ZA Cloud SRL, str. Govora 16A, 400664
Cluj-Napoca, România, Tel: +40 723 175
635, Email: office@zettacloud.ro

“Code Against Hate” Hackathon

ONLINE | 25 – 27 September 2020

Report on Hackathon’s Results and Further Steps Suggestions

ZA Cloud SRL, str. Govora 16A, 400664
Cluj-Napoca, România, Tel: +40 723 175
635, Email: office@zettacloud.ro

Delivered Solutions & Resources

We received resources belonging to one single solution: HATE:NO. This are the resources we've received to be analysed regarding HATE:NO solution:

1. GitHub repository containing the solution code: https://github.com/Zahorack/code-against-hate
2. Demo WebSite Link: http://rholly.sk/
3. Pitching Session for HATE:NO (one 1h, 6min and 22s vide): https://drive.google.com/file/d/1Ie9x2FHARPeAeAvnQwmr31eSN5qhdrRq/view?usp=sharing

HATE:NO

The solution consists of a Flask-based Web Service for exposing the NLP classification model predictions and a demo WebSite that shows how such predictions could be used in integration with a chat application.

Testing HATE:NO

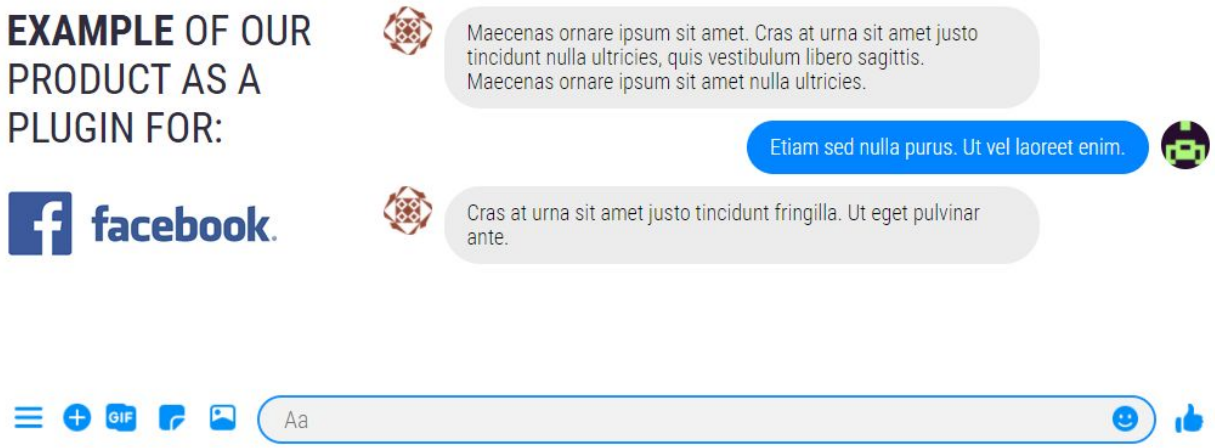
Demo WebSite Experience

The website is very minimalistic. The WebSite graphical design is clean with nice fonts and colors, in tone with the WebSite design trends of the year.

Regarding users' experience when reaching the site, the introductory part has a self-explanatory headline containing the high-level objective, but does not yet explain in detail what is the plan to reaching the objective.

ZA Cloud SRL, str. Govora 16A, 400664
Cluj-Napoca, România, Tel: +40 723 175
635, Email: office@zettacloud.ro

Regarding the “Example” part of the website, there is no clean explanation about what needs to be done to test the service. The Facebook logo is used but there is no direct connection, integration with Facebook Messenger.



Hate Speech Classification Model

The Development team does not offer any information about the training set used to train the used NLP classification model. There is also no information regarding the accuracy, F1, precision or recall that the model achieves. Therefore, we performed a battery of tests to understand the accuracy of the hate speech classification model. These tests were done automatically using a set of 29443 ground truth samples of hate and not hate speeches. Out of the 29.443 predictions, only 15.067 were correct resulting in an accuracy of the model of just **51.17%**. This is far from a production ready classification model. The industry expects AI models to be very near to 100% with 80% being the minimum accuracy that would be acceptable in a production environment.

