Neural Wave Hackathon

AI4Privacy Challenge - Hackabros Team

Kaan Egemen Sen **Ian Abuaf Pelo** **Lorenzo Zaniol**

27/10/2024

1. Introduction

In this challenge, we mask the personal information in the text/strings based on the pii-masking-400k dataset on Huggingface. To achieve this, we follow the methodology in PII Masking Problem which follows p5y privacy masking language, using the format: `[NAMEOFTHEINFO]`. The sample submission for the challenge, Piiranha-v1 Model, which already achieves remarkable results, reports an accuracy of 99.44%, with a loss of 0.0173 and precision, recall and F1 scores of approximately 93% with eighteen possible categories/labels (17 PII and 1 Non PII). In our work, we aim to improve on these results. The general idea is to concentrate on the worse performing labels: mainly *ACCOUNTNUM* and *IDCARDNUM*.

In addition to that, an heuristic engine based on our custom regex library and data structure, with the goal of ease by automating string matching and replacement (while maintaining priority control); it is used for simplify the input date for the models on input as well as identify new labels that still contain sensitive and/or private informations causing still an privacy leakage, that it is exactly what this challenges wants to patch and avoid.

As such, our complete pipeline makes use of both AI machine learning models and more traditional approaches. The full pipeline can be seen in Figure 1.

1.1. Model Analysis

We performed a brief analysis of the Piiranha-v1 Model to understand its weaknesses and assess a strategy to improve it.

One quirk of the model that we identified quickly is that it often marks tokens around the Personally Identifiable Information (PII) as PII as well. This can sometimes cause the formatting to break, for example when a phone number is placed just before an HTML tag, the piiranha model may redact part of the HTML tag as well.

2. Pre-processing

We implemented pre-processing to ease the recognition of the following labels/categories: **Phone number**, **IBAN/Tax number** and **ID card number** by exploiting our custom regex library/helper.

Instead of masking directly (like in post-processing), which may cause issues when running pretrained models afterwards, we replace the sub-string with a string of the same label-type, but that it is easily detected by the models; in other words the one that knows best and more present in the dataset. We could do this with also email but since the submitted model has 100% accuracy this would constitute an unnecessary step.

3. Mini Piidgeon Models

In this experiment, we intentionally overtrain specific sub-models to precisely identify smaller, more challenging label segments that are prone to misclassification by the pretrained Piranha model. The Piranha model, with its complexity of handling 17 distinct label categories, often confuses these smaller label segments, particularly for labels with low initial performance. To investigate, we focused on two underperforming labels, ACCOUNTNUM and IDCARDNUM, as shown in Table 1.

	Precision	Recall	F-1
ACCOUNTNUM	0.84	0.87	0.85
IDCARDNUM	0.89	0.94	0.91

Table 1: Table with metrics obtained from Piiranha model

To improve performance on these labels, we trained two versions of a piidgeon model: one optimized for maximum accuracy and another for maximum label precision. This targeted approach allowed us to compare the effects of each objective function on the model's performance.

The resulting metrics for each piidgeon are illustrated in Figure 3. Subfigure 3a shows the results from training the piidgeon to maximize accuracy, while Subfigure 3b focuses on maximizing label precision. For clarity, label 1 corresponds to ACCOUNTNUM, and label 2 corresponds to IDCARDNUM.

This experiment demonstrates the potential to enhance the Piiranha model's performance by creating specialized sub-models focused on underperforming labels. For this study, we extracted 31,000 rows of data specific to these labels from the general 400,000-row dataset, offering a more tailored dataset for training. Future work could involve more extensive data augmentation and advanced optimization techniques to further improve model performance. Expanding the number of targeted piidgeons may also lead to more robust handling of challenging label categories.

A summary of the models' validation metrics can be seen in Figure 2. In particular notice that the precision of both labels in the model trained for precision (Figure 2b) surpasses that of piiranha's model: we achieve 0.90 precision on our worst performing label of the two, while piiranha only achieves 0.84. This demonstrates the power of using smaller models over one big model.

