Bachelorarbeit am Institut für Informatik der Freien Universität Berlin

Human-Centered Computing (HCC), AG NBI

An alternative confusion matrix implementation for PreCall

– Bachelorarbeit –

Emil Milanov

Matrikelnummer: 5002982

emil.milanov@fu-berlin.de

Betreuerin: Prof. Dr. C. Müller-Birn

Erstgutachterin: Prof. Dr. C. Müller-Birn

Zweitgutachter: Prof. Dr. L. Prechelt

Berlin, 14-Juni-2020

Selbstständigkeitserklärung

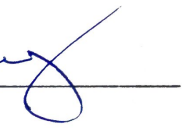
Name: Milanov	(Nur Block- oder Maschinenschrift verwenden.)
Vorname: Emil	
geb.am: 07.08.1996	
Matr.Nr.: 5002982	

Ich erkläre gegenüber der Freien Universität Berlin, dass ich die vorliegende Bachelorarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe.

Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch bei keiner anderen Universität als Prüfungsleistung eingereicht.

Datum: 14.06.2020

Unterschrift: 

Abstract

In this work, we examine literature on creating visualizations for the performance of machine learning classifiers, with our target group being users with limited machine learning experience. The underlying data is taken from Wikipedia, and more specifically ORES - Wikimedia's service, which employs a machine learning model to score edits and articles. The interface also expands on PreCall's implementation, and features multiple interactive components allowing the user to dynamically adjust parameters and see the immediate change in the classifier's performance. After providing a summary of the relevant literature, we go over the ORES API and its relevant endpoints and parameters. Then, we outline the most popular ways to visualize a machine learning classifier's performance. Following that is a thorough description of our target group, goals, and requirements, as well as the reasoning behind each design decision. Finally, there is an overview of the design and development process and we conduct a feedback session with a machine learning expert with background in ORES, and the feedback we receive is mostly positive, with some suggestions for improvement.

Contents

1 Introduction

1.1 Motivation

Wikipedia, the Free Encyclopedia is the most popular online encyclopedia in the world. At the time of writing, Wikipedia is receiving 1.8 edits per second, and there are 572 new articles coming out every day.[1] However, with such a frequency of new edits and articles, the need for some kind of a review process arises.

Understandably, a portion of these new edits are acts of vandalism - "the addition, removal, or modification of the text or other material that is either humorous, nonsensical, a hoax, or that is an offensive, humiliating, or otherwise degrading nature".[2] Since it would be humanly impossible to manually review every single new edit or article published on Wikipedia for possible vandalism, the website relies on automated quality control tools, most of which employ Machine Learning classifiers.[7]

Machine Learning is not a new field in computer science, but it has recently gained a lot of popularity in part because of improved hardware performance. Machine Learning allows us to automate complex tasks without explicitly defining the behaviour of the program, and models show a lot of promise across multiple areas, such as speech recognition, image classification, and playing games. However, mostly machine learning experts have the necessary experience needed to create and use such models. Developing visual interfaces for machine learning applications could be an approach to integrate non-experts in the field of machine learning.[4]

1.2 Context

Detecting and dealing with vandalism on Wikipedia is a task, where the domain knowledge of machine learning non-experts would be useful. ORES was developed to address this.[3] ORES is a web service for Wikimedia projects, which provides an API, allowing access to already trained machine learning models, which are able to rate, for example, whether an edit is an act of vandalism, or grade the overall quality of a Wikipedia article. However, having strict quality control policies for the automated tools can lead to new users feeling discouraged to contribute, because edits they made are being automatically reverted without additional feedback.

Thus, ORES made it part of its goal to help with the integration of new users, for

[1] `https://en.wikipedia.org/wiki/Wikipedia:Statistics`
[2] `https://en.wikipedia.org/wiki/Vandalism_on_Wikipedia`
[3] `https://ores.wikimedia.org/`

example by providing a model, which judges whether an edit was made in good faith. ORES also provides statistics and APIs, encouraging users to explore and experiment with the underlying model.[7] Despite its efforts though, ORES' documentation is somewhat lacking and users with limited machine learning experience could have trouble understanding the terminology and the functionality of the ORES classifiers.

A project which aims to make the ORES API more accessible is PreCall.[6][9] PreCall features a thorough documentation of ORES' vandalism detection model - its parameters, API endpoints and performance statistics. PreCall also features an interactive user interface, which visualizes some of the API's parameters, thus allowing a user with limited machine learning experience to still understand these parameters and the performance of the underlying machine learning model, while the interactive elements allow the user to see how the classifier's performance behaves when adjusting the settings.

1.3 Goal of this research

This research builds heavily on both ORES and PreCall, and could be regarded as an extension of PreCall, expanding on the feedback received during its evaluation. The goal is to create a visual interface, which would allow a developer with limited machine learning experience to understand the performance of ORES, explore the training data set, and pick a suitable setting for their needs. In addition, a central point of this research is not to inconvenience the user with parameters they do not fully understand, and which are not crucial to the adjusting the model's functionality. This would ensure that users need less time and mental effort to learn to understand the interface and its underlying concepts.

1.4 Structure of the thesis

Section 2 provides a concise description of the research used as a basis for this project, as well as a summary of the most popular methods for visualizing and evaluating classifier performance. The subsection for ORES also provides a short explanation of the ORES API - its relevant endpoints, parameters, and expected results. Section 3 provides a thorough explanation of the goals, requirements and design rationales behind this research, as well as the process we carried out in order to make those decisions. Section 4 describes the implementation phase of this project - the approach and the encountered challenges during this phase. Section 5 describes a simple evaluation of the application.

2 Background

The need for automated quality control tools on Wikipedia, and the solutions, which machine learning classifiers could provide, have been highlighted in the Introduction. However, bot developers on Wikipedia rarely have the extensive machine learning experience or the appropriate hardware required to train, test, and deploy their own machine learning models. Thus, there was a need for some kind of tool which would provide machine learning as a service, while remaining usable by users with limited machine learning experience. ORES was developed by Halfaker et al. and was designed to fill the requirements described above.[7]

2.1 ORES

ORES is a web service, providing machine learning as a service for Wikimedia projects.[4] On the technical side, ORES gives access to labelled training data sets, or allows volunteers to help with the expensive task of labelling and categorizing raw data. Furthermore, users can commission custom classifiers, which are in turn deployed at an API endpoint, where anyone can request a scoring of both edits and articles, and get a result in real time. Currently ORES provides 78 classifiers in 37 different languages.[7]

However, the ORES team view their creation as a solution to a socio-technical challenge. Namely, allowing a more open access to machine learning tools for Wikipedia, and encouraging new editors to keep contributing edits, reverts, and new articles. For example, ORES intentionally does not provide any preset or "gold standard", as its goal is to encourage users to inspect the classifiers for themselves. Similarly, some of the available classifiers, such as the *goodfaith* one for instance, gauge whether an edit has been made in good faith. The data could be used to give an encouraging personalised message or a helpful tip, to users, whose edits did not meet Wikipedia's standards, but were still made with good intentions.

2.1.1 API

The ORES API has been described in great detail in the PreCall paper.[9] However, for clarity, the most important endpoints and parameters will be summarized in the current section. Model-specific API calls will be made from the context of the *damaging* model, as it is the main focus of this research. More API calls, which won't be summarized here, can be found on ORES' homepage.[5]

[4] `https://www.mediawiki.org/wiki/ORES`
[5] `https://ores.wikimedia.org/v3#/scoring`

Classifiers

While ORES provides multiple classifiers, this research focuses on the *damaging* model - a probabilistic classifier, which gauges whether an edit is an act of vandalism. It's a fairly uncomplicated classifier, which takes a single or a batch of edits as an input and for each edit returns a probability that the edit causes damage to Wikipedia.[6]

Scoring edits

The primary way to use ORES would be for users to submit edits or articles to be scored according to a certain model. Since Wikipedia gets multiple new edits per second, ORES supports receiving batches of edits, and tries to keep a minimal response time.

