Interpreting Black-Box Classifiers Using Instance-Level Visual Explanations

Paolo Tamagnini
Sapienza University of Rome
Rome, RM 00185, Italy

Josua Krause
New York University
Tandon School of Engineering
Brooklyn, NY 11201, USA

Aritra Dasgupta
Pacific Northwest National Laboratory
Richland, WA 99354, USA

Enrico Bertini
New York University
Tandon School of Engineering
Brooklyn, NY 11201, USA

ABSTRACT

To realize the full potential of machine learning in diverse real-world domains, it is necessary for model predictions to be readily interpretable and actionable for the human in the loop. Analysts, who are the users but not the developers of machine learning models, often do not trust a model because of the lack of transparency in associating predictions with the underlying data space. To address this problem, we propose *Rivelo*, a visual analytics interface that enables analysts to understand the causes behind predictions of binary classifiers by interactively exploring a set of instance-level explanations. These explanations are model-agnostic, treating a model as a black box, and they help analysts in interactively probing the high-dimensional binary data space for detecting features relevant to predictions. We demonstrate the utility of the interface with a case study analyzing a random forest model on the sentiment of Yelp reviews about doctors.

KEYWORDS

machine learning, classification, explanation, visual analytics

ACM Reference format:
Paolo Tamagnini, Josua Krause, Aritra Dasgupta, and Enrico Bertini. 2017. Interpreting Black-Box Classifiers Using Instance-Level Visual Explanations. In *Proceedings of HILDA'17, Chicago, IL, USA, May 14, 2017,* 6 pages.
DOI: http://dx.doi.org/10.1145/3077257.3077260

1 INTRODUCTION

In this paper we present a workflow and a visual interface to help domain experts and machine learning developers explore and understand binary classifiers. The main motivation is the need to develop methods that permit people to inspect what decisions a model makes after it has been trained.

While solid statistical methods exist to verify the performance of a model in an aggregated fashion, typically in terms of accuracy over hold-out data sets, there is a lack of established methods to help analysts interpret the model over specific sets of instances.

This kind of activity is crucial in situations in which assessing the *semantic validity* of a model is a strong requirement. In some domains where human trust in the model is an important aspect (e.g., healthcare, justice, security), verifying the model exclusively through the lens of statistical accuracy is often not sufficient [4, 6, 8, 15, 16, 24]. This need is exemplified by the following quote coming from the recent DARPA XAI program: "*the effectiveness of these systems is limited by the machines current inability to explain their decisions and actions to human users [...] it is essential to understand, appropriately trust, and effectively manage an emerging generation of artificially intelligent machine partners*" [9].

Furthermore, being able to inspect a model and observe its decisions over a data set has the potential to help analysts better understand the data and ultimately the phenomenon it describes.

Unfortunately, the existing statistical procedures used for model validation do not communicate this type of information and no established methodology exists.

In practice, this problem is often addressed using one or more of the following strategies: (1) build a more interpretable model in the first place even if it reduces performance (typically decision trees or logistic regression); (2) calculate the importance of the features used by a model to get a sense of how it makes decisions; (3) verify how the model behaves (that is, what output it generates) when fed with a known set of relevant cases one wants to test.

All of these solutions however have major shortcomings. Building more interpretable models is often not possible unless one is ready to accept relevant reductions in model performance. Furthermore, and probably less obvious, many models that are considered *interpretable* can still generate complicated structures that are by no means easy to inspect and interpret by a human being (e.g., decision trees with a large set of nodes) [8]. Methods that rely on calculation of feature importance, e.g., weights of a linear classifier, report only on the *global* importance of the features and does not tell much about how the classifier makes decisions in particular cases. Finally, manual inspection of specific cases works only with a very small set of data items and does not assure a more holistic analysis of the model.

To address all of these issues we propose a solution that provides the following benefits:

HILDA'17, Chicago, IL, USA

978-1-4503-5029-7/17/05...$15.00
DOI: http://dx.doi.org/10.1145/3077257.3077260

(1) Requires only to be able to observe the input/output behavior of a model and as such it can be applied to any existing model without having access to its internal structure.
(2) Captures decisions and feature importance at a local level, that is, at the level of single instances, while enabling the user to obtain an holistic view of the model.
(3) It can be used by domain experts with little knowledge of machine learning.

