Detect person names in text: Part 2 (Technical)

Jan Procházka / January 19, 2021 /
Deep Learning (https://pii-tools.com/category/deep-learning/),
Personal Data (https://pii-tools.com/category/personal-data/),
Whitepaper (https://pii-tools.com/category/whitepaper/)

In Detect Person Names in Text: Part 1 (Results) (https://pii-tools.com/detect-person-names-in-text/), we benchmarked our new named entity recognizer (NER) against popular open source alternatives, such as Stanford NER, Stanza and SpaCy. Today we dig a little deeper into the NER architecture and technical details.

First, recall our main NER objectives. In short, we require our NER to be **practical**, rather than just accurate on some clean academic dataset:

- **Useful in arbitrary document contexts**: this is where most off-the-shelf solutions fail, having a narrow domain of application and failing horribly on out-of-domain (AKA real-world) documents. PII Tools must process a broad range of documents, so NER must not fail on "weird" input.

- **Accurate**: obvious, no one want to see wrong results.
- **Fast on a CPU**: PII Tools must process terabytes of data quickly. Specialized GPU hardware is only used during model training, not during model usage (inference).
- **Control of model behaviour**: every machine learning system makes mistakes, so at least we want direct control over what kind of mistakes it makes. Not all NER errors are created equal, from the user perspective, some are more embarrassing / costlier than others. We also require a robust way to implement and evaluate updates.

How NOT to build a practical NER

The most effortless way to do NER is to grab an existing pre-trained model, and just use that.

If you ever tried this, you probably found out quickly this leads nowhere in practice – the models are too brittle, trained on artificial datasets, with narrow domain-specific preprocessing. They fail (worse: fail unpredictably) when applied to real documents

dataset (the so called state-of-the-art of SOTA dataset).

The second popular way to create a NER is to fetch an already prepared model and tune it on your own data. For the NER task of "I give you text, you give me the positions of all person names within the text", training data is scarce and usually not very diverse. The state of the art models are transformers (https://jalammar.github.io/illustrated-transformer/), a special kind of neural networks, which are quite slow and memory hungry (although there is active research to make them faster). Manually labelling huge amounts of text is costly too. So this was not the way for us.

Bootstrapping a NER

Our approach is incremental, bootstrapping from existing datasets and open source tools:

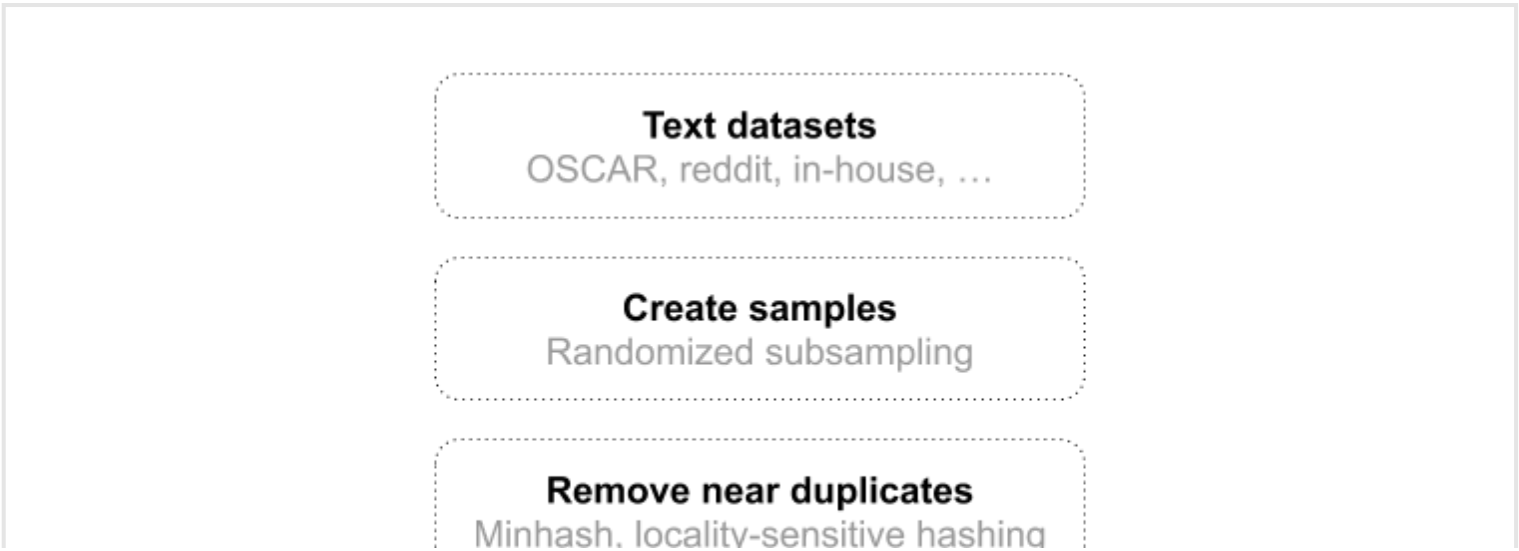
1. Prepare a large and diverse **unlabelled** dataset automatically.
2. **Annotate** the corpus automatically using multiple fast but low-accuracy open source tools.

