Bocconi

Machine Learning and Artificial Intelligence

PII Detection

Andrea Fabbricatore - 3160153
Alessandro Pranzo - 3159045
Luca Raffo - 3177868

Academic Year 2023/2024

1 Introduction

With the advent of large language models (LLMs) and the increasing focus on privacy and data protection, such as the AI Act in Europe, there is a growing need for automated Personal Identifiable Information (PII) detection. This need is particularly pronounced in industries such as legal, medical, educational, and AI model training, where repurposing data is challenging due to the labor-intensive nature of manual anonymization processes.
Existing literature highlights various approaches and models that address this issue. Prior to the rise of deep learning, traditional methods relied heavily on rule-based systems and regular expressions. While these methods were somewhat effective, they often lacked the flexibility and accuracy necessary for handling complex and context-dependent language processing tasks.

The introduction of transformer-based models, such as BERT (Bidirectional Encoder Representations from Transformers), has significantly improved performance in numerous NLP tasks by effectively capturing contextual information. Specifically, the expertise of BERT models in the realm of MLM (Masked Language Modeling) render it a great choice for the task at hand. Thus, fine-tuning these pre-trained transformer models on specific datasets for PII detection has emerged as a promising approach, offering enhanced accuracy and adaptability. The reason behind using a Transformer architecture is that of giving value to each token by contextualizing it with the rest of the data, assuming that it will positively contribute to its overall performance - such as, `William Shakespeare` versus `Andrea Fabbricatore`, which despite both being names evidently hold different contexts in the realm of PII.

The goal of this paper will be that of devising a Transformer-based PII Detection model by fine-tuning already trained BERT-based LLMs, to automatically anonymize text data for privacy concerns.

1.1 Examples

Consider the following example:

```
My friend Andrea was reading a philosophy book by Spinoza.
```

`Andrea`, should be tagged, as it is a personal name. However, `Spinoza`, despite being a personal name, is not considered as private information, as it refers obviously to the famous Dutch philosopher. The result we want to obtain should look something like the following:

```
My friend [B-GIVENNAME] was reading a philosophy book by Spinoza.
```

Similarly, we would like to mask out dates, times, locations, addresses and many more. A complete list of the labels can be found in the notebook.

2 Dataset Exploration

We will use a dataset provided by Hugging Face, called pii-masking-300k, which at the time of writing this report is the world's largest open dataset for privacy masking. It is completely generated syntethically by a proprietary algorithm, and it includes 6 languages (English, French, German, Italian, Dutch, and Spanish). Each row represents a json object with a natural language text that includes placeholders for PII.
Here is an example:

1. source text:

```
I was with Paolo in New York.
```

2. target text:

```
I was with [GIVENNAME1] in [LOCATION].
```

3. privacy mask:

```
[{"value": "Paolo", "start": 11, "end": 21, "label": "GIVENNAME1"},
{"value": "New York", "start": 25, "end": 31, "label": "LOCATION"}]
```

4. span labels:

```
[[11, 21, "GIVENNAME1"], [25, 31, "LOCATION"]]
```

5. mberttokens (indicates the breakdown of the text into tokens associated with mbert):

```
["I", "was", "with", "Paolo", "in", "New", "York", "."]
```

   notice that other common english words are split during tokenization, e.g., `subject` becomes `["sub", "##ject"]`.

6. language:

```
english
```

7. set:

```
train
```

this can be either train (79% of the dataset), or validation.

8. mbert_bio_labels (demonstrates the labels associated with the BIO labelling task in Machine Learning using the mbert tokens):

```
["O", "O", "O", "B-GIVENNAME1", "O", "B-LOCATION", "I-LOCATION", "O"]
```

The BIO labels in the context of Named Entity Recognition (NER) have the following meanings:

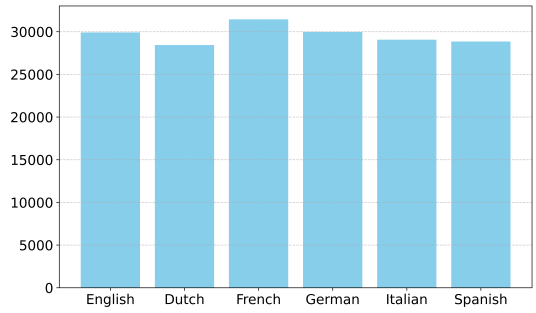
- `O` stands for `OUTSIDE`, indicating that the token does not belong to any entity that needs to be masked.