The solution we propose leverages *instance-level explanations*: techniques that compute feature importance locally, for a single data item at a time. For instance, in a text classifier such techniques produce the set of words the classifier uses to make a decisions for a specific document. The explanations are then processed and aggregated to generate an interactive workflow that enables the inspection and understanding of the model both *locally* and *globally*.

In Section 4, we will describe the workflow (Figure 2) in detail which consists of the following steps. The system generates one explanation for each data item contained in the data set and creates a list of features ranked according to how frequently they appear in the explanations. Once the explanations and the ranked list are generated, the user can interact with the results as follows: (1) the user selects one or more features to focus on specific decisions made with them; (2) the system displays the data items explained by the selected features together with information about their labels and correctness; (3) the user inspects them and selects specific instances to compare in more detail; (4) the system provides a visual representation of the descriptors / vectors that represent the selected data items (e.g., words used as descriptors in a document collection) and permits to visually compare them; (5) the user can select one or more of the descriptors / vectors to get access to the raw data if necessary (e.g., the actual text of a document).

In the following sections, we describe this process in more details. We first provide background information on related work, we then describe the methods used to generate the explanations followed by a more detailed description of the visual user interface and interactions developed to realize the workflow. Finally, we provide a small use case to show how the tool works in practice and conclude with information on the existing limitations and how we intend to extend the work in the future.

2 RELATED WORK

Different visual analytics techniques and tools have been developed to inspect the decision-making process of machine learning models and several authors advocated for the need to make models more interpretable [12]. In this section, we describe the previous work that is most relevant to our proposed model explanation process.

Many of the explanation methods investigated by researchers employ a white-box approach, that is, they aim at visualizing the model by creating representations of the internal structures of the models. For instance, logistic regression is often used to create a transparent weighting of the features and visualization systems have been developed to visualize decisions trees [22] and neural networks [14, 19]. These methods however can only be applied to specific models and, as such, suffer from limited flexibility.

Another option is to treat the model as a black-box and try to extract useful information out of it. One solution developed in the past is the idea of training a more interpretable model out

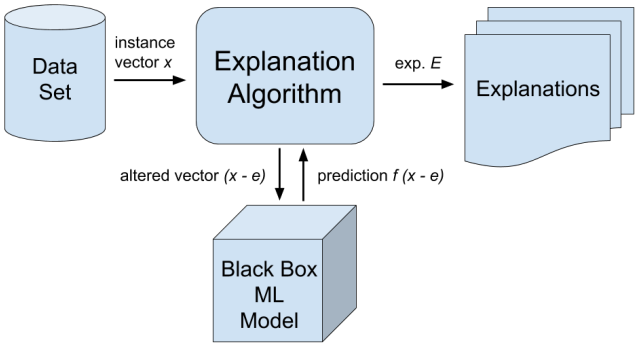

Figure 1: Illustrating the process of explanation computation. x is an observed instance vector. We treat the ML model as function f that maps a vector to a prediction score. The explanation algorithm tries to find the shortest vector e for which $f(x - e)$ is below the threshold. This vector e serves as an *explanation* for the prediction.

of an existing complex model, e.g., inferring rules from a neural network [7]. A more recent solution is the idea of generating local explanations that are able to explain how a model makes a decision for single instances of a data set [11, 13, 20]. These works are excellent solutions to the problem of investigating single instances but there are no established methods to go from single instances back to a global view of the model. This is precisely what our work tries to achieve by using instance-level explanations and embedding them in an interactive system that enables their navigation.

Another related approach is to visualize sets of alternative models to see how they compare in terms of the predictions they make. For instance, *ModelTracker* [1] visualizes predictions and their correctness and how these change when some model parameters are updated. Similarly, *MLCube Explorer* [10] helps users compare model outcomes over various subsets and across multiple models with a data cube analysis type of approach. One main difference between these methods and the one we propose is that we do not base our analysis exclusively on model output, but also use the intermediary representation provided by explanations. This enables us to derive more specific information about *how* a model makes some decisions, and go beyond exploring *what* decisions it makes.

Somewhat related to our work are interactive machine learning solutions in which the output of the model is visualized to allow the user to give feedback to the model and improve its decisions [2, 3, 21]. The goal of our work however is to introduce methods and tools that can be easily integrated in existing settings and workflows adopted by domain experts, and as such does not rely on the complex modifications necessary to include explicit user feedback in the process.

3 INSTANCE-LEVEL EXPLANATIONS

An instance level explanation consists of a set of features that are considered the most responsible for the prediction of an instance, i.e., the smallest set of features which have to be changed in the instance's binary vector to alter the predicted label. We used a variant of the instance-level explanation algorithm designed by Martens and Provost [17] for document classification.

