Annotated Job Ads with Named Entity Recognition

Felix Stollenwerk[*] [†] AI Sweden

Niklas Fastlund Arbetsförmedlingen

Anna Nyqvist[*] ICA Gruppen

Joey Öhman[*] AI Sweden

Abstract

We have trained a named entity recognition (NER) model that screens Swedish job ads for different kinds of useful information (e.g. skills required from a job seeker). It was obtained by fine-tuning KB-BERT. The biggest challenge we faced was the creation of a labelled dataset, which required manual annotation. This paper gives an overview of the methods we employed to make the annotation process more efficient and to ensure high quality data. We also report on the performance of the resulting model.

1 Introduction

Natural language processing (NLP) has been subject to vast improvements in recent years. Transformer-based models like BERT (Devlin et al., 2019) have pushed the boundaries of performance, leveraging transfer learning in NLP through unsupervised pre-training of large language models and subsequent fine-tuning for a specific downstream task. Swedish models are readily available (Malmsten et al., 2020) and have great potential to revolutionize the way text data is handled in the public and private sector in Sweden. However, despite those impressive leaps forwards, in order to create fine-tuned models that can be employed for real-world use cases, labelled datasets are still required. These datasets usually need to be manually created and this often needs to be done in-house, for reasons such as that the annotation requires domain-expert knowledge or the data contains sensitive information. In our case, the goal was to create a dataset of annotated job ads in order to train a NER model which allows one to analyze job ads on a large scale. We aimed to detect a wide range of interesting information, initially represented by 16 classes. An example excerpt of an annotated job ad is shown in Fig. 1.

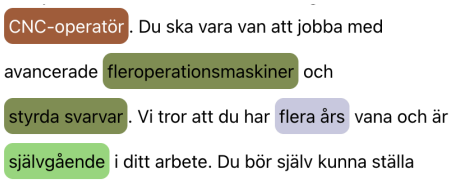

Figure 1: Example of an annotated job ad. The classes used in the excerpt are called `JOB_TITLE` (brown), `SKILL_HARD` (dark green), `SKILL_SOFT` (light green) and `EXPERIENCE_TIME` (light violet).

The difficulty of the manual annotation process depends strongly on the use case and the complexity of the dataset one intends to create. A simple annotation task can often be handled straight forwardly. In the simplest possible set-up, the dataset is evenly split up between annotators who simply go through their respective shares and annotate each sample exactly once, see the left panel of Fig. 2. However, in situations where the annotation task is more challenging, a simple strategy may be insufficient. There are mainly two sorts of potential problems:

A. The effort to reach a sufficient amount of data (usually such that a model performance plateau is approached) exceeds the budget (in terms of time or money). This can be seen as a problem of inefficiency which requires acceleration of the annotation or model learning process.

B. The data quality does not match the requirements. Annotators will always provide inconsistent or incorrect annotations to some extent. The process generally becomes more error-prone with the complexity of the task. The linguistic level of the text, the number of classes to choose from, and whether the class definitions are distinct, are examples of factors that contribute to complexity.

Our annotation project was affected by those issues

[*] Work done while working at Arbetsförmedlingen

[†] Correspondence to `felix.stollenwerk@ai.se`

as well. In the next section, we describe a few methods that we used to mitigate them.

2 Methods to improve the annotation process

Our starting point is iterative annotation in batches. In contrast to what is shown in the left panel of Fig. 2, the labelled dataset is hence scaled up gradually. This procedure, which we will refer to as *default strategy*, allows one to train a model on intermediate data after each step and to study the development of its performance on a held-out test dataset. It is the basis for the more advanced methods to be described in the following subsections.

2.1 Bootstrapping

In the default strategy, the model has a passive role and the mere purpose of training it is to track its progress in performance. At each iteration, the annotation is done from scratch, i.e. the annotators start from pure text data without any annotations. Bootstrapping essentially means that one lets the model predict on a batch of data before it is sent to the annotators. This way, the annotation task shifts towards the correction of model predictions. At the beginning of the iterative process, this is not much of an advantage as the model performance is usually rather poor. However, later in the process, the model predictions are of valuable help and have the potential to significantly simplify and speed up the annotation process. A risk that comes with bootstrapping is that annotators may trust the model predictions too much (in later stages of the iterative process) and fail to thoroughly correct them. This can lead to imperfect annotations and a feedback loop where the model is fed its own predictions.

