic. Although, Figure 17 reveals that SHAP users focused on fewer intents, while those without explanations distributed samples more evenly. Anchors was the least helpful in ading the creation of new samples; one of the users did not create samples at all, while the other created only 15 (less than any other user from NoExp and SHAP).

Word diversity We also compared how diverse the tokens were in the new sentences were with the ratio $\frac{\text{number of distinct tokens}}{\text{number of messages}}$[14]. In order to have reference values of this metric, we simulated it with the production data set. We selected a sample with the same number of messages than each of the methods (the values of *together* on the column *n. of new sentences* on Table 2), say, $n\_msgs = 108$ with SHAP, and measured the same ratio. We repeated this process 3,000 times, then plotted a histogram of the ratio values. This represents how much diverse the existing samples would be if we only had 108 of them.

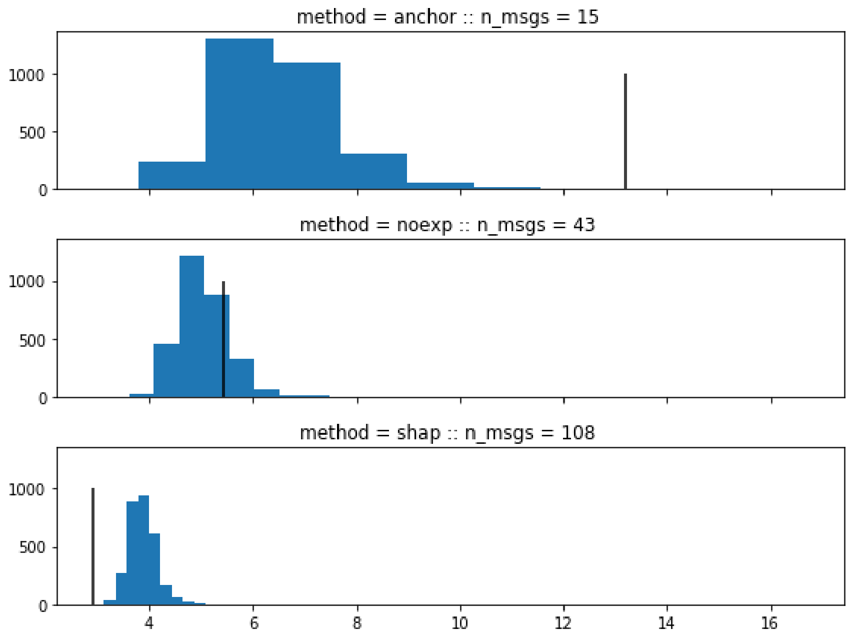


Figure 10 – Comparison of word diversity. We compare the diversity of words in the new samples from the experiment with subsets of the same size from the production data set. We randomly select a set with *n_msgs* sentences from the production dataset, and measure $\frac{\text{number of distinct words}}{\text{number of messages}}$. We repeat this 3,000 times, and plot a histogram of the ratio values. The black lines show the same metric with the samples from the experiment.

Figure 10 shows that the process of our experiment did not incite users to write sentences more varied than usual, and SHAP users wrote relatively repeated sentences. Anchors users wrote way more varied sentences compared to its respective distribution, but since it only concerns a single user that could be merely an outlier behavior.

Blacklists Table 3 shows the blacklisted tokens grouped by method. According to NoExp users, if felt unsafe to blacklist entities only based on the probabilities because they had no clue to identify which ones might be damaging. On the other hand, those who saw explanations clearly tried to attack issues with stop words. It is worth noting that the token *__entity_1* is appended to nearly any message, thus it appeared in almost all the examples. We see a few words in common between the blacklists from Anchors and SHAP users. The latter blacklisted some tokens, like *bonjour*, based on the fact that they had high importance values (either positive or negative) while it did not have any importance to comprehend the messages' intents.

As the users saw unreasonable importance values given to these tokens, we expected that getting rid of them could improve the model's performance because it would remove a source of "distraction". However, we show in Section 5.3 that this strategy did not work.

[14]We preprocessed the messages to count only tokens known by the model (thus, *Bonjour* and *bonJour* are considered as the same word), and discarded entity tokens because they are not part of the original message (appended to the message by the entity detection models).

method	**tokens**
noexp	__entity_1300
shap	"," "." "?" bjr **bonjour** car cela cependant **de** et hello la le **les** m' ma mais merci **mes mon** notre or salut svp un une votre vous **__entity_1**
anchor	-il ai **bonjour** contre **de** du **les mes** moi **mon** plus **__entity_1**

Table 3 – Blacklisted tokens grouped by method. The punctuation tokens are shown in quotes.

5.2 Qualitative analysis

We asked for feedback from the experiment users, and compiled the most compelling points in this section.

General Some of the messages from production did not fit any of the intents because they were too specific. Not rarely messages carried more than one intent, making it hard to decide how it *should* be classified. Using real messages with a large scope (about a hundred intents) was particularly complicated for those who were unfamiliar with the bot because they had to figure out the expected behavior.