5. Optimize NER for **production**: squeeze model size and performance with Tensorflow Lite, DAWG, mmap and other tricks.

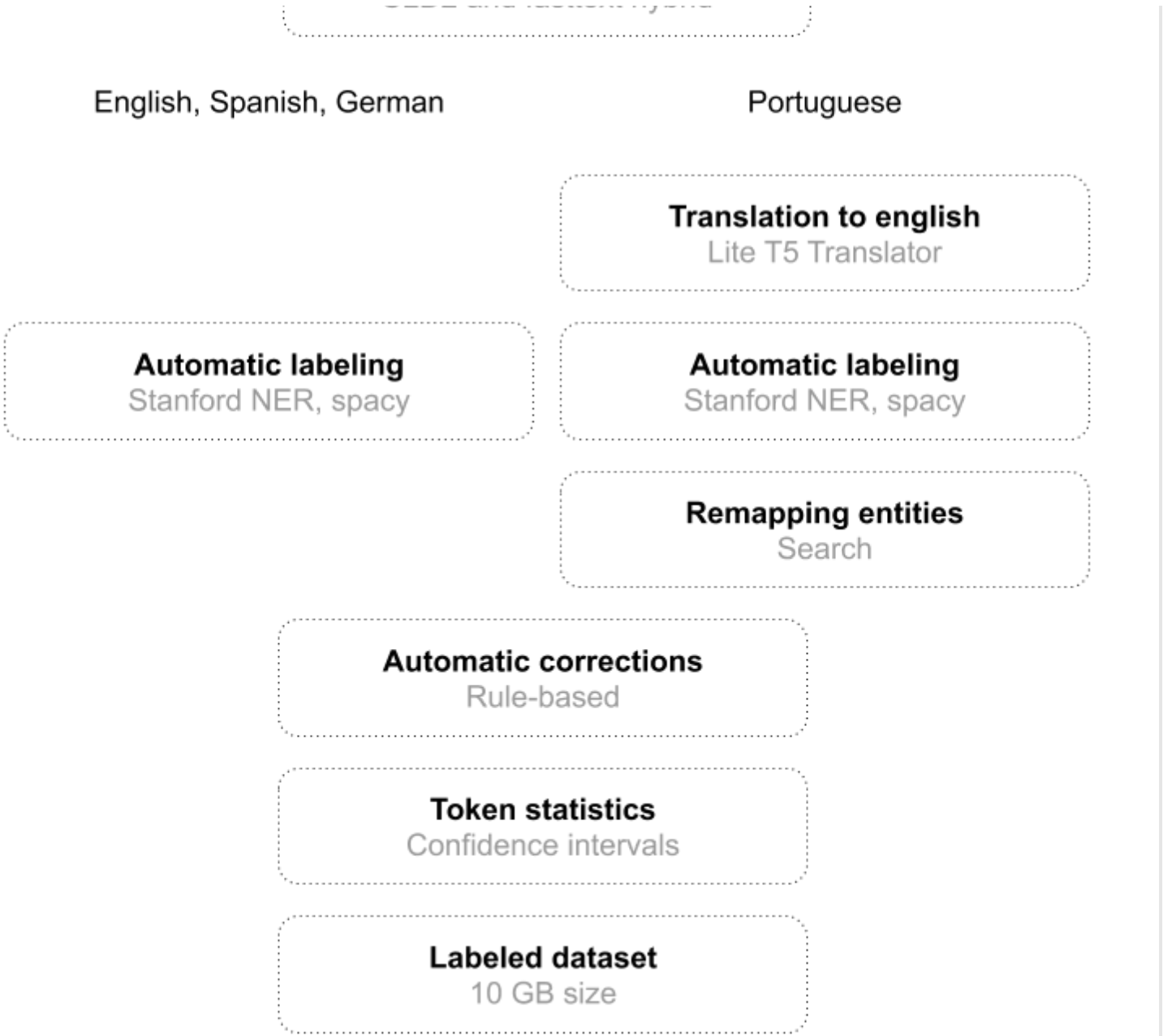
Let's go over these steps one by one.