Scoring a single edit is relatively simple - the user can set which *wiki* and which *model* they want to use. In addition they can specify which *edit* they want classified by entering its ID, as found on Wikipedia. Below you can find the API call syntax and an example call.

$$https://ores.wikimedia.org/v3/scores/\ wiki\ /\ edit\ /\ model$$

$$https://ores.wikimedia.org/v3/scores/enwiki/56782332/damaging\ [7]$$

Scoring multiple edits requires a somewhat different request structure. The model must be specified first, and then one enters the edits to be scored by separating them with a "|" character. Below is an example request to score two edits.

$$https://ores.wikimedia.org/v3/scores/enwiki?models=damaging\&revids=6123521|9125123\ [8]$$

[6] `https://meta.wikimedia.org/wiki/Objective_Revision_Evaluation_Service/damaging`
[7] Clickable link: `https://ores.wikimedia.org/v3/scores/enwiki/56782332/damaging`
[8] Clickable link: `https://ores.wikimedia.org/v3/scores/enwiki?models=damaging&revids=6123521|9125123`

Classifier Performance

Since the process of scoring a revision consist of assigning it a probability that it is damaging, we need to pick a concrete threshold value, such that when it is exceeded, we can classify the edit as damaging. One of the ways to do that would be to look at ORES API endpoint for classifier performance statistics. The API returns a JSON object with different threshold values, and multiple evaluation metrics for each threshold value. The API call looks like this and listing 1 shows an example response. The parameters one can replace are the following:

$$https://ores.wikimedia.org/v3/scores/wiki/?models=models/\&model\_info=statistics.metric$$

$$https://ores.wikimedia.org/v3/scores/enwiki/?models=damaging\&model\_info=statistics.thresholds$$ [9]

One can replace *wiki* with "enwiki" for the English language Wikipedia, or Wikipedia in any other language. The user can point to a single *model* to be evaluated, but can also input multiple models by separating them with a "|". Finally, *metric* is a placeholder for an evaluation metric one would like to see statistics for. This project uses exclusively "threshold", but for other option please refer to the next section, titled "Evaluation metrics".

The API request returns a list of multiple threshold values with a step of 0.001. Each threshold has a corresponding set of performance (evaluation) metrics. The evaluation metrics do not directly contain the number and distribution of confusion classes such as percentage of true positives however. Instead, the response consists of metrics, derivative of the confusion classes.

[9] Clickable link: `https://ores.wikimedia.org/v3/scores/enwiki/?models=damaging&model_info=statistics.thresholds`

```
"enwiki": {
    "models": {
      "damaging": {
        "statistics": {
          "thresholds": {
            "true": [
              ...
              {
                "!f1": 0.939,
                "!precision": 0.992,
                "!recall": 0.892,
                "accuracy": 0.888,
                "f1": 0.324,
                "filter_rate": 0.868,
                "fpr": 0.108,
                "match_rate": 0.132,
                "precision": 0.204,
                "recall": 0.785,
                "threshold": 0.267
              },
              ...
```

Listing 1: Example JSON repsonse from ORES showing the evaluation metrics for threshold = 0.267

Evaluation metrics

The metrics returned by the ORES API are important for evaluating the effectiveness of the model with its current threshold settings. However, people with limited machine learning knowledge will have a difficult time understanding what the metrics represent. The most relevant evaluation metrics are shortly described below. For a more in-depth description please refer to the PreCall paper.[6]

- *Precision* represents the percentage of correctly classified damaging revisions of all performed classifications. In essence, the precision is a metric of the ability of the classifier to avoid false positives. The *recall* represents the percentage of correctly classified damaging edits, from all real world damaging edits. Thus, the recall is a metric for the model's ability to avoid false negatives. This paper describes these two metrics in greater detail in section 2.2.3.

- The *accuracy* metric represents the model's ability to correctly classify both damaging and good edits, in contrast to precision and recall, which measure only the ability to classify damaging edits.

- *Filter_ rate* and *match_ rate* show the percentage of samples predicted to be positive and negative respectively.

- The false positive rate, or *fpr* for short, is the probability that the model will incorrectly classify a sample as damaging.

Optimization query

Helping users find a suitable threshold setting for their use-cases is one of ORES' primary goals. Thus they are providing an API endpoint, which allows users to specify evaluation metrics and their desired values in the form of a condition. ORES will then mathematically compute the threshold, which satisfies these conditions best. An optimization query looks the following way: [10]

$$https://ores.wikimedia.org/v3/scores/\ wiki\ /?models=\ model$$
$$\&model\_info=statistics.thresholds.true.\%27\ minimum/maximum\ \%20$$
$$metric\_1\ \%20@\%20\ metric\_2\ \%20\ <=\ /\ >=\ \%20\ metric2\_value\ \%27$$

[10]The percentages are URL encodings for white space and single quotation marks, so the final part of the optimization query looks like 'Maximum recall @ precision > 0.95'

- *wiki* - Fetch statistics for which Wikipedia. Example: "enwiki" refers to the English Wikipedia.
- *model* - Choose a model, or specify multiple, and separate them with a "|".
- *mininum/maximum* - Whether to have a the lowest possible or highest possible value of *metric_1*.
- *metric_1* - The metric to maximize or minimize.
- *metric_2* - A second metric which to keep above or below a certain value.
- <=/>= - Upper or lower bound of metric_2
- *metric_2_value* - Choose a value between 0 and 1 to keep *metric_2* above or under.

One can verbosely express an example API call as "**Maximize** the **recall**, while keeping the **precision above 0.95**" This query for instance will give us such a threshold setting, that will make sure that we catch the most possible acts of vandalism, while keeping the number of false positives low. [11]

[11] An example request could look like this:
`https://ores.wikimedia.org/v3/scores/enwiki/?models=damaging&model_info=statistics.thresholds.true.'maximumrecall@precision>=0.95'`

2.2 Visualizing Classification Error

One could describe the tuning of Machine Learning classifiers as an iterative process. The process usually involves adjusting the classifier's hyper parameters, feeding it the training data, and finally evaluating its performance using a separate dataset. These steps are often repeated multiple times until the desired performance is achieved. The most well-established visualizations are Confusion Matrices, ROC curves and Precision-Recall curves and they all fulfil different requirements and visualize different aspects of the classifiers.[5][8] This section summarises the most important features of these performance visualization methods.

However, similarly to Machine Learning as a whole, mostly machine learning experts are able to fully understand the information given by an evaluation metric, and also understand the implicit biases of the chosen performance visualisation method. Meanwhile end users with limited machine learning experience often do not "fully understand the errors and their implications" and may even "mistrust or misuse classifiers"[2] There are two additional performance visualisation methods described in this section, who were specifically created to be easily understood by end-users with limited machine learning experience.

2.2.1 Confusion Matrix

One of the most common ways to visualize the performance of classifiers is the so called Confusion Matrix. The Confusion Matrix uses a table layout with a number of rows and columns corresponding to the number of classes the classifier uses. The rows represent the distribution of the predicted classes from the classifier, while the columns represent the actual real world distribution.[11]

For instance, a confusion matrix visualising a binary classifier's performance would have two rows and columns. Typically the rows would be "positive" and "negative", representing the classes as assigned by the classifier. The columns in turn represent whether these predictions are "true" or "false". Thus, the true positives and true negatives are the correctly predicted classes, while the false positives and false negatives represent, as the name suggest, incorrectly classified samples. When evaluating a model's performance, the lower the percent of false positives and false negatives, the better the model.

		Assigned Class		
		A	*B*	*C*
Actual Class	*A*	10	2	1
	B	0	6	1
	C	0	3	8

Figure 1: A confusion matrix for a three-class problem. Retrieved from the Encyclopedia of Machine Learning.[11]

A Confusion Matrix is a great tool for visualizing the performance of binary and multi-class classifiers (see example on figure 1), as it allows the user to see which classes the model is having trouble differentiating. However, confusion matrices only help evaluate a single configuration state of the model. Furthermore, for binary classifiers, end-users with limited knowledge in machine learning might have trouble understanding the meaning behind the class labels (such as "true positives" and "false negatives").[2]

2.2.2 ROC Curve

Many classifiers (*probabilistic classifiers*) assign a probability to each sample that the sample belongs to a certain class. The probability is usually a numeric value, and the higher that value is, the more certain it is that the sample is a member of a class. By specifying a threshold value, one could use a probabilistic classifier as a binary classifier. Say we have classes X and Y and we are calculating a probability that a sample belongs

to X. We also set a threshold of 0.75. Then, if the probability assigned to a sample is higher than 0.75, then the sample is assigned to class X, otherwise it's Y.