NoExp The experiment encouraged the users to look at the training samples, making them realize that there were misplaced samples and some scopes were ill defined. As expected, users had a hard time to analyze the prediction because they had nothing to assist the task.

SHAP Although the explanations made it very clear to understand which tokens were misunderstood (see Figure 8a), it was unclear how to take action to correct bad behaviors. One of the users only took action in cases where the correct intent was the second highest probability, trying to compensate the model's mistake.

Anchors Although this method is designed to make it easy to generalize an explanation, in practice it was often challenging because long messages with long anchors required considerable effort to interpret.

5.3 Retrained models

We grouped all the new sentences from the users by method, then retrained two versions for each group, with and without the blacklist. In order to compare the usefulness of the sentences created during the experiment, we also trained a third *control* model using sentences from the production training data set. These sentences were randomly selected and removed from the test split. We did this for each method, selecting the same number of added sentences in each group. The performance was evaluated using the macro and micro average of the F1-score and recall in Figure 11. Since some of the intents had very few sentences in the test set, the averages were artificially affected with threshold effects (e.g.: if only 10 sentences are used, a single test sentence can increase or decrease the scores for that class by 10%); thus we only used intents with at least 30 support sentences in the test set.

As we can see on Figure 11, the explanation methods in general did not make improvements more efficiently than their respective control models. Surprisingly, NoExp even resulted in a better performance than using explanations - perhaps because those users tended to add samples more evenly between intents (Figure 17). Nevertheless, both SHAP and Anchors resulted in a better model even with relatively few new examples (e.g.: around 100 were created using SHAP, which is only an extra 2% of the training set).

Most surprisingly, the blacklist did not improve the model in any of the cases, and SHAP's case was the most pronounced. This possibly happened because the explanations gave the wrong impression that removing them would force the model to "pay attention" to the right words, but it degraded the text comprehension.

6 Conclusion

We presented Mindsay, its activity, and its market; then we briefly discussed how the conversational agents are are structured, how the user messages are processed, how the company's internal operators manage these chatbots, which are powered by more than 1,000 text classification models in production, and the challenges of their mission. Having set this context, we introduced basic notions of explainability and projected its interest for Mindsay's use case. Next, we described and commented a handful of methods proposed in the literature during the latest years, detailing two methods, SHAP and Anchors, that we used in a human experiment.

We proposed a procedure to imitate the continuous improvement process at Mindsay - where chatbot sessions are analyzed, and operators correct issues adding sentences to the training set. We framed this process to measure the effect of using SHAP or Anchors methods to assist the task. Unlike the real process, our procedure proactively generated local explanations, prioritized them with appropriate strategies, and users could blacklist tokens. We observed that SHAP had a positive user feedback, but its results, like Anchor's, were not better than the control method. We also saw that SHAP seems to make it easier for users to propose new sentences, but they end up using repetitive wordings. User feedback for Anchors, unlike the author's claims, showed that the explanations were not easy to use. Finally, we also realized that, contrarily to our expectations, blacklisting tokens had a negative impact on the model's performance.