Diverse dataset

Garbage-in, garbage-out. Quality of training data is essential, so we spent a lot of effort here. We built a powerful pipeline, combining automated and manual labelling in a feedback loop:



(https://pii-tools.com/)



We use various text sources, to include the kinds of contexts one could find person names in:

- OSCAR corpus (https://oscar-corpus.com/): a large set of crawled text from web pages by Common Crawl (https://commoncrawl.org/).
- Reddit (https://files.pushshift.io/reddit/comments/): forum comments from reddit.com (https://www.reddit.com/).
- Gutenberg (https://web.eecs.umich.edu/~lahiri/gutenberg_dataset.html): books from the Gutenberg Project (https://www.gutenberg.org/).
- Enron emails (https://www.cs.cmu.edu/~enron/): published corporate emails.
- Wiki Leaks (https://wikileaks.org/): documents and articles on various topics.
- Extensive gazetteers of person names and cities. Cities (home addresses) and person names appear in similar contexts, with similar formatting, and are a common source of confusion for NER models in general. So paying close attention to cities and addresses pays off in name detection.
- In-house dataset of documents, part of PII Tools, proprietary. Includes Excel spreadsheets, PDFs (incl. for OCR), Word and various corner-case formats and contexts.

Aside: Language identification

One might think that language identification in text is an easy problem. But just like with NER, existing tools have painful issues like slow speed or low accuracy across different languages. See also Radim's paper on Language Identification (https://scholar.google.cz/citations?user=9vG_kV0AAAAJ&hl=en#d=gs_md_cita-d&u=%2Fcitations%3Fview_op%3Dview_citation%26hl%3Den%26user%3D9vG_kV0AAAAJ%60).

After internal testing and evaluation of available solutions, we ended up creating a hybrid of simple & fast n-gram language detector based on CLD2 (https://github.com/CLD2Owners/cld2) and fastText language identification (https://fasttext.cc/docs/en/language-identification.html"). This is what we used for the "corpus language filtering" mentioned above.

End of aside

The next part of the dataset building process is to automatically annotate the raw corpora above. That is, mark the position of person name in each corpus document. We use existing NER engines for this initial bootstrap step. This technique is sometimes called "weak labeling".

Basically Stanford NER and spaCy were the only viable options here since the dataset was quite big (see NER performance charts (https://pii-tools.com/detect-person-names-in-text/#performance)). Even for these two relatively fast libraries, it took several days to annotate the entire corpus.

So now we had a large amount of text annotated with person names. It contained a lot of mistakes though – neither Stanford NER's nor spaCy's out-of-the-box models work very well outside of their trained domain. So another step was to clean the data, remove the obvious (classes of) errors.

That was a bit of manual work, we had to go through a subset of NER results, identify the most common patterns of errors, write transformation rules, re-annotate. For instance, no, “Ben & Jerry” is not a person, and “Stephen Crowd carpenter” is a

We use this dataset preparation strategy for English, Spanish and German. For Portuguese (for Brazil's LGPD) we had to resort to a little different approach since we found no existing NER with good enough performance. So instead, we translated Portuguese texts with the Lite-T5 translator (https://github.com/unicamp-dl/Lite-T5-Translation) to English, recognized named entities with Stanford NER and spaCy and then simply remapped them back into the original text. The automatic translation took a few weeks but required no manual work.