- `B` stands for `BEGINNING`, indicating the beginning of an entity that needs to be masked.

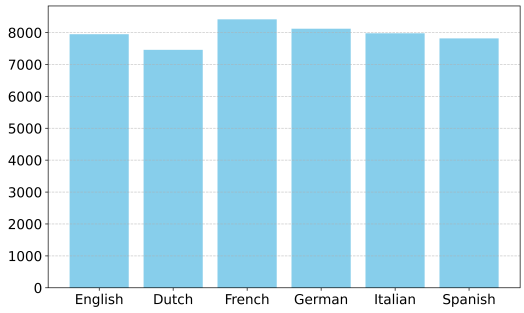
- `I` stands for `INTERMEDIATE`, indicating that the token is part of an entity that has already started.

We have a total of 46 unique labels, plus the `OUTSIDE` which is not connected to any masking.

We can check that languages are equally split (both on training and validation sets):

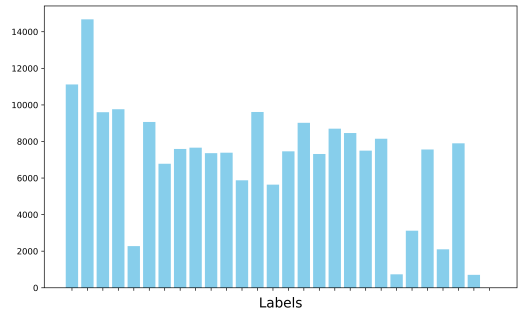


(a) Languages distribution in Train

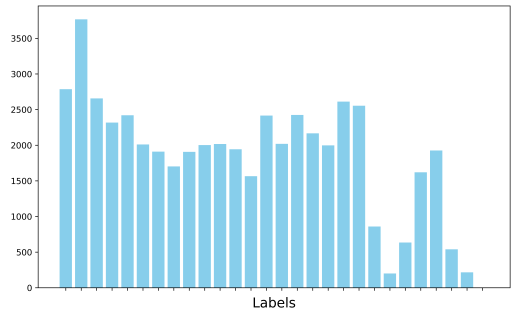


(b) Languages distribution in Validation

We have the same amount of data for each language. To make our notebook computationally feasible, we proceed to keep only the English data. To check whether we have a balanced dataset, we want to plot the distribution of the labels. As we can see, the labels are distributed quite evenly in both datasets. There is an outlier from above (`TIME`), and some outliers from below (`CARDISSUER, GEOCORD, LASTNAME3`). We expect our predicitons to be accurate, aside the ones regarding this last class of labels.



(a) Labels distribution in Train



(b) Labels distribution in Validation

3 Methodology

Our approach involves using two pre-trained `BERT` models, fine-tuning them on our custom dataset tailored for PII detection, and finishing with an ensemble. Specifically we chose `ALBERT` (A Lite BERT) and `DistilBERT` due to their efficiency and performance. `ALBERT` is known for its parameter efficiency and lower memory footprint, making it suitable for tasks requiring high performance with limited resources. `DistilBERT`, on the other hand, offers a compressed version of BERT, maintaining a balance between speed and accuracy.

We can break down our pipeline into 3 steps:

1. Firstly we import two copies of the pre-trained `ALBERT` model. Both will be fine-tuned on our dataset, with different hyperparameters or regularization methodologies. We will select the best out of the two.