4. Post-Processing

We created a post-processing step to the pipeline that handles the following additional labels/categories: **MAC address**, **IPv4**, **IPv6** and **UUID v1-v7** by exploiting our custom regex library/helper.

This masks the labels by replacing the sub-string with the respective string with the p5y privacy masking language format.

5. Problems encountered

During our experiments we encountered a pytorch bug, while doing memory optimization, it can be seen in this link: Pytorch Bug

Another major issue we encountered is with regards to the tokenizer used by the piiranha model which we base our own model on. We initially assumed that the tokenized column in the dataset used the same tokenizer as the piiranha's model, that is `DebertaTokenizer`, however we found that the dataset uses a `BertTokenizer` instead.

This lead to the classification of the tokens which we use as target not aligning with the input tokens and the model returning unpredictable results. We therefore decided to keep our current code and switch the model's tokeniser to the `BertTokenizer`. This fixed our issue.

Links

- GitHub: `https://github.com/Neural-Wave/project-Hackerbros`

A. Appendix

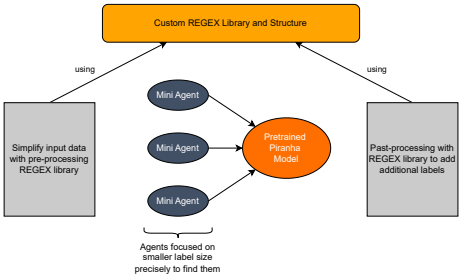
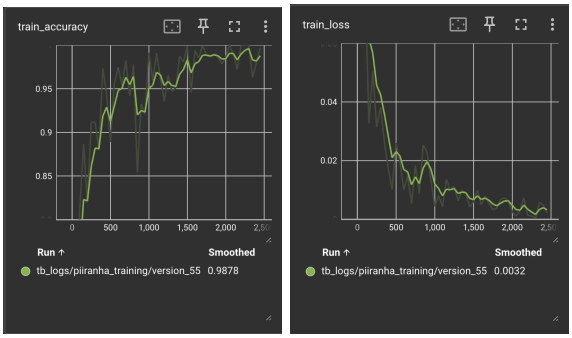


Figure 1: Custom pipeline



(a) Train accuracy of piidgeon

(b) Training loss of piidgeon

Figure 2: Training activity of piidgeon precision


```
"val_metrics": {
  "val_loss": 0.017158156260848045,
  "val_precision": 0.9362433552742004,
  "val_recall": 0.9460272192955017,
  "val_f1": 0.9381852149963379,
  "val_accuracy": 0.9460272192955017,
  "val_precision_label_0": 0.997978925704956,
  "val_recall_label_0": 0.9977092146873474,
  "val_f1_label_0": 0.9978418350219727,
  "val_precision_label_1": 0.9276978373527527,
  "val_recall_label_1": 0.8929612636566162,
  "val_f1_label_1": 0.9062042832374573,
  "val_precision_label_2": 0.8830540776252747,
  "val_recall_label_2": 0.9474120736122131,
  "val_f1_label_2": 0.9105081558227539
}
```

```
"val_metrics": {
  "val_loss": 0.020924272015690804,
  "val_precision": 0.9471548199653625,
  "val_recall": 0.9440147876739502,
  "val_f1": 0.9436668157577515,
  "val_accuracy": 0.9440147876739502,
  "val_precision_label_0": 0.9976325631141663,
  "val_recall_label_0": 0.9984686374664307,
  "val_f1_label_0": 0.9980491399765015,
  "val_precision_label_1": 0.9340932369232178,
  "val_recall_label_1": 0.9032707810401917,
  "val_f1_label_1": 0.9159783720970154,
  "val_precision_label_2": 0.9097387790679932,
  "val_recall_label_2": 0.9303042888641357,
  "val_f1_label_2": 0.9169729948043823
}
```


(a) Trained piidgeon to maximize accuracy

(b) Trained piidgeon to maximize label precision

Figure 3: Metric information for the piidgeon