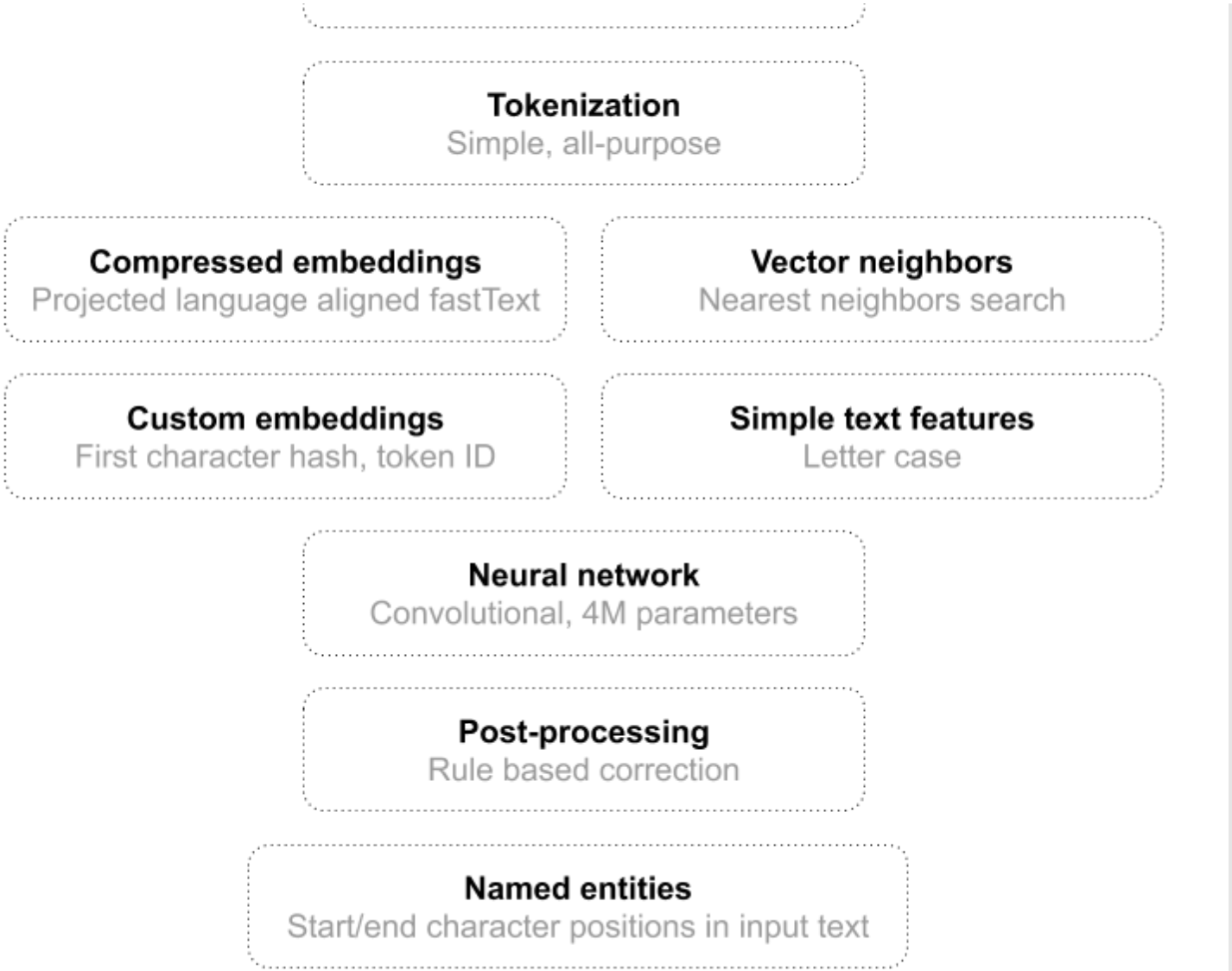
In the end we built 10 GB (20M training samples) of annotated text, with surprisingly high quality considering it was produced mostly automatically.