2. Secondly we repeat the procedure for two copies of `DistilBERT`.

3. Lastly we finish with an ensemble of the best `ALBERT` and the best `DistilBERT`.

More specifically the first copy of `ALBERT` will be fine-tuned for an epoch with an `ADAM SGD` on a batch size of 16. The second copy will be fine-tuned on an `ADAM SGD` on a batch size of 64, together with a regularization given by a label smoothing with $\epsilon = 0.2$. In order to fine-tune the models, we add a linear layer at the end of both to let the number of logits match the number of labels of our dataset (this is needed to evaluate the cross-loss entropy, otherwise we wouldn't be able to match the size of the softmaxed logits).

The same exact procedure will be applied to the two `DistilBERT` copies.

Once we have the best two models, it may happen that one of the two is better in predicting a certain class of labels, and viceversa. In order to account for this, we merge the two models with a linear regression: for each label `i` there will be a coefficient $0 \leq$ `alpha_i` ≤ 1, so that we are going to weigh the softmaxes for both the models' outputs on the fixed label. To clarify, let's make an example:

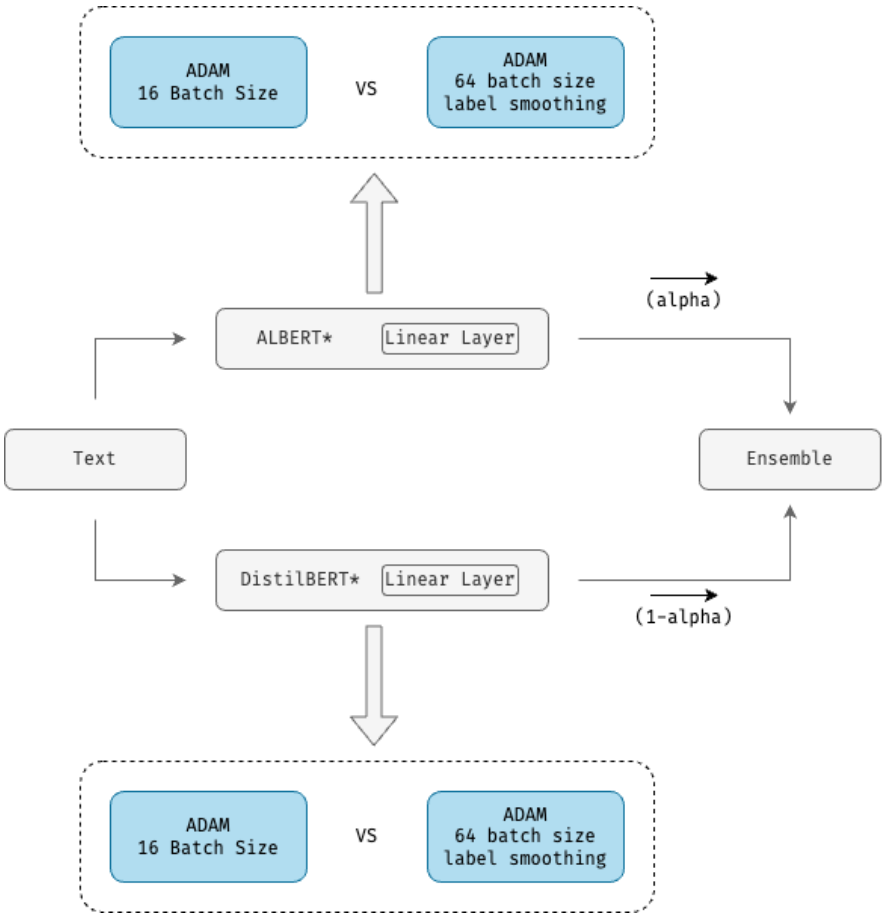


Figure 3: Pipeline

let's suppose we are trying to infer a certain word, for instance `Alessandro`.

Let's say that `DistilBERT*` (from now on we will add `*` to the name of the model when we refer to the one with 'the best out of the two classes of hyper-parameters', while we use the original name to refer to both) predicts a soft-maxed logit vector $\mathbf{x}$, and `ALBERT*` predicts a soft-maxed logit vector $\mathbf{y}$ for `Alessandro`.

If during the training we learned that when the first model infers label i it is right most of the times, it would be smart to give x_i a big weight α_i, and to y_i a smaller weight $1 - \alpha_i$.