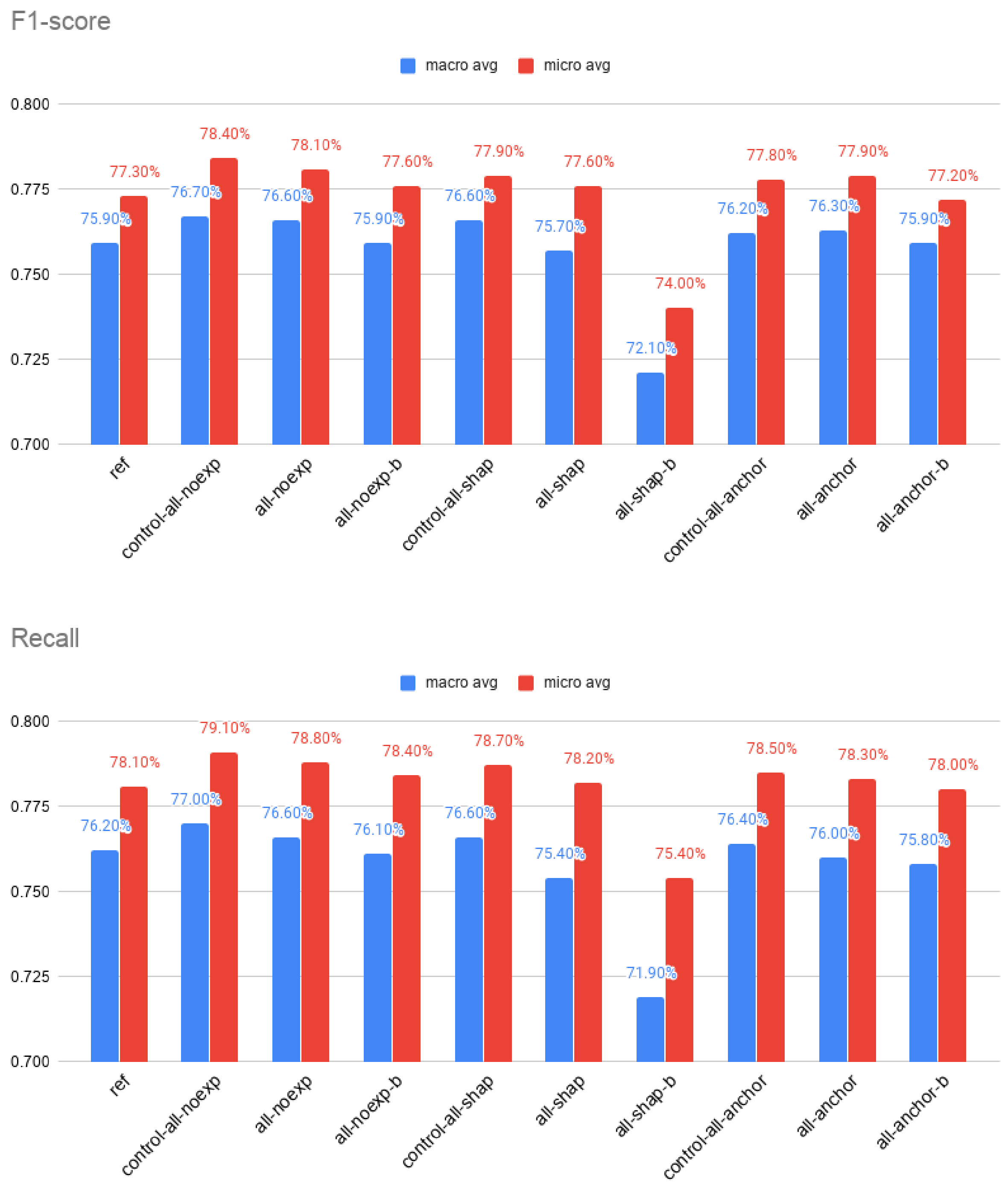


Figure 11 – Performance comparison of different models. *ref* is the original model (without new sentences from the experiment); the prefix *all-* denotes that all the sentences from the users of the same method were used together; the suffix *-b* means that the blacklist was used, otherwise it was not; the prefix *control-* denotes models that were re-trained the same number of new sentences than their pairs without the prefix, but with samples that already existed in production and were ignored for the training (i.e.: sentences not from the experiment).

7 Bibliography

[1] A. B. Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. García, S. Gil-López, D. Molina, R. Benjamins, R. Chatila, and F. Herrera. Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI. *arXiv:1910.10045 [cs]*, Dec. 2019. URL http://arxiv.org/abs/1910.10045. arXiv: 1910.10045.

[2] J. Chen, L. Song, M. J. Wainwright, and M. I. Jordan. Learning to Explain: An Information-Theoretic Perspective on Model Interpretation. *arXiv:1802.07814 [cs, stat]*, June 2018. URL http://arxiv.org/abs/1802.07814. arXiv: 1802.07814.

[3] B. d' Alessandro, C. O'Neil, and T. LaGatta. Conscientious classification: A data scientist's guide to discrimination-aware classification. *Big Data*, 5(2):120–134, Jun 2017. ISSN 2167-647X. doi: 10.1089/big.2016.0048. URL http://dx.doi.org/10.1089/big.2016.0048.

[4] X. Jin, Z. Wei, J. Du, X. Xue, and X. Ren. Towards Hierarchical Importance Attribution: Explaining Compositional Semantics for Neural Sequence Models. Sept. 2019. URL https://openreview.net/forum?id=BkxRRkSKwr.

[5] E. Kaufmann and S. Kalyanakrishnan. Information Complexity in Bandit Subset Selection. *Journal of Machine Learning Research*, page 24, 2013.

[6] S. M. Lundberg, P. G. Allen, and S.-I. Lee. A Unified Approach to Interpreting Model Predictions. Technical report, 2017. URL https://github.com/slundberg/shap.

[7] V. N. Murthy, V. Singh, T. Chen, R. Manmatha, and D. Comaniciu. Deep Decision Network for Multi-class Image Classification. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2240–2248, June 2016. doi: 10.1109/CVPR.2016.246. ISSN: 1063-6919.

[8] H. Núñez, C. Angulo, and A. Català. Rule extraction from support vector machines. In *ESANN*, 2002.

[9] M. T. Ribeiro, S. Singh, and C. Guestrin. Why Should I Trust You? Explaining the Predictions of Any Classifier. 2016. doi: 10.1145/2939672.2939778. URL http://dx.doi.org/10.1145/2939672.2939778. ISBN: 9781450342322_eprint: 1602.04938v3.

[10] M. T. Ribeiro, S. Singh, and C. Guestrin. Anchors: High-Precision Model-Agnostic Explanations. In *AAAI*, 2018.

[11] M. T. Ribeiro, S. Singh, and C. Guestrin. Semantically Equivalent Adversarial Rules for Debugging NLP models. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 856–865, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1079. URL https://www.aclweb.org/anthology/P18-1079.

[12] S. Robinson. Interpreting bag of words models with SHAP. https://sararobinson.dev/2019/04/23/interpret-bag-of-words-models-shap.html, 2019. Accessed: 2020-05-07.