Neural network model

PII Tools is written in Python, and so is the model pipeline. We used Tensorflow to define and train our production model on the annotated corpus above. The pipeline uses several kind of features, to maximize accuracy.

As mentioned earlier we used a lightweight convolutional neural network as the heart of our NER model. The model accepts a text, tokenizes it to a sequence of tokens (token positions) and spits out a sequence of name positions. A position is an offset within the original document, so we can exactly **match and remediate** detected names later.

(https://pii-tools.com/)



INFERENCE ON A TRAINED MODEL.

enough space to be smart. We had the greatest success with a simple tokenizer that recognizes 4 token groups: alphabetic sequences, whitespace, digits and other characters. Being smarter than that always provided worse results.

Input Features

Another critical point here is how to choose token features. Features have to be fast to compute and as informative as possible. The best performance-information trade off was achieved by combining:

- simple "letter case" features;
- precomputed language aligned fastText vectors, projected with umap (https://github.com/lmcinnes/umap) to a lower dimension;
- trained short token embedding to cover tokens not in precomputed vectors;
- trained vector embedding of the first character of a token to differentiate between token groups and to differentiate characters inside the “other” token group.

For the pretrained embedding vectors we have to keep a dictionary of relevant tokens and their vectors, which could potentially be fairly big, consuming too much RAM. Thus for each language we picked a subset of:

training dataset), and

3. tokens with tightest confidence interval to be a part of a name.

Furthermore, we included another set of tokens whose vectors were in close neighborhood of the tokens identified above, and assigned them the vector of their respective neighbor. This optimization trick increased our dictionary size 2-4x depending on the language, while costing negligible extra space.

We then massaged features through a convolutional network with standard 1D convolutions and leaky ReLU activation units. By deepening the network to 10+ layers we can achieve a reception field of 20+ tokens, which is enough for this task. This way we avoided costly LSTM, GRU or transformer's attention units. Outputs of the neural networks are tags indicating whether the current token is the beginning, middle part or an end of a person name, or a non-name token.