This is exactly what we will do: we train the parameter α with 5 epochs of training after we fine-tuned our models.

4 Data Pipeline

The pre-processing portion for this task was quite lengthy given that we had to match the source texts and span labels to different tokenizers accounting for the inherent shifts that come with it. Specifically, we implemented a quite robust procedure to ensure that our dataset correctly represented the original data for both our models `ALBERT` and `DistilBERT`. We first split the dataset into Train and Test, and only selected English as aforementioned - leaving us with 29908 and 7946 data points respectively.

We only made use of the `full_text` and `span_labels` features as we would then take care of the tokenization process with the respective models' tokenizers, moreover we also made use of the

Spacy libary to standardize the procedure as will be now shown.

The main issue when preparing text for NER is that the span labels which are previously matched to indices of the full text will now have to be matched to tokens that will then be inputted into the BERT model. Specifically, we had to find a way to do such a matching. This was done with the following procedure:

1. The source text was originally parsed with the `en_core_web_sm` pipeline from spacy and thereby split into words (not tokens) which were generalized regardless of the direction we wanted to apprehend in the following steps.

2. We then used the `offsets_to_biluo_tags` to simply match the span labels to the NER tags of the tokens by keeping a rolling sum of the indices passed so far in the sentence. Note that there were instances where the tags could not be properly alligned due to indices and slicing issues and these were therefore skipped and not included in the final datasets used for the trainings.

3. The now aligned BILUO tags were converted to the more standard nomenclature BIO tags that we explained before through a simple substitution method - we moreover thought that the added simplicity of this naming convention might help our model perform better. In addition, the original tags in the dataset included indices at the end of them such as `GIVENNAME1`, which were stripped.

4. The tokenizers from the respective `ALBERT` and `DistilBERT` models were used, setting truncation to TRUE to fit their max length and letting the tokenizer know that we wanted the Spacy tokens to be considered as the original unique words - from which we received word ids which will be useful later on.

5. Once the examples were tokenized they underwent an aligning process to account for the shifts and splits in words that occur when using a BERT tokenizer (such as the appearance `SEP` and `CLS` tokens as well as the evident splitting of words that comes from a BPE style embedding). Specifically the tokens underwent the following procedure, we found the word id (i.e. the index of the original spacy word from which the new token came from) and matched the NER tag (its encoded numerical id generated by us) by finding the original tag assigned in step 3), moreover if a tagged word was split into more tokens only the first token was tagged - while the others received a place holder -100. The `SEP` and `CLS` also received a value of -100 so that the model would not account them when masking.

We then finished the train and test datasets by keeping the `input_ids` (dictionary id of a token in our BERT models), mapped `BIO_labels` and some other data useful for our personal testing process.

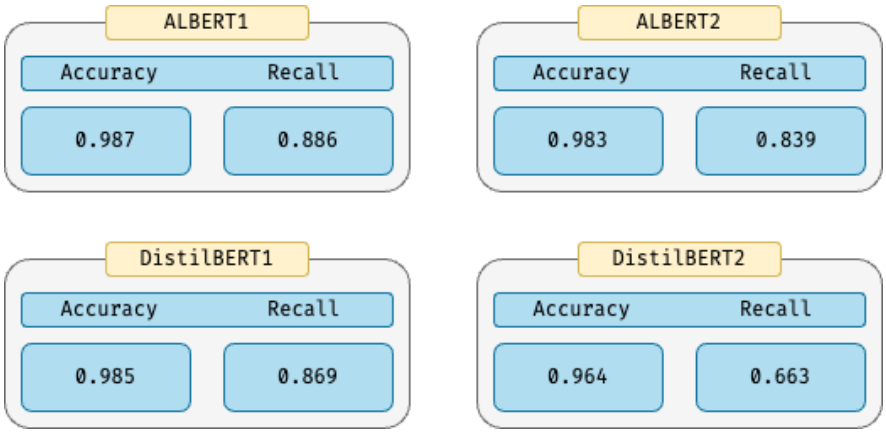
The ensemble pipeline followed a similar procedure whereby we tokenized the dataset with both models and saved pairs of all the data mentioned above (so they could be fed into both `ALBERT` and `DistilBERT`), however in this case we kept the BIO labels in the original Spacy version for a reason that will be explained later on in the Ensemble Section.