Modelling the performance of such a classifier with a confusion matrix would be inefficient, as the confusion matrix could show the performance only for a specific threshold value. A far better visualization, which would also convey the change of performance for various threshold values would be a so called ROC Curve (receiver operating characteristic curve).

This type of visualization is a two-dimensional graphical plot, where the X and Y axes represent the false-positive and true-positive rates respectively. In addition there is a line plot, which denotes the intersection of these two rates. (See fig. 2) For instance, a classifier might have a true-positive rate of 0.8 and a false-positive rate of 0.5 for a certain threshold setting. That means that an actual "positive" sample has an 80% chance of being correctly classified as "positive", while an actual "negative" sample has a 50% chance of being falsely classified as "positive". Depending on the task, such a performance is not ideal. It is generally valid, that the closer a ROC curve lies to the top-left corner of the plot, the better the classifier performance is.

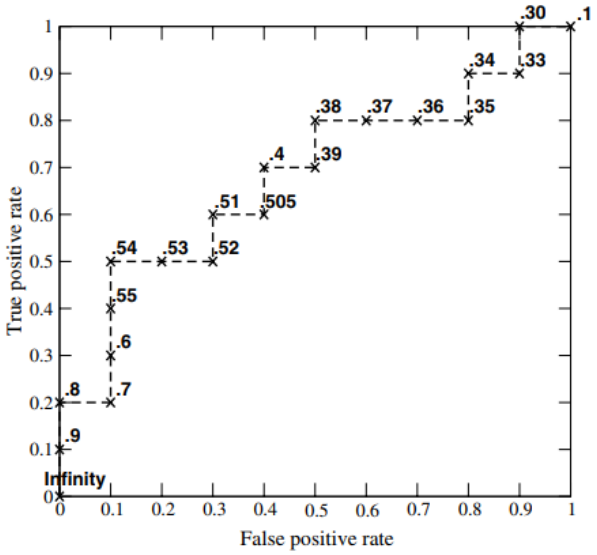


Figure 2: An example of an ROC curve. Each point has its corresponding threshold setting as a label next to it.[5]

An advantage of ROC Curves is that they provide information on the classifier's performance which is not dependent on the number of class instances in the test dataset. For instance if one were to add 1000 "positive" new data points in the test set, without adding any new "negative" ones, a confusion matrix will change, but an ROC curve will not.[8]

2.2.3 Precision-Recall Curve

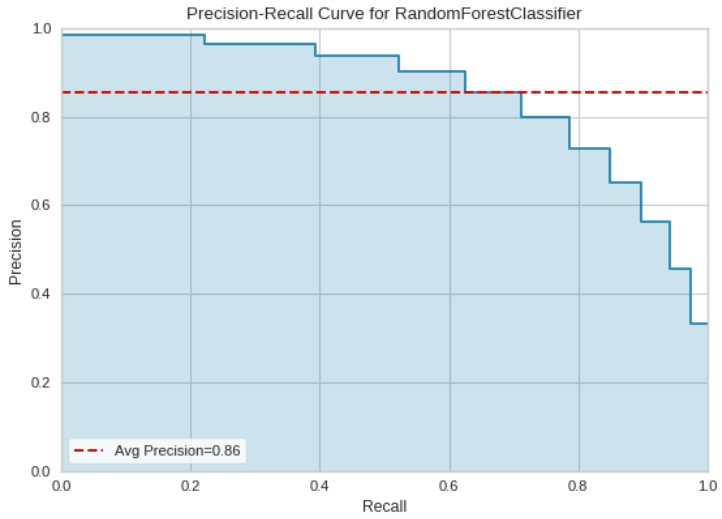


Figure 3: An example precision-recall curve by the python machine learning visualization library Yellowbrick.[12]It visualizes the typical trade-off, where as the recall increases, the precision falls.

The Precision-Recall curve has the same structure as the ROC curve, but it portrays the relationship between other metrics. In this case, both precision and recall are metrics which capture the ratio of the correctly predicted classes (true positives) to other classes. For instance, precision is the ratio of the true positives to the total number of classes the classifier assigned as "positive" (true positives and false positives). Recall on the other hand is the ratio of true positives to the real world positive classes (true positives and false negatives).

$$precision = \frac{tp}{tp + fp} \qquad recall = \frac{tp}{tp + fn}$$

Precision-Recall curves have a similar advantage as ROC curves - namely, they can visualise classifier performance for the whole threshold range, while remaining unchanged when adding more samples to the test data set. The difference to ROC curves though, is that precision-recall curves are more informative when one has an unbalanced dataset - few positive and many negative classes for example.[10] When evaluating a model with such a curve, the closer the curve is to the top-right corner, the better the model. However both ROC and Precision-Recall curves have the disadvantage that they are very difficult for non-experts to understand, as they feature domain-specific metrics, and the threshold plot is implicit.[2]

[12]`https://www.scikit-yb.org/en/latest/api/classifier/prcurve.html`

2.2.4 PreCall

An attempt to make classifier performance evaluation easier for non-experts has been made in the PreCall project. PreCall is a visual interface for ORES, developed by Tom Gülenman and published by Christoph Kinkeldey.[9] The goal of the project is to visualize the performance of ORES "damaging" model, with the addition that the visualization should be interactive and easy to understand by people with limited machine learning experience.

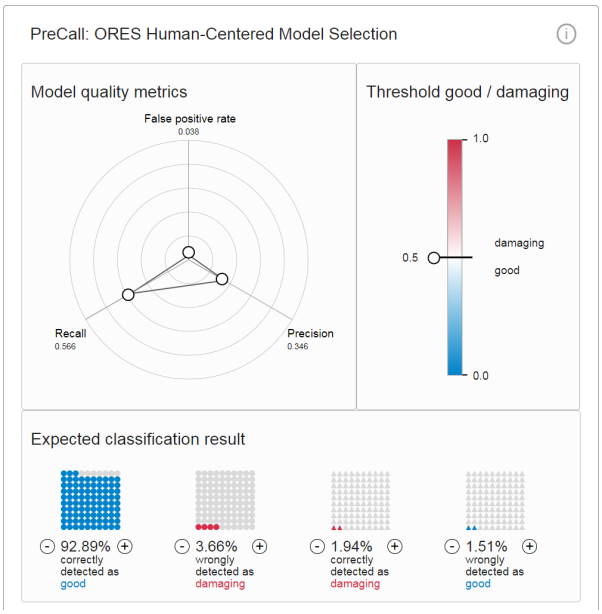


Figure 4: PreCall's visual interface[9]

The interface features multiple interconnected interactive components - when the user interacts with one of them, all others change accordingly. The main interaction takes place in the radar chart and the threshold bar. In the radar chart, the user has access to a few model quality metrics, where they can adjust the precision, recall, or false positive rate. The functionality of the threshold bar is simple - it's a slider where the user can choose a desired threshold, for which to visualize the classifier's performance.

Finally, on the bottom side of the interface there is an alternative visualization of a confusion matrix. The labels such as "true positives" and "false negatives" are named in such a more intuitive way. The percentages are also represented by blue or red-coloured icons. The color represents the classifier predictions, while the shapes represent the actual classes. However, in PreCall's evaluation, users described this visualization as "confusing".

PreCall is important for this work, as the paper publication includes a thorough documentation of ORES - API endpoints, parameters required and metrics returned.

Furthermore, this work will reuse parts of PreCall's code base, in order to speed up development.