In order to make the most of bootstrapping and maximize the acceleration of the annotation process, it is imperative to train a near-optimal model after each step. This is challenging, however, as the amount of data changes after each step, from tiny in the beginning to rather large in the end. As the optimal number of training epochs heavily depends on the amount of data (Zhang et al., 2021; Mosbach et al., 2021), using the standard method of fine-tuning (Devlin et al., 2019; Wolf et al., 2020) implies that a hyperparameter search is required to find the optimum, every time the model is trained. To avoid the considerable effort that comes with this, we used our own fine-tuning method based on early stopping and a custom learning rate schedule (Stollenwerk, 2022). It automatically trains for a near-optimal number of training epochs, and can be applied throughout the whole bootstrapping process.

2.2 Active Learning

Another well-established method to accelerate the annotation process is active learning, see e.g. (Settles, 2009; Olsson, 2009) for a general introduction. The default process samples each batch randomly from the available raw data. In contrast, active learning aims to select the batch of data that is most informative for the model. The idea is that this enables the model to learn faster, often resulting in a steeper learning curve (see e.g. (Yuan et al., 2020)). However, note that this expectation is mainly based on empirical results, and there is no guarantee that active learning helps in every single use case. In fact, there are also cases where active learning has shown to have no or even a negative effect (see e.g. (Schein and Ungar, 2007)).

There are many different variants of active learning. They can be categorized according to whether they use uncertainty or diversity sampling (or both), which family of models they can be applied to (general models, neural networks, language models), and their very definition of informativeness. Since there were no published results on active learning in conjunction with Swedish language models when we started working on this project, we did some experiments ourselves (Öhman, 2021). We simulated the iterative annotation process using publicly available ground truth datasets for named entity recognition in Swedish, and tried out some of the active learning methods using different (acquisition) batch sizes. We observed a modest positive effect of active learning (Öhman, 2021), and decided to use the simple query-by-uncertainty method for our own job ad annotation. Note that this course of action is based on the assumption that the effect of active learning on our own job ad dataset is similar to what we observed in our experiments with the aforementioned ground truth datasets. However, it is important to note that there is no guarantee for this, especially as our job ad dataset differs from the ground truth datasets by having a very specific theme and a more complex class system. Moreover, there is no way to verify in hindsight that the assumption was correct without enormous additional effort[1].

[1] One would have to annotate a second, randomly sampled

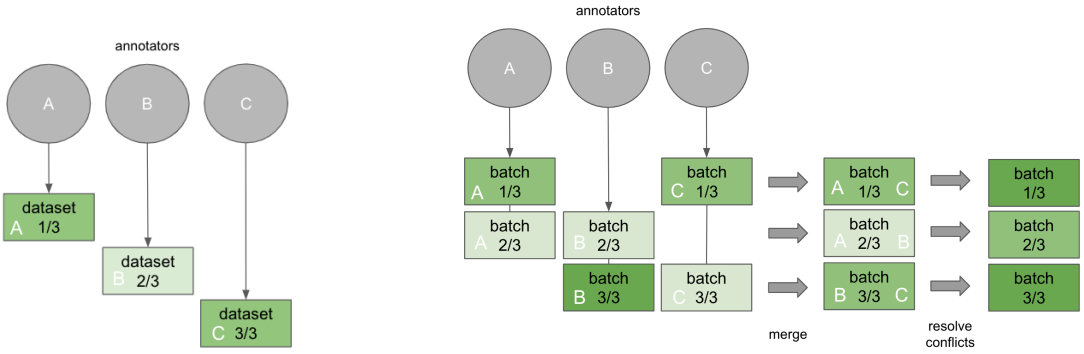

Figure 2: *Left:* Simple annotation strategy. The whole dataset is divided into N parts for N annotators. Each annotator takes care of their part independently, without any double-checking. The color of the parts represents the quality of the annotations (dark = high quality, light = low quality) and the figure shows an example. *Right:* Our (more sophisticated) annotation strategy. Firstly, each sample is annotated by two different annotators. After merging, the conflicts are discussed and resolved in a collective effort. A concrete example is shown in Fig. 3.