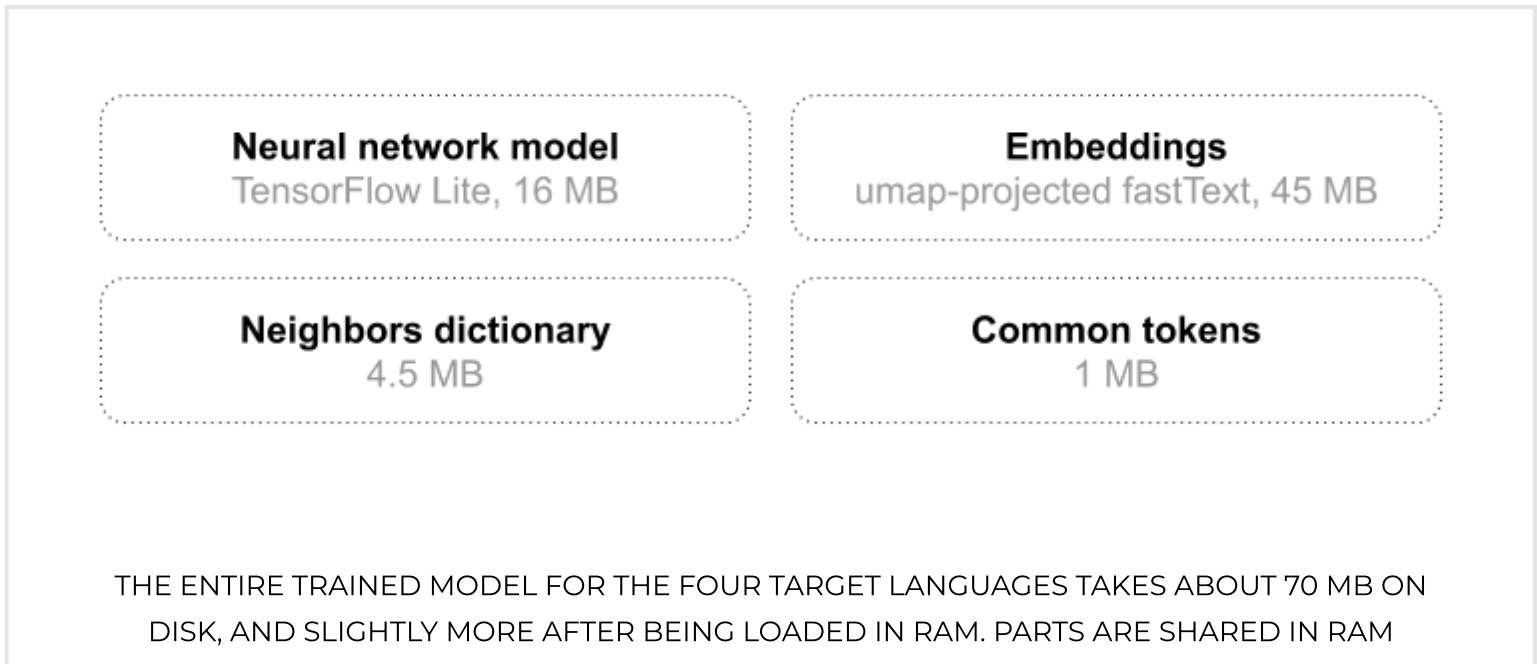
Post-processing

To make sure the model output is reasonable, we then apply a few lightweight post-processing steps, such as ignoring person names which do not contain any statistically-strong name token, or where all tokens are common words. For this, we

model learns not to do this type of mistake again. In practice, we find that having post-processing steps is highly desirable anyway, both as a quick approximation of a retrained model and also as a fail-safe paradigm, for data patterns that CNNs have trouble picking up from a limited number of training samples.

Model minimization