It is worth mentioning that some interesting results were encountered during this phase, specifically

that unlike expected, the word ids are not always sequential - it may be possible that a tokenizer might split words in a sentence in such a way where an original word id is entirely skipped (or more so, engulfed by the surrounding words), this can especially happen with punctuation when being considered as a word and then being taken up by a token in the embedding process of our BERT models. This caused some problems in the ensembling procedure as it complicated the matching and standardization of the logits of each model, as we will discuss in the **Results and Further Developments** section.

5 Fine-tuning

The models were trained using the parameters mentioned in the previous sections and using the HuggingFace Transformers library - specifically we made use of the `AutoModelForTokenClassification` class as it allowed a very neat implementation of a dropout and a linear layer to match the number of labels used in our PII detection task - the importance of the dropout was that it allowed the model to reduce co-adaptation (that we would then forcefully introduce in the ensembling part) and better generalize on the data. After all the trainings were done we received the following results:



In our task of PII detection as one may imagine False Negatives are very costly (as it would mean a leak of sensitive data - tagging a token as normal despite containing personal information), for that reason the metrics we are interested in, besides mere accuracy are the recall rates ($\frac{TP}{TP+FN}$). Given this notion we can then see that the best models we should use in the ensembling are `DistilBERT1` $\longrightarrow$ `DistilBERT*` and `ALBERT1` $\longrightarrow$ `ALBERT*` .

6 Ensembling Architecture

As previously shown in the depictive diagram of the model architecture, an input must be tokenized by both the `ALBERT` and `DistilBERT` tokenizers before being fed into their respective models. Once the logits are produced, they form matrices where each row is a vector of logits over the number of labels (thus we find a mismatch in the first dimension, as it comes from different tokenizers, while the second dimension stays the same being the number of labels). However, to ensemble these outputs, the number of rows must be fixed to the same size so that the parameter α can be used to combine the predictions of both models, because we want to combine predictions on every word, regardless of the way it was tokenized.

Before explaining the ensembling process, it is important to note that a softmax function is applied to both BERT models' outputs and to the ensembled outputs. This is necessary because weighing two logits with different norms can be ineffective; logits with the highest norm would dominate regardless of the parameter α. The final result is softmax-ed again to provide clearer, more distinct peaks.

Now going back to process of the ensembling and specifically how the logits were made to match - as they are indeed logits coming from different tokenizers, despite representing the same thing. An `ensembler` function was created - it makes use of the previously obtained word ids in the data pipeline as well as the logits generated by both models to cohesively reshape the two models' output as follows:

1. Remove the padding tokens such as `SEP` and `CLS` and respective rows in the matrices. Moreover, calculate an average over the rows to act as a placeholder label distribution to be used in the following step as will be shown.

2. Iterate through the word ids, and average the logits of the rows coming from the same original Spacy word; in the case that the current word id is not sequential to the previous, also include a place holder row in between this and the previous one. The latter step is done to account for the "missing" word ids that occur when tokenizing with different models and therefore due to the mismatch that comes from a BPE combination of tokens.

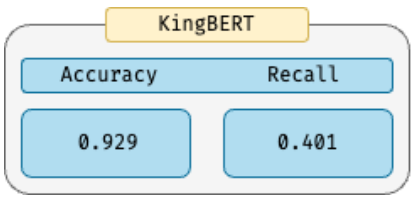
Following these steps, the result is two matrices (one for each BERT model) with the same shape, where the number of rows matches the number of words in the original Spacy embeddings. Evidently, the evaluation labels are the Spacy BIO tags, which share the same dimensionality as the matrices produced by the ensembling process. These can be evaluated using Cross Entropy Loss to determine the optimal parameter α for combining each token's labels.

By ensuring that the logits from both models are aligned and appropriately normalized, the ensembler can ideally combine the predictions, leveraging the strengths of both `ALBERT*` and `DistilBERT*`.