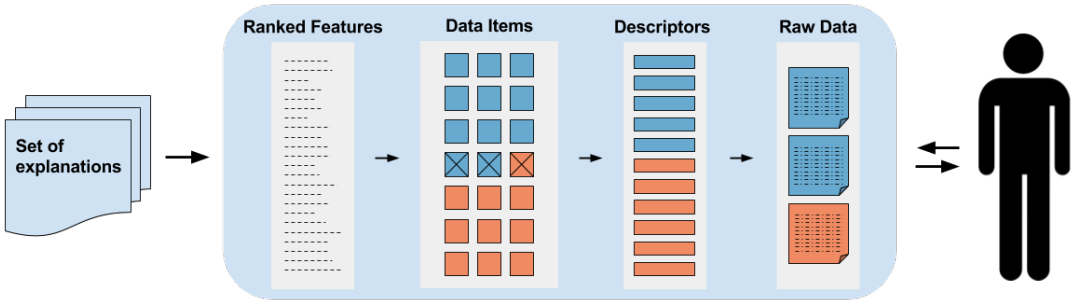

Figure 2: The *Rivelo* workflow involves the following user interactions: selection of features, selection of the explanation, vectors inspection, and exploration of the raw data. By switching back and forth between those steps, the user can freely change the selections in a smooth and animated visualization. This explanation-driven workflow can extract global behaviours of the model from patterns of local anomalies through human interaction.

The overall process of the explanation generation is illustrated in Figure 1. *Rivelo* computes an explanation for each instance using the input-output relationships of a black-box model.

The technique works by creating artificial instances derived from observed values in order to examine the influence of the features to the output. It assumes binary feature vectors. Starting with the original binary vector x and its predicted label, the algorithm "removes" features from the vector creating an artificial instance $x \quad e$. Features are added to the vector of "removed" features e until the predicted label of $x \quad e$ is different than the one of x. The set $E = \{k \mid e_k = 1\}$ of "removed" features is then called an *explanation* of the original vector x.

Removing in this context indicates the change of a feature that is present to not being present. We chose the term "removing" because it is particularly intuitive for sparse binary data. The technique works only for sparse binary data where it is possible to assign prediction responsibility to components relative to present features. This is the case for bag of words in document classifiers, but also for any kind of data where instances represent a set of items, like medications in the medical domain and products bought or liked in market basket analysis.

The machine learning model is assumed to deterministically compute a prediction score $f(x)$ between 0 and 1 for a given input vector x. The predicted label is then computed by comparing this score to a given optimal threshold. This threshold is computed on the training data minimizing the number of misclassifications.

The process of removing features from a vector is the core of the explanation algorithm. The original algorithm by Martens and Provost [17] consists of successively removing the feature with the highest impact on the prediction score towards the threshold until the label changes. This results in the most compact explanation for a given instance. If the prediction score cannot be changed towards the threshold by removing a feature the instance is discarded without outputting an explanation. To increase the number of explained instances, in order to provide a fuller picture of the model's decisions in our visualization, we relax this restriction by removing random features with no impact on the prediction score. Oftentimes, after removing some features randomly the prediction score starts changing again leading to an explanation for this otherwise discarded instance. However, removing features with no impact on the prediction score violates the compactness property of the resulting explanation. In order to restore this property we add a post-processing step that re-adds features that do not contribute to a favorable prediction score change. This process is time consuming requiring us to pre-compute explanations offline.

Another difference of our approach to the algorithm by Martens and Provost [17] is the handling of explanations that grow too long. Explanations that are long are not intuitively interpretable by a user thus they are discarded in the original algorithm. As we are adding random features, in some cases the explanation might grow too long before the compacting step. In order to not discard explanations that are only temporarily too long we perform the length check after this step.

4 RIVELO: THE EXPLANATION INTERFACE

We implemented the workflow we briefly described in the introduction in an interactive visual interface we call *Rivelo*[1]. The application in its current implementation works exclusively with binary classifiers and binary features and it assumes to receive as an input a data set and a trained classifier. The data set must also contain ground truth information, that is, for each data item what is the correct label the classifier is supposed to predict. Once the system is launched, it automatically computes the following information: one explanation for each data item; information about whether the prediction made by the classifier is correct or incorrect (including false positives and false negatives); the list of features, ranked according to how frequently they appear in the explanations. For each feature, it also computes the ratio between positive and negative labels, that is whether the feature tends to predict more positive or negative outcomes, and the number of errors the classifier makes when predicting items whose explanations contain that feature.