[13] L. S. Shapley. A value for n-person games. In A. E. Roth, editor, *The Shapley Value: Essays in Honor of Lloyd S. Shapley*, pages 31–40. Cambridge University Press, Cambridge, 1988. ISBN 978-0-521-36177-4. doi: 10.1017/CBO9780511528446.003. URL https://www.cambridge.org/core/books/shapley-value/value-for-nperson-games/1AA9D343DE7A87A97F69E999D329B57A.

[14] M. Sundararajan and A. Najmi. The many shapley values for model explanation, 2019.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2017.

[16] A. Wan, L. Dunlap, D. Ho, J. Yin, S. Lee, H. Jin, S. Petryk, S. A. Bargal, and J. E. Gonzalez. NBDT: Neural-Backed Decision Trees. *arXiv:2004.00221 [cs]*, Apr. 2020. URL http://arxiv.org/abs/2004.00221. arXiv: 2004.00221.

Appendices

A Bibliographic review

In this section I comment the most interesting aspects of some articles reviewed during our bibliographic research. I highlight features that make these methods unique, but they are not necessarily the driving factors that lead our choices. For instance, Jin et al. [4] propose an approach that creates explanations more informative than LIME [9]; however, its explanations are more laborious to interpret, its computation is more expensive, and the code available was hard to integrate with Mindsay's stack.

A.1 "Why Should I Trust You?" Explaining the Predictions of Any Classifier

Ribeiro, Singh, and Guestrin [9]

The authors introduces a generic formulation, creating a base framework to generate local explanations. A model decision is explained by training a simple model in a limited input region such that the original model can replaced it. This local proxy is a linear model, which the user can easily interpret looking at its weights. The formulation of the method is flexible, making it convenient to create variations of the original version, for instance using different types of proxy models, or using different loss functions when training the them. The author also introduces a technique to select which sentences to expose to the user in order to maximize the model comprehension with a fixed budget of explanations.

A.2 A Unified Approach to Interpreting Model Predictions

Lundberg, Allen, and Lee [6]

The authors shows that several earlier methods in the literature fit a common framework, that they call *additive local explanations*, then introduce three desirable properties to it. The only solution that respects these properties can be found using the Shapley values [13] associated to the problem, creating an interesting connection between machine learning and game theory. The authors present slight variations of the same formulation, using different assumptions that lead to adapted approximators of the Shapley values, then show how their approach is faster and more stable than a previously proposed technique. The SHAP (SHapley Additive exPlanations) values can be applied to any family of classifier, but the article lacks clear guidance about the configuration of the framework for specific applications.

A.3 Learning to Explain: An Information-Theoretic Perspective on Model Interpretation

Chen, Song, Wainwright, and Jordan [2]

This article attacks the explainability problem using an information theoretic approach. Similarly to other methods, Learning to Explain (L2X) uses a secondary (trained) model to inspect the target one, but it does not create a proxy model (a simpler classifier that learns to mimic the explained one); instead, it trains a regressor to predict the mutual information between the input's features and the model's output. One can use its approximations to select the features of an example that are most important for a prediction. When used with text, this gives the k

most relevant words for a predicted class. The most interesting characteristic of L2X is that it generates a global explainer (that is, that considers the entire input space) capable of creating local explanations (related to a specific prediction). Besides, this paper presents an analysis of computational cost to generate explanations comparing it with LIME [9] and SHAP [6] - whose respective papers completely ignore the subject.

A.4 Anchors: High-Precision Model-Agnostic Explanations

Ribeiro, Singh, and Guestrin [10]

In this article, the authors propose a distinct type of local explanation, showing to the user a sub set of features that guarantees that the model's behavior does not change. The computation method used interestingly adapts the KL-LUCB algorithm [5] to find such subsets. Anchor explanations directly attack an issue often ignored in other articles: the precision of local explanations. The motivation for using local (as opposed to global) explanations is the assumption that simple models are reliable to understand the underlying model if it is limited to a smaller input space. However, the authors often ignore this aspect, not validating the hypothesis, therefore not telling the user when an explanation is trustable. An anchor explanation directly tells you what are the conditions to extrapolate conclusions from an explanation; it measures quantitatively how trustable those extrapolations are, and it provides concrete examples of neighbor inputs (relative to the explained one) where the explanation is applicable and when it is not.

A.5 Semantically Equivalent Adversarial Rules for Debugging NLP models

Ribeiro, Singh, and Guestrin [11]

Semantically Equivalent Adversarial Rules (SEARs) is a debug tool designed to search for classification mistakes and summarize them. The method searches adversarial examples without changing the semantic meaning of the sentence, which is an essential difference from other adversarial attacks. This process is repeated with a labeled dataset, then, most interestingly, the post-processing step synthesizes the model's fragilities with logic rules, for instance, telling the user that changing a generic sentence from *"NOUN is ADJECTIVE"* to *"NOUN's ADJECTIVE"* causes misclassifications. These rules are useful because they provide actionable insights to augment the model's training set; in the previous example, the user could programmatically augment the training dataset to provide it with such variations.