If you recall, accuracy is only one of our design goals. Being practical in terms of computational speed (CPU) and memory footprint (RAM) is equally important.



THE ENTIRE TRAINED MODEL FOR THE FOUR TARGET LANGUAGES TAKES ABOUT 70 MB ON DISK, AND SLIGHTLY MORE AFTER BEING LOADED IN RAM. PARTS ARE SHARED IN RAM

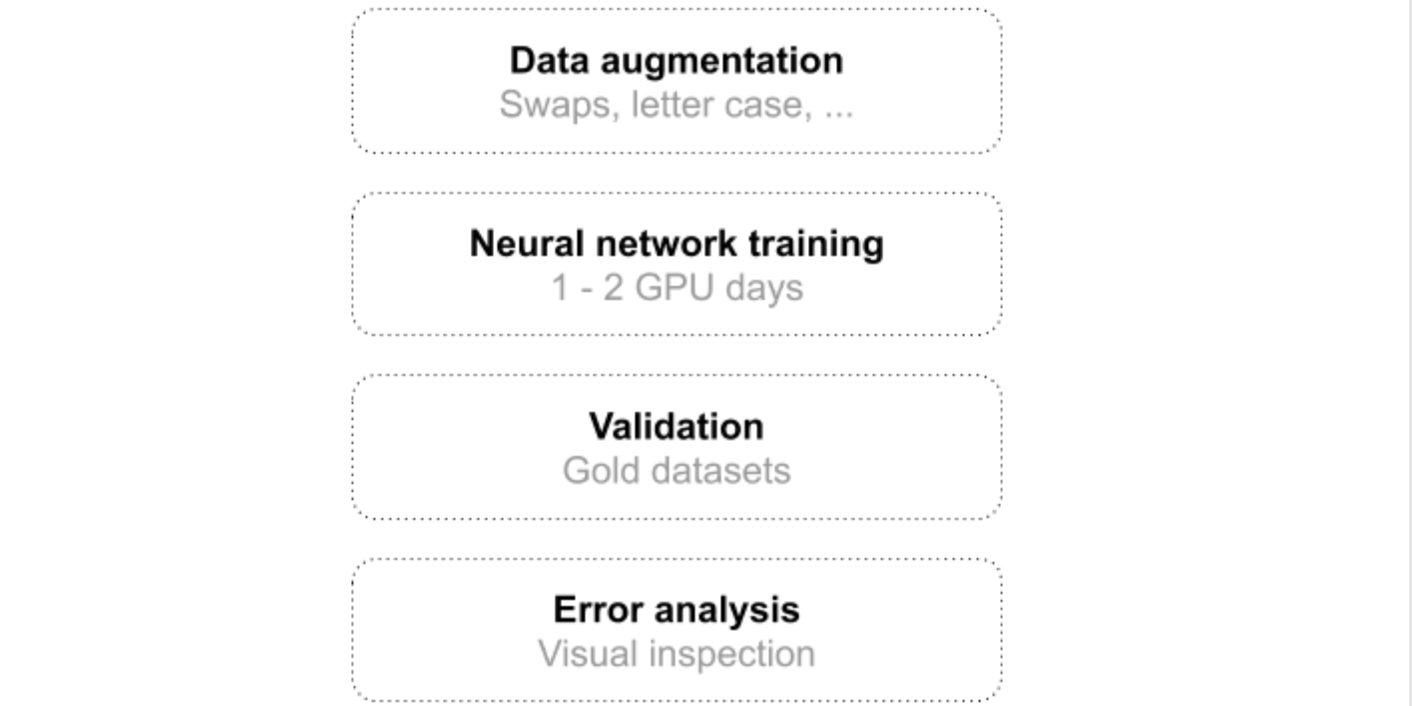
To compress the necessary model dictionaries and save even more RAM and CPU, we use DAWG (https://github.com/pytries/DAWG), an excellent FSA (finite state automaton)-based storage. Our (versioned) model representation takes up roughly 70 MB on disk, which includes:

- the trained Tensorflow Lite convnet model,
- precomputed vector embeddings for 300k+ tokens, for all languages combined,
- a token dictionary, and
- the corpus token statistics.

In short, everything that's needed to run the name detection in production.

PII Tools uses parallelization for performance, so some of these data structures are shared in RAM between worker processes using mmap (https://en.wikipedia.org/wiki/Mmap). This allows further memory reduction on heavy-load systems with many scan workers.

Training and optimization

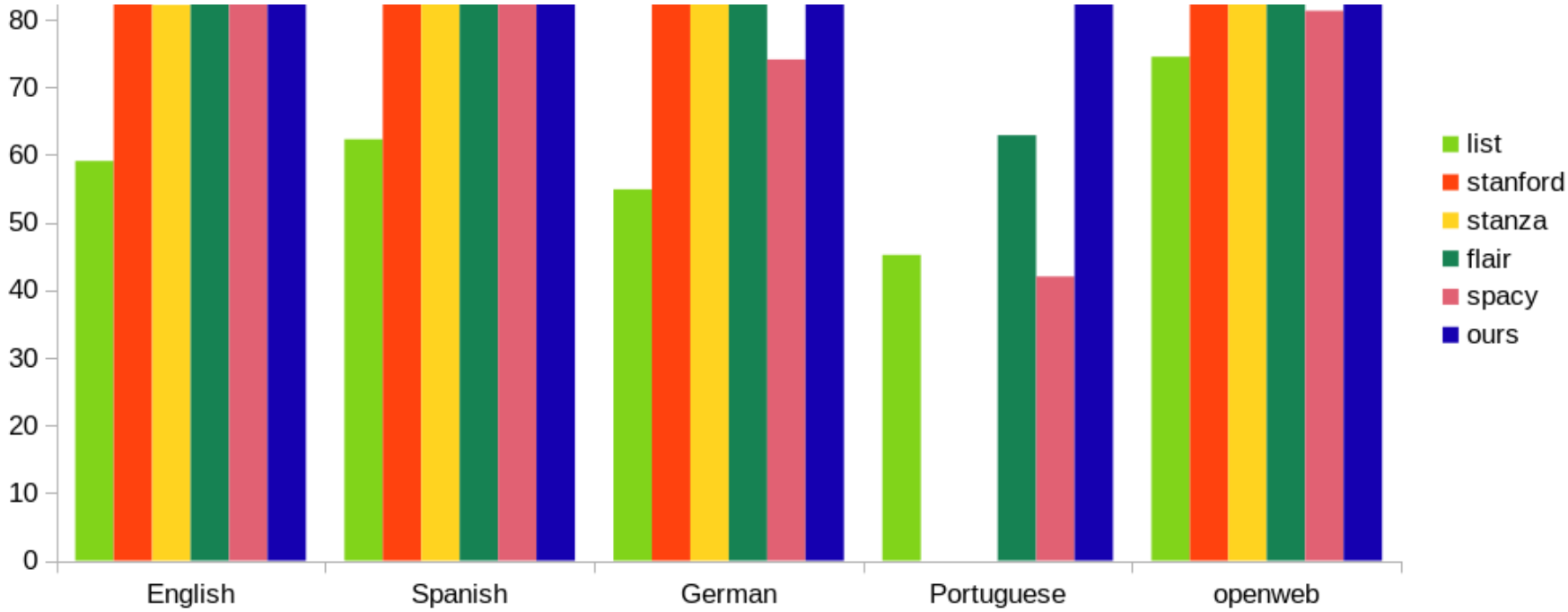


NER TRAINING PIPELINE.

Training the whole model takes 1 to 2 days on a low-end server hardware with 8 CPUs and 1 GPU. 10 GB of training data has to be processed while applying data augmentation on-the-fly.

Another trick we found useful is to inject more information into training via multi-tasking. The neural network is not only trained on predicting token tags, but it also has to guess the language of text in the receptive field, guess how likely a computed token is inside a name or predict tags for tokens in close neighborhood. This acts as a regularizer for the training and the model internalizes additional information without needing to explicitly provide it on input.

Conclusion



MONEY SHOT FROM THE PREVIOUS POST – F1 SCORES FOR DIFFERENT DATASETS AND NER SOLUTIONS. VISIT NER RESULTS (HTTPS://PII-TOOLS.COM/DETECT-PERSON-NAMES-IN-TEXT/) FOR MORE BENCHMARKS AND SPEED & RAM COMPARISONS.

PII Tools is a commercial product, so while we cannot release the full source and data, I hope you at least found my tips here useful. If you'd like to see the NER model in action, alongside our other detectors for passport scans, home addresses, PII, PCI and PHI information (https://pii-tools.com/how-to-evaluate-pii-discovery/), contact us with your business case.

(https://pii-tools.com/)

tools.com/schedule-a-demo/).

Download our AI whitepaper

Detecting Personal Names in Text

Your name

Your email address

Agree to Privacy Policy (/privacy-policy/). We never share your data with third parties. Unsubscribe any time.

Download whitepaper

Tags:CCPA (https://pii-tools.com/tag/ccpa/), CNN (https://pii-tools.com/tag/cnn/), convolutional neural networks (https://pii-tools.com/tag/convolutional-neural-networks/), deep learning (https://pii-tools.com/tag/deep-learning/), detect names (https://pii-tools.com/tag/detect-names/), GDPR (https://pii-tools.com/tag/gdpr/), HIPAA (https://pii-tools.com/tag/hipaa/), KVKK (https://pii-tools.com/tag/kvkk/), LGPD (https://pii-tools.com/tag/lgpd/), named entity recognition (https://pii-tools.com/tag/named-entity-recognition/), names in text (https://pii-tools.com/tag/names-in-text/), NER (https://pii-tools.com/tag/ner/), person names (https://pii-tools.com/tag/person-names/), unstructured text (https://pii-tools.com/tag/unstructured-text/)

Meet the author

(https://pii-tools.com/)

Jan Procházka's bio:

Data science engineer at PII Tools in one life. Accredited physiotherapist in another.

Schedule a demo

Your Name (required)

Your Email (required)

(https://pii-tools.com/)

Send

Recent Posts

PII De-Identification vs. Masking vs. Redaction (https://pii-tools.com/pii-de-identification-vs-masking-vs-redaction/)

The New CPRA Umbrella Covers HR Data (https://pii-tools.com/the-new-cpra-umbrella-covers-hr-data/)

Regular Audits: Do You Really Need Them? (https://pii-tools.com/regular-audits-do-you-really-need-them/)

Do They Even Matter?—The 3 Largest GDPR Fines To Date (https://pii-tools.com/do-they-even-matter-the-3-largest-gdpr-fines-to-date/)