The user interface is made of the following main panels that reflect the steps of the workflow (Figure 3): a *feature list panel* (1, 2) on the left, to show the list of ranked features; the *explanations panel* (3, 4, 5) next to it, to show the explanations and data items containing the selected features; the *descriptors panel* (6), containing a visual representation of the vectors / descriptors of the data items selected in the explanations panel; and the *raw data panel* (7, 8) containing the raw data of selected vectors. We now describe each panel and interaction with them in more details.

[1]Rivelo GitHub repository at: https://github.com/nyuvis/rivelo.

Figure 3: Showing the user interface of *Rivelo*. (1) is a selected feature and (2) are its relative indicators for average prediction, errors and frequency within the explanations set. Some of the features on this list are grayed out because they wouldn't return any explanation if added to the query. (3) is the selected explanation with the number of explained documents and the number of used features. (4) are the cells representing each explained instance, with color resembling the prediction value. Among them, (5) represent some false positive instances. (6) is the descriptor representing visually one of the explained instances. (7) is the raw data related to the same descriptor. (8) shows in bold how the explanation feature is used in the text data.

The **feature list panel** displays the features computed in the beginning and displays the following information in a list: the name of the feature, the ratio of positive labels, the number of errors and the number of explanations it belongs to. The ratio is displayed as a colored dot with shades that go from *blue* to *red* to depict ratios between 0% and 100% true outcomes. The number of errors and the number of explanations are depicted using horizontal bar charts. Users can sort the list according to three main parameters: (1) *number of explanations*, which enables them to focus on importance, that is, how many decisions the classifier actually makes using that feature. (2) *ratio of positive labels*, which enables them to focus on specific sets of decisions, and (3) *number of errors*, which enables them to focus on correctness, that is, what features and instances create most of the problems.

Once one or more features are selected, the **explanations panel** displays all explanations containing the selected features. The explanations are sorted according to the number of explained instances. Each explanation has a group of colored cells next to it ((4) in Figure 3) that represent one instance each. The color of the cell indicates the value of its prediction (blue for positive and red for negative) and different shades depending on the prediction value. When the cell represents an instance that is classified incorrectly it is marked with a small cross, to indicate the error ((5) in Figure 3).

By selecting an explanation or a single cell we can display the explained instances in the **descriptors panel**. The panel displays a list of visual "descriptors", each depicting the vector of values used to represent an instance in the data set ((6) in Figure 3). The descriptor is designed as follows. Each rectangle is split into as many (thin) cells as the number of features in the data set. A cell is colored with a dark shade (small vertical dark segments in the image) when a feature is present in the instance and left empty when it is not present. A cell is colored in green when it represents a feature contained in the explanation. The background color represent the predicted label.

The main use of descriptors is to help the user visually assess the similarity between the selected instances according to which (binary) features they contain. When looking at the list of descriptors, one can at a glance learn how many features are contained in a instance (that is, how many dark cells are in it) and how similar the distribution of features is. Descriptors are particularly useful in the case of instances with sparse features, that is, when the number of features present in a given instance is much smaller than the number of those which are not present.

Next to the vectors panel, the **raw data panel** displays the actual raw data corresponding to the selected descriptors. This is useful to create a "semantic bridge" between the abstract representation used by the classifier and the descriptor representation, and the original data. In our example, the data shown is text from a document collection but different representations can be used in this panel to connect the abstract vector to the actual original data contained

in the data set (e.g., images). In this case we also highlight, in each text snippet, the words that correspond to the features contained in the vector. Similar solutions can be imagined for other data types.

One additional panel of the user interface is accessible on demand to obtain aggregate statistics about the explanation process and the classifier (not shown in the figure). The panel provides several pieces of information including: percentage of explained instances over the total number of instances, number of instances predicted correctly and incorrectly for each label, number of features, number of data instances as well as the prediction threshold used by the classifier to discriminate between positive and negative cases.