A.6 Towards Hierarchical Importance Attribution: Explaining Compositional Semantics for Neural Sequence Models

Jin, Wei, Du, Xue, and Ren [4]

Inspired by additive importance approaches like LIME [9] and SHAP [6], the hierarchical importance attribution methods are like an evolution in this sub-field of explanations. Like the former, the latter assign real values to parts of a sentence that are associated to how much they influence the probability of a class. However they do not see a sentence as a simple combination of tokens, but as a hierarchical structure - such as a constituency parsing tree. The importance values of a sentence's constituents are not required to linearly add up to the model's output probability, and they have a degree of context independence, therefore giving them a sense of global class-polarity (a positive or negative influence towards a given class). These two characteristics enable more complex, informative visualizations, for instance, showing if a model is capable of understanding negations. Furthermore, the notion of context independence creates a link between local and global explanations, which does not exist in linear explanations because they are tied to an input instance.

A.7 NBDT: Neural-Backed Decision Trees

Wan, Dunlap, Ho, Yin, Lee, Jin, Petryk, Bargal, and Gonzalez [16]

The Neural-Backed Decision Tree is an architecture adaptation that can be applied to any classification neural network after training, resulting in a more interpretable version of the model without hurting its performance. It uses the learned network to generate a decision tree based on each classe's centroid vector on the last layer (each row corresponds to a class vector in the learned feature space). The construction of the decision tree smartly leverages the knowledge gained by the base network during the training by clustering these class vectors. The final architecture can be inspected at prediction time, bringing light to how the model differentiates the classes. This type of analysis can reveal implicit biases in the model, such as learning to classify from the background instead of the content in image applications.

B Other explainability methods

In this section we summarize some of the methods that we considered during our research that ended up discarding, and explain why they were discarded.

B.1 LIME (Local Interpretable Model-agnostic Explanations)

The core assumption for LIME is that one can approximate a complex model with a linear one by restraining the input space to a small region; then one can directly analyse the weights of the linear proxy model.

Let x be a sentence with the tokens x_j, where $j = 1, \ldots, d$. We then introduce the *simplified input* $z' \in \{0,1\}^d$, where z'_j represents the presence (if $z'_j = 1$) or absence (if $z'_j = 0$) of the token x_j. This defines the transformation $h_x(z')$, which takes each vector z' to a variation of the sentence x. For instance, take the reference phrase $x =$ "I love Paris"; the vector $z' = [1,0,1]$ gives the sentence $z = h_x(z') =$ "I Paris" because the second token is set to 0.

The proxy model $g(z') = w_g \cdot z'$, with $w_g \in \mathbb{R}^d$, is trained with samples from the simplified input space to fit $f(h_x(z')) \in [0,1]$[15], the original model's output probability with the corresponding sentence. In other words, we wish to find a $g \in G$ such that $g(z') \approx f(h_x(z')) = f(z)$. We use a proximity measure $\pi_x(z) = \exp\left(-D(x,z)^2/\sigma^2\right)$, where D is the cosine distance, to give more weight to samples closer to the original sentence x, and we penalize the model's complexity using the function $\Omega(g) = \infty \mathbb{1}_{[\|w_g\|_0 > B]}$, where B is the *budget* parameter (i.e.: the maximum number of tokens with non-zero importance values). Putting it all together, we solve the following optimization[16]:

$$
\begin{aligned}
g^* &= \underset{g \in G}{\operatorname{argmin}} \, \mathcal{L}(f, g, \pi_x) \; + \; \Omega(g) \\
g^* &= \underset{g \in G}{\operatorname{argmin}} \sum_{z' \in \{0,1\}^d} \pi_x(z) \left(f(z) - g(z')\right)^2 \; + \; \Omega(g)
\end{aligned}
\tag{16}
$$

Then we use the model's weights, w_{g^*} as the importances values. Figure 12 shows an intuitive representation of the elements that compose LIME's formulation. In this toy example we see a non-linear decision boundary between two classes (red and blue), and a linear approximation close to instance in bold. The sizes of the samples in the illustration represent the weights $\pi_x(z)$ of the perturbed sentences.

B.2 Hierarchical context-independent importances

Following the same basic principle from LIME and SHAP (the association of importance values to individual tokens, also known as *polarity* when used with sentiment analysis models), Jin et al. [4] propose a framework that explains not only individual tokens, but also subphrases in a sentence. Unlike those, however, their algorithms generate non-additive explanations - that is, $\phi_{j':j''}(f,x) \neq \sum_{j=j'}^{j''} \phi_j(f,x)$[17]- that are independent of their contexts. This makes it possible to

[15]With a multi-class model, there is one proxy $g \in G$ per class.

[16]The notation was simplified by implicitly replacing $h_x(z') = z$.

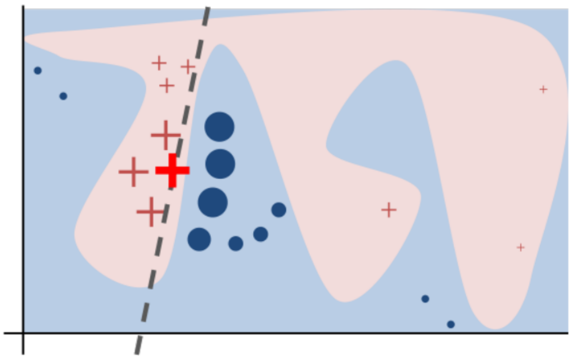


Figure 12 – Toy example of an explanation with LIME illustrating the main elements of framework.

extract more information about a model's behavior from a single explanation.