2.2.5 CLASSEE

CLASSEE is another project developed in the context of making visualizations to convey classifier performance to users with limited machine learning experience. In their paper, Beauxis et al. test multiple classifier performance visualizations among machine learning experts, math experts, and non-experts.[2] The research found, that confusion matrices and ROC curves are more difficult for end-users to understand. Instead Beauxis et al. propose a bar chart histogram as an alternative. (See fig.5)

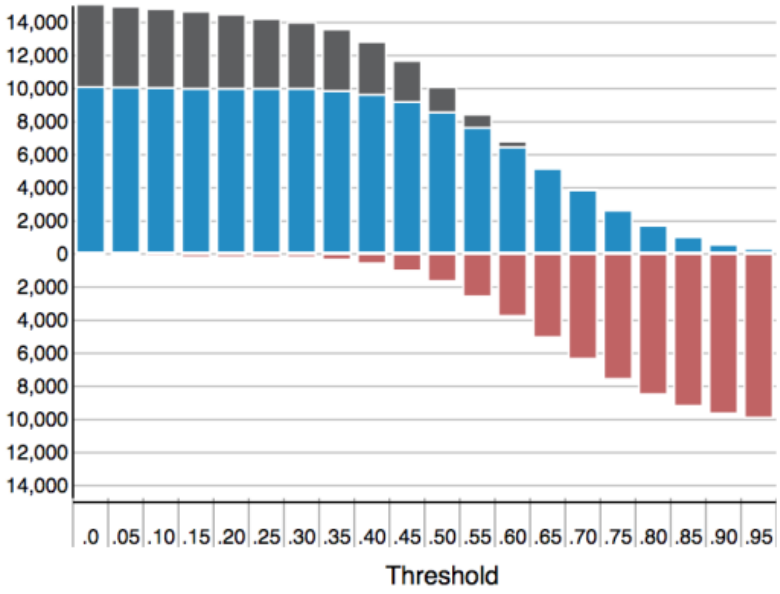


Figure 5: A bar chart histogram for classifier performance.[2]

The bars represent the class distribution, where each predicted class has a different color, and the positive classes are above the zero-line, and the negative classes - below it. Depending on the classifier, true negatives can often be excluded from the visualization, as they are rarely relevant and would just clutter the visualization.[2]

This work often references the CLASSEE paper, as the insights how end-users interact with classifier performance visualizations, and what they understand or don't understand are important for the current research. Furthermore, for the development of this interface we used the bar-chart histogram from CLASSEE as one of the main components.

3 Conception

This section describes the process before the development phase of the interface. It identifies the type of audience that will use the interface, it describes the goals the interface wants to achieve and specifies the requirements. The design section in turn describes the iterations of the planning process, as well as the rationales behind every decision taken during it.

3.1 Goals

Identifying the target audience is a key moment of a project's design phase. When it comes to machine learning classifiers for Wikipedia, this project builds both on the technology and philosophy of ORES. ORES has been designed to "push even further on the crucial issue of who is able to participate in the development and use of advanced technologies" and has been built to support "an open-ended set of community efforts to re-imagine what machine learning in Wikipedia is, and who is it for"[7]. Thus both PreCall and this research are targeting the same audience. Kinkeldey et al. describe them as volunteers, who "do not have deep enough technical expertise in machine learning terms and practices, and therefore, they lack the expertise to develop the machine prediction models necessary to power quality control tools"[9].

Having a target audience in mind makes specifying the goals much easier. We want to create a visual interface, which will help users (non-technical experts) to better understand the functionality of machine learning classifiers, and more specifically, ORES' "damaging" model. However, we want to adhere to ORES' philosophy of an open-ended approach and want to give the user freedom to explore the data set and settings for themselves without providing any presets or default settings.

Since access to this interface will be provided as an in-person service, we expect users to come to find a threshold setting for their model once or mostly twice, and we cannot expect them to spend longer than one or two hours. Thus the interface needs to be easy to understand and use from the first time, so that the users feel comfortable using it as quickly as possible.

The interface needs to be interactive. We believe that after observing the relationships between different components and ORES metrics, the user can get a better grasp on how the classifier work and how different parameters affect the classifier's performance.

Finally, while as a result of the first requirement the interface will be lacking many domain-specific words or visualizations, it is still important than we don't omit all of them. We expect that the user would want to visit multiple services or try alternative

ways to find a suitable threshold for their use-case, and it is most certain that they will encounter language and concepts, specific to the domain of machine learning. Thus, our interface needs to help the user learn basic domain-specific concepts.

3.2 Design rationales

Having defined a set of requirements the interface needs to fulfil, helps us delve deeper in the design phase of the project, and make decisions on how the interface is going to look like and what its functionality will be. The requirements are needed in order to both guide and constrain every single decision we take. Reaching the design rationales described in this section was an iterative process. We started with a "low-fidelity prototype" of the interface, which we reworked multiple times, analysing at every step what works, what could be better, and what should be abandoned.

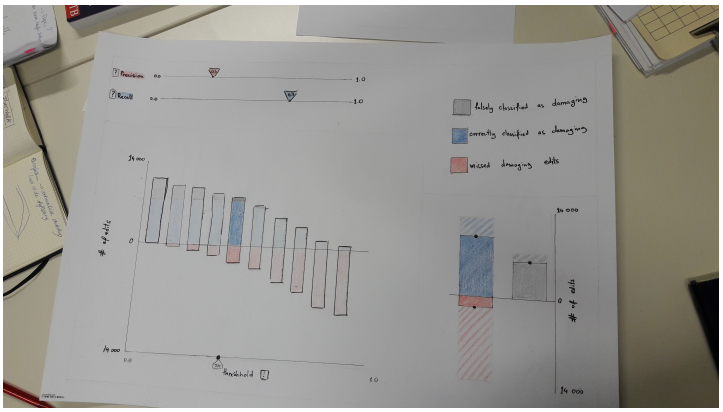


Figure 6: Initial prototype design, featuring a histogram, selector view, and precision and recall sliders.

Figure 6 represents the initial design of the interface. The focal point is the histogram, as proposed by CLASSEE. To the right is a bar chart representation of a confusion matrix for the current threshold setting. The sliders are interactive, so the user can drag them and adjust the desired confusion distribution. The top of the interface features a precision and a recall slider, as inspired by PreCall.

The feedback from this iteration was that the precision and recall sliders are excessive. They are too similar to PreCall and at the same time they are too domain specific for technical non-experts to understand. A proposed histogram improvement was that the bars be connected, in order to show the user that they are part of the same data space. An additional interesting idea was to implement gradients, which would signify the concentration of edits between two threshold values (e.g. 0.7 and 0.8). All of these issues were addressed in the second iteration (Figure 7)

After discussing the second prototype, some further improvements were required. We decided to opt for more consistency, thus the histogram bars were to be split in two groups - true positives stacked on top of false negatives, and false positives stacked on top of true negatives. This would ensure a consistent look with the selector view. At this point we also decided to add two more components - the confusion matrix and the

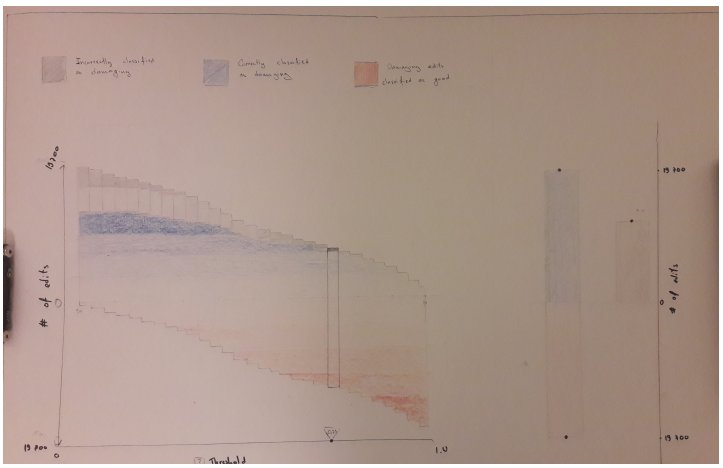


Figure 7: Second prototype design.

edit viewer. (The rationales behind these components are described in section 3.2.2). The layout of these components also played an important part in the discussions. It was decided that the histogram would not be the main interaction point, and instead the selector view should be the focal point of the interface. Figure 8 shows the final interface layout.