2.3 Annotation Cross-checking

As explained in Sec. 1, the task to annotate job ads is quite hard. This became apparent early in our initial experiments, where we found a very low inter-annotator agreement. This made a simple annotation approach (cf. Fig. 2) unsuitable for the task as it would have led to inconsistent annotations and thus low quality data. Inevitably, some sort of cross-checking was needed to overcome the problem. We aimed for an approach where every job ad was annotated by at least two annotators. Moreover, we wanted it to include a part where all annotators collectively annotate and have the opportunity to discuss examples together. The idea behind this is that it helps the annotators to learn how to annotate consistently, and to get feedback in order to successively improve the annotation guidelines and refine the class system (more on the latter in Sec. 2.4). Last but not least, we aimed for a procedure that—despite involving the aforementioned elements that add complexity—was as time-efficient and simple as possible. We came up with the annotation strategy depicted in the right panel of Fig. 2. Each batch of data is annotated in three stages:

1. *Individual annotation*: The batch is divided into N parts, where N is the number of annotators. Each part is duplicated and assigned to two different annotators. Each annotator works on their two parts individually. The result is that each part of the batch comes with two different annotations.

2. *Merge (automated)*: Each part of the batch is "merged". To begin with, this means that its two annotations are compared. The entities that coincide are simply kept. If the two annotations differ, for instance as a word was tagged by annotator A but not by annotator B, the word in question is turned into a special entity with the label `???`, indicating that there is disagreement that needs to be resolved. In the case of overlapping annotations, both variants are kept and assigned the special label. After this step, there is only one (merged) annotation for each part of the batch.

3. *Collective annotation*: The annotators go through the merged parts of the batch together, discuss the cases where the annotation differed, and resolve the conflicts. The result is considered ground truth data.

An example excerpt of a job ad annotated according to this procedure is illustrated in Fig. 3.

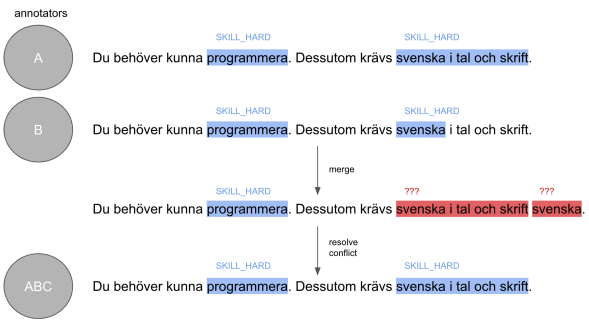

Figure 3: Excerpt of a job ad that gets annotated separately by two annotators A and B. The result is merged before the conflicts are resolved together by all annotators.

dataset under the very same conditions as the original, actively sampled dataset.

During the whole process, we tracked the inter-annotator agreement using various metrics, see (Nyqvist, 2021) for details.

2.4 Class System Adjustments

The initial class system that we chose before starting the annotation process was designed to match the needs of stakeholders and the downstream applications that would eventually make use of the NER model predictions. The system was quite ambitious with 16 different classes, and it was by no means obvious to us how well the model would learn to predict them. With this in mind, we incorporated the opportunity to dynamically adjust the class system during the iterative process in our framework. After each iteration, we used the following sources to evaluate how well the process worked for each class: 1. discussion after the collective annotation session (qualitative feedback) 2. inter-annotator agreement (quantitative feedback) 3. model performance, e.g. in terms of f_1 score and the confusion matrix (quantitative feedback). If we found one or more classes to be difficult and saw no signs of improvements over multiple iterations, we adjusted the class system. Such an adjustment is defined as a mapping of the problematic classes to (presumably) unproblematic classes, including `O`. In particular, one may simply drop a class (`A` $\rightarrow$ `O`), incorporate a class into another, existing class (`A` & `B` $\rightarrow$ `B`) or merge two classes to form a new class (`A` & `B` $\rightarrow$ `C`). Note that a class system adjustment always represents an irreversible[2] simplification. Our class system was adjusted several times and we ended up with 10 instead of 16 classes. For more details and quantitative results, we again refer to (Nyqvist, 2021).