Take the example of the Figure 13a, for instance. While an additive explanation, on the left, would force one of the two words to be negative in order to ensure that the sum of the factors equals the negative output, the context-independent approach makes it clear that the model is capable of understanding negation. Notice how the word "interesting" alone gets a positive polarity (i.e.: it has a positive importance towards the class *positive sentiment*), which makes sense if it was used out of this context because the word by itself sends a *positive sentiment*. The word "not" gets a neutral value because it can be both a positive or negative depending on its context, it lacks of intrinsic meaning. Then the explanation makes it clear that, when combined, the two words are correctly understood as a *negative sentiment*.

Although a powerful tool, we did not pursue with this method for a few reasons. First, the number of classes of our use case (a few hundreds) could make its usage too complex, overwhelming the users with too much information. Second, the method heavily depends on being capable of generating sentences with conditional distributions, which would demand more data than what we have available. Third, and perhaps most importantly, the Python code available is only compatible with PyTorch and is not yet mature/stable.

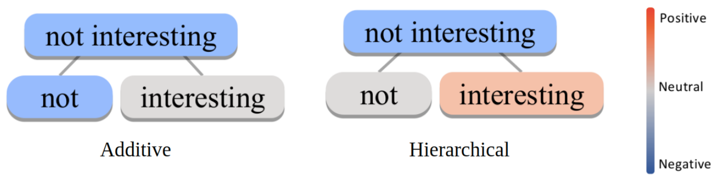
B.3 Learning to Explain (L2X)

Chen et al. [2] propose an information-theoretic approach more than two orders of magnitude faster than SHAP and LIME. Instead of optimizing a proxy model for each explanation, this method consists of training a single model that learns to predict the mutual information between subsets of the input (in our case, tokens in the sentence) and the output variable (the class probability).

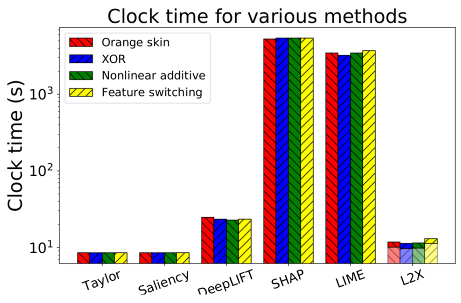
During the explaining phase, the model takes a sample sentence x with d tokens, and outputs a weight vector $\omega \in \mathbb{R}^d$ representing the importances of each respective token. Since L2X has a single training phase, where each explanation is an estimation with the trained model, the computational efficiency largely surpasses other methods, as we can see on Figure 13b.

The inconvenience with this method is that one ends up with a second learning problem. You

[17] $\phi_j(f,x)$ denotes the polarity of the token x_j in for the prediction $f(x)$, and, similarly, $\phi_{j':j''}(f,x)$ denotes the polarity of the subphrase from the token $x_{j'}$ until the token $x_{j''}$.



(a)



(b)

Figure 13 – (a) Additive v.s. hierarchical context-independent explanations of a sentiment analysis model. The additive explanation (on the left) assigns values to the individual words (*not* and *interesting*) such that the sum equals the probability of the whole sentence (*not interesting*), which is negative, so the word *not* gets a negative value. On the other hand, the hierarchical explanation assigns a neutral polarity to *not* and a positive polarity to *interesting*, although together they have a negative meaning. (b) (Figure extracted from [2]) Comparison of computational efficiency of several explanation methods. The y-axis shows the clock time of explaining 10,000 samples in log scale. The semi-transparent bar in L2X (Learning to Explain) represents the training time. L2X is faster than LIME and SHAP in various problems.

would need to train not only a classifier model, but also its respective explainer each time. This condition would be impractical at Mindsay because each user node (see Section 2.1) would require a second training, giving the users two times more models to monitor and keep track of. Besides, it is hard to know when a prediction from this model can be trusted, as the authors do not provide guidance about it in the paper. Finally, the usability of this technique is less flexible than others like LIME and SHAP, while it provides a similar kind of explanation. Therefore, we decided to discard this method.

C Practical considerations

In this section we list practical concerns encountered while implementing the two methods in our experiment.

C.1 SHAP

Number of samples Although the approximated solution hugely decreases the number of calls to the model f, it still requires a considerable number of evaluations when using textual inputs. The default (and recommended) number of samples is $n = 2 \times d + 2048$; that is, if a call takes $1msec$, an explanation takes (at least) around 2 seconds. As we wish to explain a large number of explanations (thousands or tens of thousands)[18], we had to precompute them in advance before showing to the user.