Below you’ll find a sample from the classification tests we’ve performed. The testing results table has 3 columns: the first column contains the input text, the second shows the classification model prediction (binary: hate / not hate) and the third column represents the ground truth, what is expected from the human evaluator.

Input Text	HATE:NO	Ground Truth
hello world	hate	not hate

the table is black	hate	not hate
you should not use the n-word	hate	not hate
she is bad at this	not hate	not hate
africans have a very diverse culture	not hate	not hate
she is a beautiful black woman	hate	not hate
she is a beautiful white woman	hate	not hate
what's your opinion on white people in general?	hate	not hate
covid is not chinese made	hate	not hate
we should not blame asians for these issues	not hate	not hate
kikes control everything	not hate	hate
jews control the banks	not hate	hate
she looks like a hooker	not hate	hate
muslims are a threat	not hate	hate
arabs should be deported	not hate	hate
don't be a nigger	hate	hate
she is bad at this because she is a woman	hate	hate
go back to the kitchen	hate	hate
you are such a faggot	not hate	hate
raging faggot	not hate	hate

HATE:NO Code Review

An archive containing the code was downloaded from https://github.com/Zahorack/code-against-hate on October 23rd. The structure of this looks like the following:

- **config**: directory with one empty config.py file

- **model**: directory containing the classification model; it seems to be a TensorFlow model - there is no indication what data was used for training and testing and what were the hyperparameters used for training the classification model
- **static**: directory containing the static resources used by the HTML demo site - images and CSS files
- **templates**: directory containing the HTML pages of the demo site - index.html seems to be the home page of the demo site containing all functions needed to integrate the prediction solution
- **server.py**: Python Flask-based HTTP server that does the prediction using the pretrained model. See more details below.

HTML Demo WebSite

The objective of the demo site is to show how a 3rd party application would interact with the prediction server. Checking the submitted message happens inside the JavaScript function at line 53, checkSubmittedMessage:

```
function checkSubmittedMessage() {
        if ($("#fcb-chat-input").val().length > 0) {
            $.get('http://rholly.sk:80/api/v1/check', {message: $('#fcb-chat-input').val()},
function (data, textStatus, jqXHR) {
                if (data.result == 1 ) {
                    $("#popup-overlay").removeClass("display-none");
                    $("#popup-buttons-no").focus();
                } else {
                    save_fb_message();
                }
            }).fail(function() {
                save_fb_message();
            });
        }
    }
```

Given text is sent to the /check end-point of the API. The data is sent in clear-text via a HTTP GET request. The expected response seems to be a binary classification: 1 for “hate speech” and 0 for not “hate speech”. When “hate speech” is predicted, the “popup-overlay” div is shown, otherwise the message is “saved” using the save_fb_message() JavaScript function.

save_fb_message() JavaScript function adds a new message to the page. The messages are volatile, meaning that they do not persist after the index.html page is reloaded.

server.py

This is a Flask based implementation of the HTTP server serving the solution. Flask is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. To run the server you need a Python environment where Flask and Tensorflow packages are installed. The implementation tries to bind the server implementation on port 80, that means that it needs to run on the server with root privileges.

Each time the server starts, the Tensorflow classification model is loaded. Catastrophic errors when loading the Tensorflow classification model are not catched or handled.

There are two end-points implemented by the server:

1. @app.route('/', methods=['GET']) - that loads the index.html of the Demo WebSite.
2. @app.route('/api/v1/check', methods=['GET']) - implements the /check end-point: given a text, it uses the previously loaded classification model to predict if the message is “hate speech” or not. The development team considers hate speech any prediction value greater than 0.5.