5 USE CASE: DOCTOR REVIEWS ON YELP

In this section we present a use case based on the analysis of a text classifier used to predict the rating a user will give to a doctor in Yelp, the popular reviews aggregator. To generate the classifier, we used a collection of 5000 reviews submitted by Yelp users. We first processed the collection using stemming and stop-word removal and then created a binary feature for each term extracted, for a total of 2965 binary features. We also created a label out of the rating field, grouping together reviews with 1 or 2 stars for negative reviews and those with 4 and 5 stars for positive reviews. This way we created 4356 binary vectors, one for each review, of which 3334 (76%) represent positive reviews and the remaining (24%) negative reviews. Then, we trained a random forest model [5] based on the data and labels just described and obtained a classifier with an area under the ROC curve score equal to $AUC = 0.875$. We exported the scorer function of the model, the predicted label and the ground truth label for each review of test data to generate the input for *Rivelo*. We also computed the optimal threshold for the scorer function 0.6. The value is closer to 1 to compensate for the high amount of positive reviews in the data set.

After the first computation of explanations, which took around 1.5 hours to complete, we explained 64% of the reviews with explanations of length up to 5 words. By post-processing, which took about 1.5 hours as well, we were able to reduce the size of 34.5% of the 1563 explanations that were longer than 5 features. This way we increased the number of explained instances by 542, explaining 76.52% of instances of our test set. By compacting long explanations in the post-processing, we are able to explain 12% more instances than the original algorithm by Martens and Provost [17].

Figure 3 shows the results obtained by the entire procedure.

Looking at the set of features sorted by frequency one can readily see that most of the model decisions take place to predict the positive label and that many of the words used capture adjectives that represent positive sentiments such as, "friendly", "love" and "recommend", which have been labeled correctly as positive words. By taking a closer look we also see less obvious words, that do not seem to have straightforward role in the classification, even if they are used more than others in explanations. For example, the word "Dr." is used frequently to explain positive reviews. Using the error bar indicator next to "Dr." ((2) in Figure 3), we can also see that the feature has a very low false positive rate.

When we select this feature, the interface shows all the explanations and instances containing it. We can then see that the large majority of cases is predicted by the word "Dr." alone or a combination of this word with some other positive property such as "great" and "recommend". We can also see that some of the instances are classified incorrectly, especially those in which the explanation contains exclusively the word "Dr.". To better understand this trend, we inspect several raw text documents associated to these instances and figure out that reviewers tend to use the name of the doctor preceded by the word "Dr." whenever they have to say something positive, but they tend to refer to the practitioner as "the doctor", using a more generic terminology, when writing a negative review.

Through this inspection, we can also better understand how the few false positives explained by "Dr." happen. When we look at the raw text of selected false positive cases we see that most of them represent rare cases where the patient is using the word "Dr." without using the doctor name (e.g. "Dr. is very nice but the staff is rude."). The model therefore tends to be confused by outlier cases in which a contradiction of the rule is present.

Another interesting word is the word "call", which, as shown in the figure, tends to predict negative reviews, even though with a somewhat high error rate. While not immediately obvious why this word leads to negative reviews, we figure out, through the visual inspection enabled by our application, that reviewers tend to mention cases in which they have called the doctor's office and received poor assistance. A similar case is the feature "dentist" (not visible in the figure), which also tends to predict negative reviews. The feature however has also a somewhat higher error rate which means many positive reviews are misclassified as negative. Through closer inspection, we realize that the classifier is not able to disambiguate cases in which the word "dentist" is used in a positive context such as "awesome dentist".

The most common word in explanations with hundreds of associated documents is "great". The majority of those documents are true positive, but we can still find and select the few false positive the model generates. Selecting all the misclassified positive reviews containing "great" we spot an interesting problem: some of the reviewers sometime use the word "great" in a sarcastic way, making the detection of a negative connotation too hard for the classifier. Similarly, we also notice that the word "great" is sometime used in conjunction with a negation, that is, "not great", making it once again too hard for the classifier to make the correct prediction with its current configuration. An interesting aspect of this last case is that the manual inspection of misclassified instances can lead to ideas on how the classifier could be improved. For instance, in this last case equipping the classifier with means to detect negation may lead to improved performance.

6 DISCUSSION

Our use case shows how the proposed solution can help figure out major decisions made by the classifier, spot potential issues and possibly also help derive insights on how problems can be solved. The system, however has, in its current implementation, a number of relevant limitations, that we discuss below.

First, *Rivelo* works exclusively with binary classification and binary feature sets. While this specific configuration covers a large set of relevant cases (e.g., we tested the system with a medical data set describing drugs administered to patients in emergency rooms to predict admissions), many other relevant cases are not covered; notably cases in which features or the predicted outcome are not binary. To solve this problem we will need to develop explanations