Background Another important parameter is the background data set[19], which was more recently discussed in detail by Sundararajan and Najmi [14]. This is the distribution used to replace a feature's value when it is absent. With other types of input this can be a conditional empirical average - conditioned to the input being explained. However, in our use case, the most simple and feasible way is to replace a token by a mask token or simply remove it from the sentence.

Non global features Finally, we used SHAP by defining a new explainer for every explanation, therefore not considering tokens as features that can be compared from one example to another. This is important because it allows us to treat repeated tokens as distinctive tokens, and, unlike Robinson [12], we do not limit the explanations to consider bag of word models, keeping the sequence of the tokens in the perturbed sentences during the proxy optimization. Furthermore, this approach avoids confusing explanations like Figure 14, where the explanation gives positive importances to the *non-presence* of tokens.

Unsuccessful approximations The author's implementation[20]- used in this project - sometimes failed to converge and find the correct SHAP values. To avoid confusion with nonsense explanations, we got rid of all the explanations that did not meet the local accuracy property within a fixed margin. In other words, if

$$|f(x) - g^*(x')| > \tau \tag{17}$$

we rejected the explanation g^*. In our experiment, we used $\tau = 0.03$.

Non-zero base values As you can notice from Equation 1, the proxy models have a bias parameter ϕ_0. That means that if a model returns a non-zero probability with an empty sentence, one or more classes might get a positive probability as a *base value*, as ϕ_0 is called. This is also a possible source of confusion because classes with non-zero base probability are forced to give

[18]Link to the documentation in the source code.

[19]Although this parameter is barely mentioned in SHAP's original paper, it is required to use the method in their Python package.

[20]Link to SHAP's author's official implementation.

negative importance values to a few tokens in order to cancel out this value. Our problem contains more than a hundred classes, and we verified that only one class had a non-zero base value, so we decided to ignore this issue - assuming the risk of eventually getting unclear explanations with that particular class.

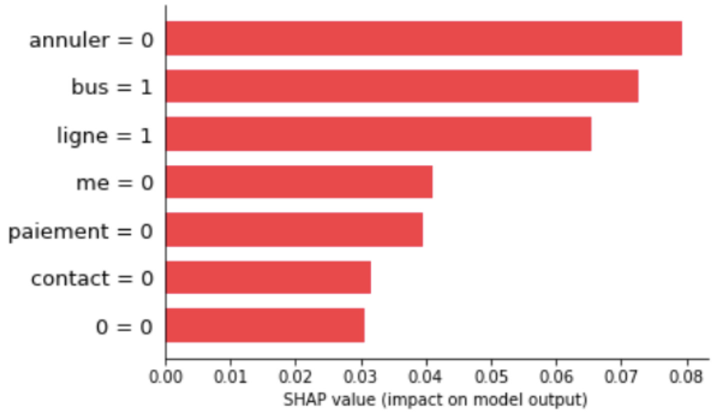


Figure 14 – SHAP: Using the bag of words approach with SHAP generates confusing artifacts when absent tokens gets non-zero importance values.

C.2 Anchors

Hyperparameters As the reader might have realized, in Section 3.3.2, where we describe Anchors's framework, we introduce several hyperparameters that are neither justified nor optimized. The reason for that is the lack of orientation about these values in Anchors's original paper. [10] use "reasonable" values, which are not explained, and leave the sensitivity analysis for future work, which we could not find. We used the same values ($B = 10$, $\delta = 0.05$, $\varepsilon = 0.10$), except for the minimum precision, τ, which we set to $\tau = 0.85$.

Perturbed distribution Anchors heavily depends on the capacity of querying the distributions $D_x(\cdot)$ and $D_x(\cdot|A)$; the better the variations are, the better we have a realistic view of the model's behavior. The author's implementation offers the possibility of using distributions based on word embeddings and transformers to generate these distributions, but unfortunately we could not use them because the technical adaptions required would not fit our time constraints. Instead, we used the simplest version, which only masks the original words with the `UNK` (for *unknown*).

Disconnected tokens As Anchors was not specifically designed for sequential data like text, its input variations ignore the notion of order between the features (tokens in our case), so it queries the model with sentences where disconnected words are exchanged. Considering that we used the masking scheme for the perturbation distribution, this results in potentially meaningless - or, at least, hardly interpretable - anchors like the one in Figure 9. Perhaps an adaptation of the method could address this issue to make sure that only subsequent words are selected together.

Multi-class overlap Last but not least, Anchors do not guarantee the uniqueness of a solution; not even the uniqueness of predicted class. That is, a sentence can have several anchors which actually are associated with different prediction in a multi-class classifier, potentially making it confusing for the user. Like the author, we simply ignore other possible solutions and show an anchor associated to the origianally predicted class, but note that other explanations are possible.