HATE:NO Readiness for Production And Integration

The solution we are currently reviewing is just a simple demo implementation and shows basically how one could serve the results of a Tensorflow classification model via a simple API implementation. As is, **the solution is not ready for production purposes.**

While lightweight and easy to use, Flask’s built-in server is not suitable for production as it doesn’t scale well and by default serves only one request at a time. The built-in Flask web server is provided for development purposes only.

With it you can make your app accessible on your local machine without having to set up other services and make them play together nicely. However, it is only meant to be used by one person at a time, and is built this way. It can also serve static files, but does so very slowly compared to tools which are built to do it quickly. This does not matter when only one person is accessing it, so it’s perfect for what it is meant for.

When running a Web App in production, you want it to be able to handle multiple users and many requests with a very quick pace. There are a multitude of deployment options for a productive environment described here: https://flask.palletsprojects.com/en/1.1.x/deploying/.

When it comes to integrating this solution with publishers or social media platforms, the 3rd party publisher or social media platform has two options:

1. Run the server inside their own infrastructure and do predictions as needed.
2. Send the messages to the server deployed outside their organization and operated by another entity (eventually, an entity where the Dev Team of HATE:NO is integrated).

In an enterprise environment, we see that option 2. above is not feasible since the messages to be checked need to travel outside their organization. Important security measures need to be employed and the infrastructure running the prediction server needs to offer great performance and security.

Option 1. above would be the only way such a solution could be productivized in an enterprise environment. But this option means that the Development Team loses control over the solution and there is no simple way, for example, to update the prediction model, other than going through the 3rd party published or social media platform acceptance and deployment terms and conditions.

Suggestions on Further Steps

HATE:NO solution is a minimalistic demonstration of how one could serve hate / no hate classification predictions as part of a simple API end-point. If the purpose of the solution was just to show that such a prediction could happen in real-time, then this purpose was reached.

On the other hand, the solution is far from reaching the production state.

Here are couple of suggestions for the HATE:NO development team to bring their solution nearer to a production-ready state:

1. **The accuracy of the classification model should be increased** to at least 80%, with the aim to reach more than 90%. This can be achieved by gathering enough samples of “hate” and “not hate” texts and labeling these together into a massive annotated text corpus. This is a very time consuming process that demands good knowledge of what is hate and what is not hate speech to avoid as much as possible biases that the model could inherit through training. Gathering enough information could be done by identifying online forums, blogs or other places containing hate speech and running scraping scripts over thousands of comments and texts. Afterwards, these raw materials need to be analyzed (semi-automatic processing is possible if you find some rules that would help you label hate speech automatically) and labeled correctly. With the corpus ready for

training, the classification model needs to be retrained, tested and retrained again until the achieved accuracy is reached. Also, make sure that other evaluation scores like F1, precision and recall are good enough for production.

2. **The architecture of the solution should be reviewed** so that would be suitable for a production integration. The Overview of Online Hate Speech Detection Solutions document, resource that was available before the hackathon described two suggested architectures, one suggesting that building upon OSMod - The ConversationAI Moderator App would give the solution a very solid head start. This is due to the fact that OSMod is a machine-assisted human-moderation toolkit with all the orchestration functionality ready out-of-the-box, like Moderator Authentication, Comment State Flow în integration with modules ready for getting comments out of YouTube video, for example, and the Assistant Protocol, the component that bridges between the OSMOD backend and the machine learning models. Here is the place for you to hook up the NLP classification model. The system offers an out-of-the-box Worker / Task Queue that is essential for running in a production environment where hundreds of thousands of messages need to be processed by the system. Last but not least, there are three ways to deploy the system in production - you can find all these stuff here: https://github.com/conversationai/conversationai-moderator/tree/master/deployments.

Emil Ştefănco

CEO, Zetta Cloud

m:+40 723 175 635 | e:emil@zettacloud.ro |
w:www.zettacloud.ro | a:Cluj-Napoca, România