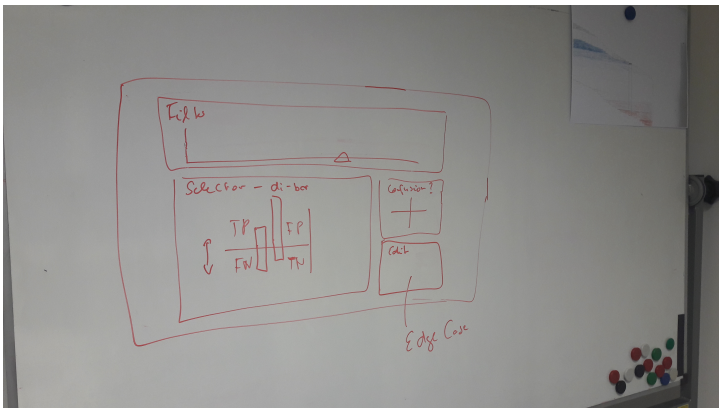


Figure 8: The final prototype layout. Featuring the selector view as the focal point, and the addition of a standard confusion matrix and an edit viewer.

3.2.1 General decisions

Domain specific concepts

PreCall's interface featured components, where the user could adjust the desired precision and recall settings. However, we decided to keep the parameters which the user has access to to a minimum. The main reason for this is that the feedback from one

of the prototypes was that these parameters are domain-specific and users with limited machine learning knowledge might need a lot of time to understand what these metrics represent. Similar measures we can take to reduce the domain-specific concepts the user encounters is to rename the model's classes in a more intuitively understandable way. For instance "true positives" could be replaced by "edits correctly classified as damaging".

True negatives

When using a bar chart to visualize classifier performance, such as the histogram or the selector view (See section 3.2.2), one can decide to not display the true negatives, as they are often not especially relevant for the performance evaluation, and can clutter the interface.[2] We tried that in one of the iterations of the low-fidelity prototype. However, since we have technical non-experts as our target group, we need to make sure that the interface provides all the relevant information to them. It would be wrong to make the assumption that the users will find the true negative bar irrelevant. Thus, we have decided to intentionally keep it, even though it might clutter the interface somewhat.

3.2.2 Components

Having the requirements in mind, we decided on four interactive components to be part of the final interface design: A **histogram**, based on the CLASSEE project by Beauxis et al.[2], a **selector view**, which would be a close-up of a certain threshold setting, a standard **confusion matrix**, and an **edit viewer**, which would showcase concrete edits.

The histogram and selector view were designed as a pair - the histogram visualizes the "big picture", while the selector view showcases the current setting, and initially both of them were supposed to be the main interactions of the interface. However, after a few iterations, we discovered, that the interaction the histogram provides is rather limited, so the focal point of the interface should be the selector view.

Selector view

The selector view is very similar to a confusion matrix, in the sense that it shows the distribution of true positives, false positives, true negatives and false negatives for a certain threshold value. However the visualizations resembles that of a bar-chart - each metric is visualised by a bar. More specifically, the bars would be stacked in groups of two, where the top bars would grow upwards and the bottom would grow downwards. Initially the idea was that every bar in the selector view would have a handle, so that the user can adjust whether they want more or less of that certain class. However, after testing this approach we wanted to make it clear to the user that there is a special

relationship between certain confusion classes - namely the sum of true positives plus false negatives, and false positives plus true negatives are constant. Thus, we decided that instead of dragging each bar to make adjustments, the user can instead drag the "zero-line", so that the bar remains constant, just the colours distribution changes. Both bars' handles should be connected by a line, in order to convey to the user that changing the value of one of them, will also update the other.

Histogram

Based on the research by Beauxis et al.[2] which concluded that end-users are most comfortable with understanding classifier error via a bar-chart, we decided to include one in our interface. The histogram would show the confusion distribution for the whole range of threshold values available. Figure 5 represents the initial design of the histogram we chose. However, after the decision to include a Confusion Matrix, we decided that we should keep our visuals consistent. Thus, the histogram should resemble both the selector view and confusion matrix more closely, so the false positives won't be stacked on top of the true positives, but would rather be on the side parallel to them. (See figure 9)

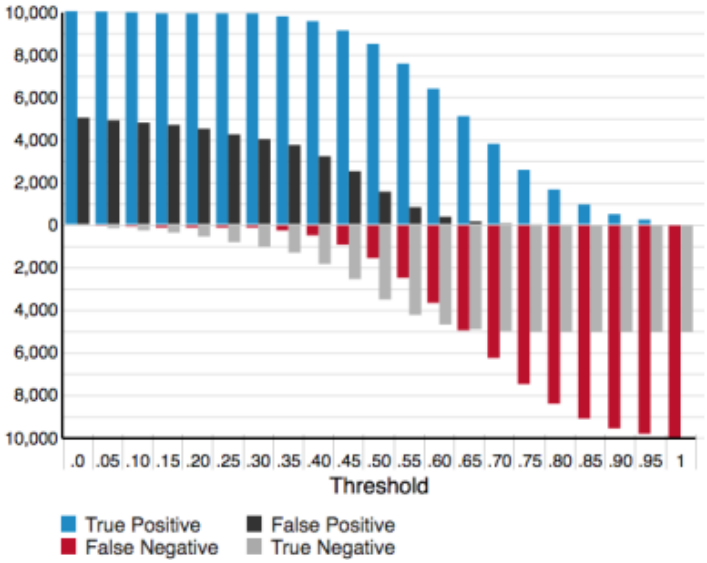


Figure 9: From the different histogram designs which CLASSEE provided, we this is the one we chose.[2]

The histogram would also include a vertical bar, which would show which setting is currently selected, and would also be draggable so that the user can select a desired threshold setting in that view.

There was a lot of discussion regarding the position of the histogram on the interface. The consensus was that the histogram is supposed to function as a map of sorts, so it should always be in view, but it shouldn't be the focal point. The best way to achieve this would be to place the histogram on top of the page, taking the whole width.

Confusion Matrix

Initially we thought of explicitly not including a confusion matrix, as they require some domain knowledge to be understood, which we specifically wanted to avoid. However, it occurred to us, that even if the user has limited machine learning experience, almost all ways to visualize classifier performance used in other services would feature some domain-specific visualization style. Thus we decided that we wanted to help users learn some basic domain-specific concepts and terminology.

Since confusion matrices are easier for end-users to understand in comparison to ROC curves,[2] we decided to include one in the interface. The confusion matrix will also feature some interaction, in the form of buttons for each class, where the user can either increase or decrease the percentage of samples they want in the current class.

Edit viewer

The rationale behind the final component is that thus far all of our components convey information through numbers and relations of numbers. However, threshold settings and true positive rates could be abstract, so we needed something which the user could relate to the real world. Thus, the decision to include an edit browser, where the user could search for real edits with a desired probability to be damaging.

However, the idea of manually searching for edits and thresholds seemed way too complex, and we also thought it would be more interesting to display edge cases - damaging edits, which when the user increases the threshold a bit would be classified as harmless edits. We believe that this component would help the user understand the significance of the difference between various threshold values.

4 Prototype

This section describes the structure of the prototype, as well as the decisions taken and challenges encountered during the implementation phase. The section should also serve as a simple documentation of the project, describing important objects and functionality.

PreCall was developed as a Javascript React[13] prototype. Since this project is intended as a successor of PreCall, it was developed in React as well and it reuses a portion of PreCall's code base. React is a Javascript framework, built specifically for the task of developing user interfaces. Since Javascript programs run in the browser, React allows users to build platform-independent applications. React also renders visual components asynchronously and uses the concept of "state", where it only re-renders components whose "state" data structure has changed. This is an extremely useful feature, especially for a project like PreCall and its successor, where there are interconnected visual elements, with expensive data processing function behind them. The source code of the prototype can be found on Github.[14]

Notation

The development process will be described component-wise. React **Components** will be in bold text, *functions()* will be in italic, and states and props will be underlined.