3 Model Performance

A total of 260 job ads were annotated by 5 people using the outlined methods. 200 of them were used to fine-tune KB-BERT, while 30 were held out as a validation dataset for hyperparameter search (see Sec. 2.1). The remaining 30 job ads served as a test dataset to assess the model's performance. Following standard practice for NER, we used the f_1 score on the entity level as the main evaluation metric. We determined it separately for each individual class and used the micro-average as a single number to quantify the model's general performance. Both model training and evaluation were done using the *nerblackbox* package (Stollenwerk, 2021). The results are shown in Tab. 1.

We find a reasonable performance on most of the classes, specifically the ones that are most important for downstream applications. However, the numbers are not comparable with those typically reported on NER benchmark datasets, see e.g. (Malmsten et al., 2020). We attribute this to the increased complexity of the problem, i.e. the large number of classes with intricate, overlapping definitions (that even humans struggle with, as evidenced by the low inter-annotator agreement). An additional, particular challenge are classes that contain many multi-word entities, most prominently the `JOB_TASK` class. The model often correctly identifies entities, but struggles to predict their boundaries accurately. This is reflected by the significantly better f_1 score on the token-level that is observed for those classes, see Tab. 1.

Class	entity	token
SKILL_HARD	0.77	0.87
SKILL_SOFT	0.78	0.76
JOB_TITLE	0.90	0.88
JOB_LOCATION	0.76	0.76
EMPLOYER_TITLE	0.84	0.86
JOB_TASK	0.48	0.74
EDUCATION_DEGREE	0.62	0.85
JOB_TIME	0.66	0.85
EXPERIENCE_DURATION	0.62	0.80
EMPLOYER_BENEFIT	0.73	0.63
micro-average	**0.72**	**0.79**

Table 1: Performance of our NER model. The numbers represent the f_1 score evaluated on the test dataset, on the entity and token level. The classes are ordered by estimated importance for downstream applications.

4 Conclusions

This case study highlights some typical challenges one may face when annotating real world data. We covered some advanced methods that helped us to accomplish the task. The resulting NER model shows a reasonable performance in consideration of the intricacy of the problem.

Acknowledgements This work was done in conjunction with the Vinnova-funded research project *Språkmodeller för svenska myndigheter* (Language models for Swedish authorities).

[2]The existing annotated data can easily be mapped from an old to a new class system. However, if an additional batch is annotated using the new class system, it can not be mapped back to the old class system.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

Martin Malmsten, Love Börjeson, and Chris Haffenden. 2020. Playing with words at the national library of sweden – making a swedish bert.

Marius Mosbach, Maksym Andriushchenko, and Dietrich Klakow. 2021. On the stability of fine-tuning bert: Misconceptions, explanations, and strong baselines.

Anna Nyqvist. 2021. Bootstrapping annotated job ads using named entity recognition and swedish language models. http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-305901.

Fredrik Olsson. 2009. A literature survey of active machine learning in the context of natural language processing.

Andrew Schein and Lyle Ungar. 2007. Active learning for logistic regression: An evaluation. *Machine Learning*, 68:235–265.

Burr Settles. 2009. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison.

Felix Stollenwerk. 2021. nerblackbox: a python package to fine-tune transformer-based language models for named entity recognition. https://github.com/flxst/nerblackbox.

Felix Stollenwerk. 2022. Adaptive fine-tuning of transformer-based language models for named entity recognition.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics. https://www.aclweb.org/anthology/2020.emnlp-demos.6.

Michelle Yuan, Hsuan-Tien Lin, and Jordan Boyd-Graber. 2020. Cold-start active learning through self-supervised language modeling.

Tianyi Zhang, Felix Wu, Arzoo Katiyar, Kilian Q. Weinberger, and Yoav Artzi. 2021. Revisiting few-sample bert fine-tuning.

Joey Öhman. 2021. Active learning for named entity recognition with swedish language models. http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-303866.