7 Results and Further Developments

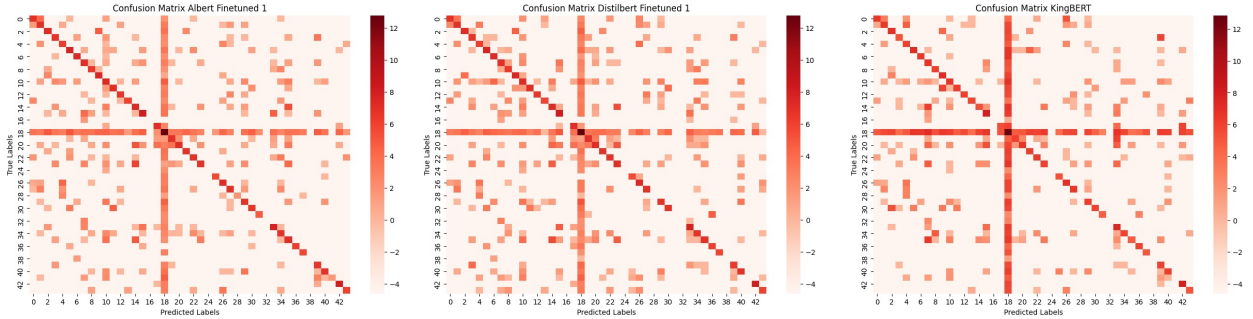
After completing training of the ensemble model `KingBERT`, it is now important to compare the performances of the various trainings to understand whether the ensembling worked and what could have been done to further improve the results obtained.

Specifically the ensemble performed as follows when evaluated on the same test set as the previous `ALBERT` and `DistilBERT` evaluations. The performances this time were:



As is evident when comparing to previous results, there was a drop in performance after the ensembling process - this might come as a surprise as the model should in theory be, in the worst case scenario, performing as well as the least performing input model (i.e. the `ALBERT*`).

Moreover, let's also visually compare the confusion matrices to see whether they can give us some insight into the various models PII detection performances. To do so, it is important to note that the label 18 represents the O tag (i.e. non-sensitive data). Moreover, the vertical line above the 18 tag represents False Negatives (i.e. what we are trying to minimize - so the darker it is, the worse the recall).



(a) ALBERT (b) DistilBERT (c) KingBERT

Visually the metrics previously presented are quite visible in the image as most models have very high levels of accuracy - as generally there are few misclassifications in comparison to the very intense color of the diagonal which represents good classifications. In addition, it can be seen that the vertical line (False Negatives) of the ensembling model are much more intense than for the individual ones, and the reasons for this will be discussed in the limitations part.

Overall, it seems as though a good result was reached and that the idea for the ensembling can both theoretically and practically work - despite some issues in the implementation that evidently hindered its performance. Specifically, as discussed the main issue was not so much the drop in performance but rather the drop in recall (which means that PII data was often classified as not sensitive) - this might actually be due to the way in which the data was reshaped in the ensembler. First of all, taking an average of the the logits representing the tokens of a single word is a quite risky approach as it may have potentially led to the softmax-ed probabilities not being as peaked in the output matrix (increasing the risk for misclassification). Moreover, the calculation of the placeholder row, to include when skipping word ids, might have had a big role in the drop of performance as one may argue that the model was being trained on setting an α that mislabelled the data purposefully (as the placeholder did not in fact match the label it was meant to have - so there was a trade off of wanting to maximize performance for right and wrong labels).

The extension of this project would be that of finding solutions to the issues mentioned above (by for example, taking a max of the rows instead of the average, taking the logits of the first token

of the word and so on...) and rethinking the way the ensemble model is trained and the outputs of the BERTs are matched. Moreover, if more individual models were finetuned it would be interesting to also compare the performance by using majority voting procedures or even fine tune on different datasets and labels to see which combinations would maximise results. The main implementation that would benefit our solution would be the one of devising our custom tokenizer, that could be trained to handle the same tokens on both models, so to smooth the process in the ensembler and avoid mismatching problems.