D Auxiliary figures

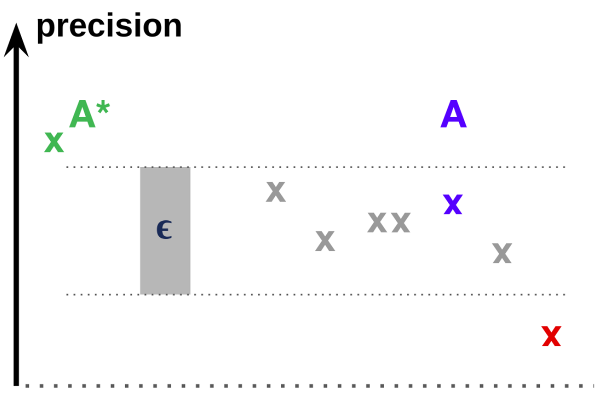
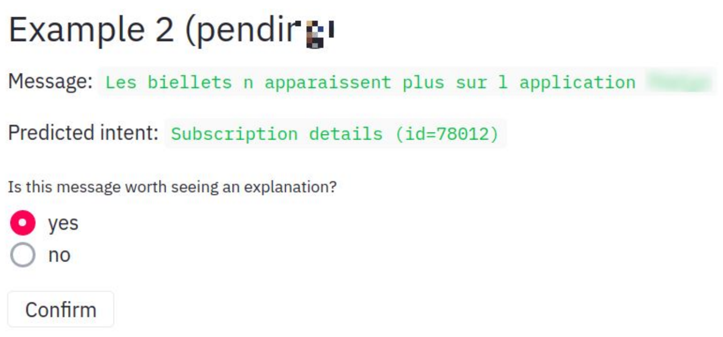


Figure 15 – Illustration of the result guarantee from the KL-LUCB algorithm [5] in Anchors' formulation with $B = 1$, the number of candidates kept at each step of the beam search (see Equation 10). This result tells us that the KL-LUCB's output has a probability of $1-\delta$ of being within a margin of size ε under the optimal latent reward (the predicate's precision in this case). With $B > 1$ (i.e.: case where the algorithm searches the $B-$best latent rewards), the algorithm guarantees that the lowest latent reward of the selected arms (predicates for Anchors) is within the $\varepsilon-$margin bellow the worst latent reward of the optimal set of arms.



Example 2 (pendir[redacted]

Message: `Les biellets n apparaissent plus sur l application` [redacted]

Predicted intent: `Subscription details (id=78012)`

Is this message worth seeing an explanation?

[✓ yes]
no

Confirm

Figure 16 – Sentence confirmation prompt. During the experiment sessions, the users could reject the sentences proposed by the pick algorithm, for instance, because the sentence was too long, or because it is too similar to previously seen sentences.

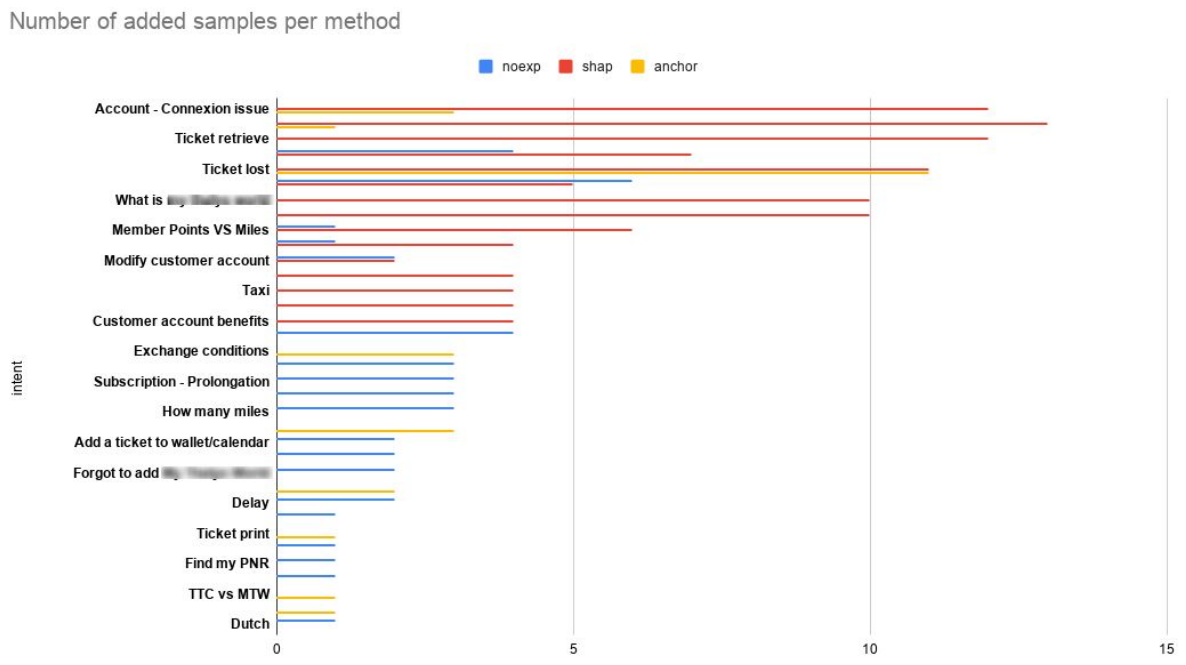


Figure 17 – Number of examples added to each intent per method. We aggregated the users' new sentences (issue of the experiment sessions). We see that SHAP users concentrated their efforts on a handful of intents, while NoExp users covered more varied intents with less sentences per intent.