[13] `https://reactjs.org/`
[14] `https://github.com/emomicrowave/PreCall`

4.1 Structure

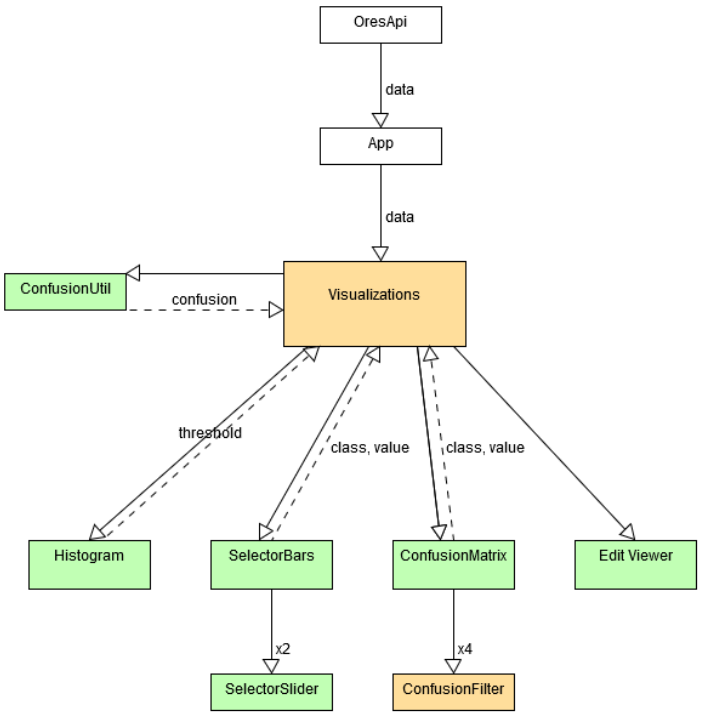


Figure 10: Structure of this project. Components in black were reused from PreCall. Components in orange were taken from PreCall and modified, whereas components in green are new. The dashed arrows show what kind of values does each component update.

Figure 10 shows the object structure of this program, and the color-coding represents whether a component was taken directly from PreCall, modified, or implemented from scratch. Top-level components taking care of the initial API call (**OresApi**) and the Application wrapper (**App**) were unchanged, as this project uses the threshold statistics from ORES' "damaging" model.

The **Visualizations** class acts both as a graphical container for rendering the components, as well as a "manager" of sorts, in the sense that it processes the data from the currently manipulated component, and updates all others. The visual components are a **Histogram**, a **Selector View**[15], a **Confusion Matrix**, and an **Edit viewer** (all of which are described in section 4.2.2).

ConfusionUtil however is not a visual component, but more of a helper class, that stores and calculates confusion values. Its functionality is described in greater detail in the following section.

[15]Corresponds to *SelectorBars.js* in the source code

4.2 Implementation

4.2.1 Backend Functionality

OresApi

Described as PreCall's power source,[9] this class performs a single GET request to ORES to obtain the statistical data for the "damaging" model. It is important to mention that the data is fetched only once on start up, and is then passed directly to the **Visualizations** component as an array of thresholds and their corresponding evaluation metrics (precision, recall, etc.). It wasn't necessary to change this component's implementation, as we are working with the same ORES data as PreCall.

Visualizations

In React, when updating the state of a component, the program will also update the state of all child components and re-render them on the screen. Therefore, if you have multiple components which should be updated at the same time, the best practice would be to implement them as children of the same parent component: **Visualizations** in this case.

PreCall's implementation featured a *setNewValues()* function, which when given a single metric and its value (precision, recall or threshold), would use it as an index to find its corresponding data points in the data array, and would update the state, so that all components would re-render. However, since in this project multiple components are requesting values for true positives or true negatives, which are not explicitly provided by the ORES API, a somewhat different function was required, which would also search all the possible confusion distributions.

The function is called *updateEverything()* and uses the **ConfusionUtil** component to find suitable confusion distributions in order to update the state of all components. Updating the state of a React component will trigger the re-rendering of the component itself, which will, by extent, update the rendering of all the child components.

Since the **Visualizations** component is already instantiated in **App**, and that's the instance that has the relevant state, creating new instances in the child-components wouldn't be a good practice. Instead access to the *updateEverything()* function is provided as a callback property to each child component.

ConfusionUtil

Almost every component requests confusion data of some sorts. The **Histogram** requires the confusion matrix for each threshold value, and the **SelectorView** and **ConfusionMatrix** display the classifier performance for the current threshold setting. That is why there was a need for a separate component which would handle calculating and storing all confusion values.

Many functions in this Component were reused from PreCall's **ConfusionFilter** class. However, access to the component functions needed to be made public, in order for multiple components to be able to access them. When calling the class' constructor, the program iterates over each possible threshold and computes the corresponding positives and negatives. This is done in the *calculateConfusion()* function (see Listing 2).

```
1  // "data" variable contains the threshold statistics from the ORES API
2  calculateConfusion(data) {
3      for (let i = 0; i < data.length; i++) {
4
5          //save threshold
6          this.thresholds.push(data[i].threshold);
7
8          //necessary constants
9          const filters = this.round(100 * data[i]['filter_rate']);
10         const matches = this.round(100 * data[i]['match_rate']);
11
12         //calculate confusion values
13         const tp = this.round(matches * data[i]['precision']);
14         const fp = this.round(matches - tp);
15         const tn = this.round(filters * data[i]['!precision']);
16         const fn = this.round(filters - tn);
17
18         //fill arrays
19         this.allTPs.push(tp);
20         this.allFPs.push(fp);
21         this.allTNs.push(tn);
22         this.allFNs.push(fn);
23     }
24 }
```

Listing 2: calculateConfusion()

The most important function in this class is *setConfusion*, which when given a confusion class (e.g "true positives") and a value, would search the array of all possible class values for the closest match, and would return the index (See Listing 3). This function is called by *updateEverything* in **Visualizations** every time a component requests a change in the confusion distribution.

```
1  setConfusion(confusionValue, wantedValue) {
2
```

```
3       //get the full array of TPs, FPs, TNs or FNs from state, depending
            on what Button has been clicked
4       let fullArray = [];
5       if (confusionValue === 'TP')
6           fullArray = this.allTPs;
7       else if (confusionValue === 'TN')
8           fullArray = this.allTNs;
9       else if (confusionValue === 'FP')
10          fullArray = this.allFPs;
11      else if (confusionValue === 'FN')
12          fullArray = this.allFNs;
13
14      let wantedIndex;
15
16      if (this.isNumber(wantedValue)) {
17          //user passed value in %
18          let closest = Infinity;
19          wantedIndex = 0;
20
21          //find index of closest existing value to the one specified by
                user
22          for (let i = 0; i < fullArray.length; i++) {
23              if (Math.abs(wantedValue - fullArray[i]) < Math.abs(
                    wantedValue - closest)) {
24                  closest = fullArray[i];
25                  wantedIndex = i
26              }
27          }
28      } else {
29          return
30      }
31      return wantedIndex;
32  }
```

Listing 3: setConfusion()

4.2.2 GUI Description

Histogram

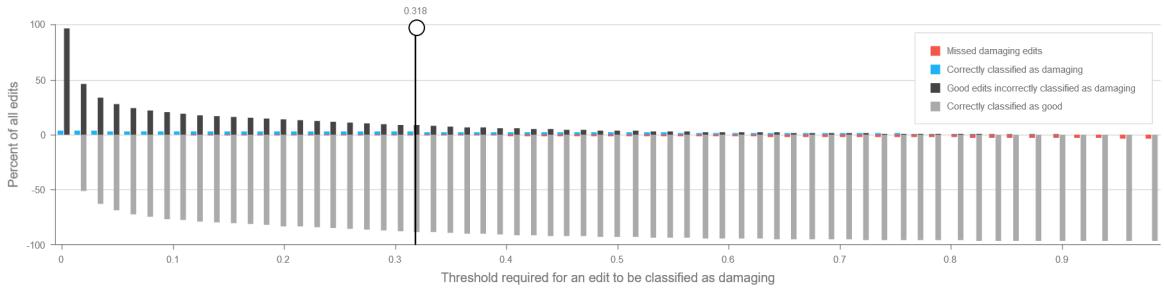


Figure 11: The final version of the histogram, including a threshold slider.

The first and top-most component is a **Histogram**. The design rationale behind this component was to use it as a map of sorts, that would show the user the current threshold setting and provide an overview of the possible confusion distributions over the whole dataset.

The visual style of the histogram has been achieved with the pre-made component called "Grouped stacked histogram", provided by the *DevExtreme* framework.[16] DevExtreme is a huge suite of widgets for responsive UI design and provides various pre-made visual designs, as well as the necessary backend functionality to process the visualized data. From the multiple libraries or frameworks which provided pre-made Histograms, DevExtreme's implementation was easiest to use, while providing most control.

The data for the histogram is processed once at the start of the program, and consists of all possible thresholds and their corresponding confusion classes. In order to achieve the look of the CLASSEE Histograms, the values of false and true negatives have been multiplied by -1. This way, the bars representing both values would be stacked under the true and false positives respectively and would be pointing downwards.

Plotting all of the 1000 possible thresholds would heavily clutter the interface, that's why the **Histogram** has the reduce property. Provided with the numeric value of n, the histogram will only draw every n-th element, making the visualization much more tidy.

In order to visualize the current setting, the Histogram features a vertical bar, which reflects the current position in the dataset (current threshold value), but can also be dragged, so that the user can explicitly change the threshold from there. Since the bar

[16] `https://js.devexpress.com/Documentation`

is a **ConstantLine** component directly provided by DevExtreme and cannot be dragged itself, to achieve the dragging effect we use the *react-draggable*[17] component to create a handle which the user can interact with.

In order for the handle to be able to influence the threshold and vice versa, we needed to manually track and adjust the position of the handle. We are also keeping track of whether the react component's *render()* method is called from an internal or external state update - because we need to also change the threshold handle's position on an external state update.

Selector view

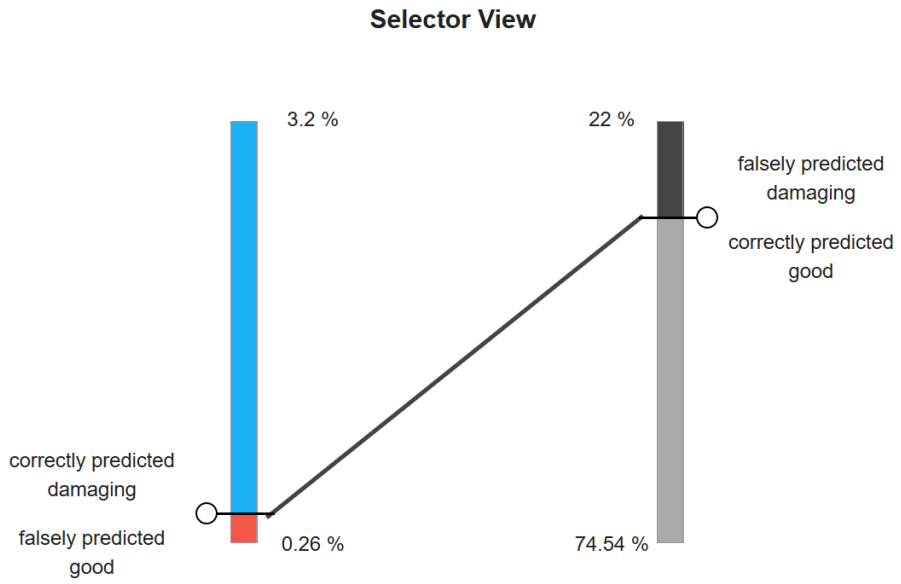


Figure 12: Two selector sliders allowing the user to directly adjust the current confusion.

The selector view features two instances of the **Selector Slider** components, and thus represents an alternative confusion matrix visualization, inspired by CLASSEE.[2] The selector sliders are essentialy the "ThresholBar" from PreCall, however the visuals and the source data have been adjusted for the needs of this project. Since in the design rationales we decided that we needed to highlight the constant relationship between true positives and false negatives, and false positives and true negatives, instead of dragging the columns to set new values, the user should drag the difference between them. This functionality could easily be implemented by two sliders.

[17] `https://github.com/STRML/react-draggable`

The **Selector view** acts as a wrapper for the two sliders, and features a simple *update()* function, which registers which slider was just updated, and sends the appropriate metric and value to the *updateEverything()* function in **Visualizations**. The domain of the sliders is calculated once on start up, where the left slider allows values between zero and the sum of true positives and false negatives. Similarly the right slider allows values between 0 and the sum of false positives and true negatives. Since slider implementations can usually store a single value, the left slider stores the true positives, and the right - the false positives. However, all the confusion classes can correctly be displayed by subtracting the current slider value from the highest possible value allowed.

The two sliders are instances of **Selector Slider**, which in turn is a modified version of PreCall's **Threshold slider**. However, since we have two instances, we needed some way of setting different colors and positions for both sliders. Initially the idea was to provide them as properties. However, this approach would require to significantly alter the code provided by PreCall's implementation, thus risking introducing bugs or even breaking the component. Thus, we set the id of each slider, and then the slider instance performs a check for what is the id, and sets the style accordingly.

In order to make it clearer to the user, that interacting with one slider, will change the value of the other one as well, we decided to connect the handles with a line. In order to implement this functionality we used an **SVG** HTML Component, where we draw a line and explicitly set its Cartesian coordinates. For the X coordinate, we just needed to calculate the offset according to all the parent container's sizes and margins. The same needed to be done for the Y coordinate, with the addition of multiplying it with the current slider position, in order for the two handles to appear connected.

Confusion matrix

Figure 13: A standard confusion matrix with buttons for each class.

As already indicated in the Design rationales (section 3.2.2), we needed to implement a standard confusion matrix to help the user learn some concepts specific to the domain of machine learning. A further requirement was that the Confusion Matrix is interactive, so that the user can enter adjust the values of confusion classes they want. The latter was almost entirely implemented by the **Confusion Filter** component in PreCall.

The **Confusion Filter** is a component developed for PreCall, and its functionality hasn't been changed. Each confusion filter represents a cell of the confusion matrix. There are some additional features such as a plus and minus button, where the user can increase or decrease the current confusion value. The most important functions are *pmConfusion()*, which handles the clicking of the buttons and either incrementing or decreasing the corresponding value by 0.1. Then, there is *setConfusion* which is essentially the same as in **Confusion Util** (see section 4.2.1), with the difference that instead of returning an index it calls *updateEverything()*.

Since the interactivity requirement is satisfied by the **Confusion Filter**, we just needed a class to instantiate 4 confusion filters and style them appropriately so they look like a standard confusion matrix. This is precisely the role of the **Confusion matrix** component: it acts as a wrapper for multiple **Confusion Filter** components, providing them with a callback function (*updateEverything()*) and taking care of rendering and styling.

Edit viewer

Figure 14: The implementation of the edit viewer.

As per the decision in the design rationales, we also implemented an edit viewer. The idea behind it was to allow a user to relate a threshold value to a real world example. The approach was to display an edit, whose damaging probability is extremely close to the currently selected threshold.

In order to achieve that, we used the dataset ORES was evaluated on,[18] and since the dataset consisted of only the revision ids and whether they are damaging or not, we needed to submit the ids to the ORES API for scoring.

The implementation was rather straightforward - load the edit ids and damaging probabilities from a file and sort them according to probability. Then, we use a binary search to find the closest probability to the current threshold, fetch the edit from Wikipedia, and display the contents. Wikipedia has a function which allows the users to compare two revisions. By substituting the revid in the following URL, Wikipedia returns a diff between the specified and the previous edit. In order not to make too many requests to Wikipedia, a timer was implemented ensuring that a request is made only if the current threshold hasn't changed for 2 seconds.

$$https://en.wikipedia.org/w/index.php?diff=prev\&oldid=revid \quad [19]$$

Furthermore, the edit viewer allows the user to pick whether they want to see a damaging or a harmless edit. In order to achieve that we needed two different dataset files so that searching for a suitable edit is easier. Since the edit viewer fetches the edit with a damaging probability closest to the current threshold, this allows the user to view the "edge case" - namely if the threshold value was a little higher or a little lower, the edit would be classified differently. We are certain this will help the user better understand the consequences of their preferred threshold setting.

4.2.3 Challenges

Performance issues

Probably the biggest challenge we faced has been performance issues. The function *updateEverything()* is rather expensive computationally and because of the multiple sliders it gets called each small step any slider takes. An additional contributing factor to the bad performance was that every time the user would interact with the **Histogram** or the **Confusion Matrix**, *updateEverything()* would update the state of other components. However, during the state update of the **Selector View**, *updateEverything()* would be

[18] `https://github.com/wikimedia/editquality`
[19] Clickable link: `https://en.wikipedia.org/w/index.php?diff=prev%26oldid=7156231`

called again. The result of this behaviour was that when the user would interact with the Histogram's threshold slider, for each step the slider makes, the expensive *updateEverything()* would be called twice.

In order to solve this, we implemented a timer and added a caller parameter to *updateEverything()*. This way we could ensure the function is only called every 25ms, which made all sliders behave smoother. Furthermore we added a check, so that the function makes sure it hasn't been called by the **Histogram** or the **Confusion Matrix** in the last 100ms, and only then gets executed.

Another improvement which helped a lot was using binary search when searching for a concrete or a closest threshold value. With these changes, the application's performance was noticeably better and all the interactive components behaved more smoothly.

Component layout

The layout of the application has been planned ever since the iterative design phase. However, after implementing all of the components we noticed that the user needed to constantly scroll up or down in order to view all of the components at once. For example if the user wanted to interact with the **Histogram**, they wouldn't be able to see the **Edit viewer** and vice versa. After much tweaking of CSS settings and changing scale and layout, we decided that the best solution would be to shrink the **Histogram** horizontally, and displaying the **Confusion Matrix** to the right of it. This way all components would be visible at the same time and the layout of the whole application does not break. Figure 15 shows the final look of the visual interface.

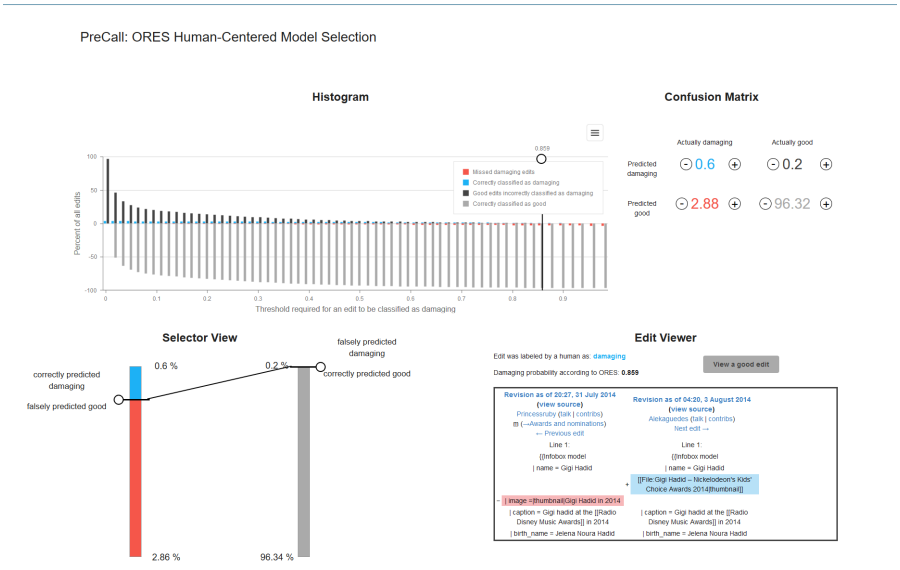


Figure 15: The final view of the program.

5 Feedback

During the implementation phase of this project, a feedback session was performed. We invited a machine learning expert with background in the development of ORES to a conference call. The App was deployed on Heroku[20] and the expert had time to familiarize themselves with the application. The session was free-form in the sense that we described our target group, our design rationales, and our process, while the machine learning expert gave us feedback based on their experience with Wikipedia users and ORES.

The feedback we received was mostly positive, although there were suggestions for potential improvements. Each component was described as useful and easy to understand, when regarded separately, with the edit viewer receiving special praise for being a way to relate a threshold value to a real-world feature. The histogram and selector view being scaled version of one another also received compliments, although the expert admitted that it took them a while to understand the relationship between both. A possible reason is that the scales are different, while the selector sliders have the same size, the blue-red bar on the histogram is way smaller than the dark-light-grey one, contributing to visual clutter and confusion.

A improvement was suggested regarding the data density - the expert explained that many Wikipedia bots use a threshold setting > 0.85 in their configurations,[21] so it would've been useful if one could zoom in or have a finer selection when interacting with thresholds close to 1, as according to him this range is "the most interesting". However, the expert correctly recognized the relationships of the selector sliders, i.e that interacting with the false-positives (right) slider would produce smaller changes on the true-positives (left) slider. This allows the user to fine-tune the desired amount of true positives and thus the desired threshold value.

[20] `https://stark-journey-35616.herokuapp.com/` - (Try refreshing the page if the App doesn't load on the first try.)

[21] `https://github.com/wikimedia/operations-mediawiki-config/blob/master/wmf-config/InitialiseSettings.php#L25969-L26318`

6 Conclusion and outlook

6.1 Conclusion

This research proposes a visual interface, allowing users (primarily Wikipedia bot developers) with limited machine learning knowledge to better understand the functionality and parameters of ORES' "damaging" model, and to easily choose a suitable threshold value for their needs. The design decisions for this interface are based on the insights gained from PreCall and other relevant literature about visualizing classifier performance. The interface features a bar chart histogram, which visualizes confusion classes across all possible threshold settings, an interactive selector view and standard confusion matrix, where the user can adjust the current confusion distribution to their needs, and an edit viewer, which displays a real Wikipedia revision in order to highlight the consequence of the currently selected threshold setting. The evaluation consisted of a free-form trial and discussion with a machine learning expert with background in ORES.

6.2 Outlook

Including, as Kinkeldey et al. would describe them, technical non-experts[9] in the field of machine learning is an important step to adoption. More and more developers need to find and address issues in machine learning applications, despite not having thorough theoretical background.[3] In addition, a more accessible machine learning understanding would mean that users could apply their extensive domain knowledge from other areas in order to develop advanced multi-purpose applications. A way to address that would be to develop visualizations for machine learning models.[4] And in general, the field of Human-Computer Interaction needs more research in order to reveal which assumptions about users' interaction with machine learning models are true, and which are false.[1]

From the already conducted evaluation of the interface proposed in this paper, simple improvements would be to add a way to adjust the desired data-range, so that the user could for example focus on thresholds above 0.85, and to add visual aids in order to convey the connection between the histogram and the selector view. Conducting further evaluations with multiple technical non-expert participants will undoubtedly highlight potential improvements of this interface. On a larger scale, the proposed interface could be used with alternative datasets, both related and unrelated to Wikipedia, in order to evaluate the interface's usefulness in providing understanding for general machine learning problems. It would also be interesting to expand this current interface for multi-class probabilistic problems. In general we are confident in the potential of our software, and would be excited to see further developments.

References

[1] S. Amershi, M. Cakmak, W. B. Knox, and T. Kulesza. Power to the people: The role of humans in interactive machine learning. *AI Magazine*, 35(4):105–120, Dec. 2014.

[2] E. Beauxis-Aussalet, J. van Doorn, and L. Hardman. Supporting end-user understanding of classification errors: Visualization and usability issues. Aug. 2019.

[3] D. Chen, R. K. E. Bellamy, P. K. Malkin, and T. Erickson. Diagnostic visualization for non-expert machine learning practitioners: A design study. In *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 87–95, 2016.

[4] J. J. Dudley and P. O. Kristensson. A review of user interface design for interactive machine learning. *ACM Trans. Interact. Intell. Syst.*, 8(2), June 2018.

[5] T. Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861 – 874, 2006. ROC Analysis in Pattern Recognition.

[6] T. Gülenman. Precall: A visual interface for threshold optimization in machine learning model selection. *B.Sc. thesis, HCC, Institute of Computer Science at the Free University of Berlin*, 05 2019.

[7] A. Halfaker and R. S. Geiger. Ores : Facilitating remediation of wikipedia ' s socio-technical problems. 2018.

[8] M. Hossin and S. M.N. A review on evaluation metrics for data classification evaluations. *International Journal of Data Mining and Knowledge Management Process*, 5:01–11, 03 2015.

[9] C. Kinkeldey, C. Müller-Birn, T. Gülenman, J. J. Benjamin, and A. Halfaker. Precall: A visual interface for threshold optimization in ml model selection. *ArXiv*, abs/1907.05131, 2019.

[10] T. Saito and M. Rehmsmeier. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PLOS ONE*, 10(3):1–21, 03 2015.

[11] K. M. Ting. *Confusion Matrix*, pages 209–209. Springer US, Boston, MA, 2010.