End-to-end NLP System Building

Balasubramanyam Evani
bevani@andrew.cmu.edu

Neel Pawar
npawar@andrew.cmu.edu

Abstract

We present the use of an end-to-end Natural Language Processing (NLP) system that performs a custom Named Entity Recognition (NER) on scientific text. Our method of NER is different from conventional NER in that it only tags the entities directly described or compared in the respective scientific papers. The project starts with scraping for a large corpus of scientific papers, which we further curate to identify best quality and most influential papers, so as to facilitate some few-shot learning qualities in the model. We further create a dataset from these papers by manually labelling them with 7 different tags and creating a labelled dataset in the CoNLL format. We finetune and compare three different models on this dataset, the original BERT and two variants of SciBERT on this dataset. We observe improvements by the use of SciBERT over the baseline BERT architecture, and compare the F1 scores of the models to analyze their performance on two different test datasets. We further present our findings from this end-to-end NLP system.

1 Introduction

Named Entity Recognition is the extraction and labeling of specific entities in a piece of text, typically a writing. In this project, we examine performing the task of NER on scientific text using SciBERT-cased, SciBERT-uncased and BERT. We create our own dataset from scratch by scraping PDFs, curating best papers, extracting text, cleaning text, labelling text manually with 7 different tags, where tags are put only if a specific piece of text directly relates to the paper being analysed (eg. a direct model being compared to the model in paper, or a dataset directly being trained on by the model discussed in the paper). We further create a dataset from this labelled text, in the CoNLL format. We curate best papers to help our goal of learning from less data, and it helps us achieve good performance with a handful of papers. We further analyze the use of the three different pretrained models and compare and contrast them using F1 score metrics. Finally, we perform a significance testing to confirm in-depth performance metrics of these models and choose the best performing model. We present our findings, lessons learned and deductions as well as methodologies used to improve our pipeline and generalize better.

2 Dataset Creation

2.1 Collection of raw data

We collected the raw PDFs using arXiv API Access and ACL APIs. We queried the arXiv API for papers related to relevant topics such as transformers, BERT models, deep learning models, attention, RNN and LSTM models with frequent use of an NLP tag with the queries. We queried the most recent papers from ACL Anthology, from ACL-2022, EMLP-2021, NAACL-2022 to get the most recent and relevant work in NLP, since we observed newer models relative to Transformers to be more relavant to the majority of scientific literature in NLP. We then created a csv of all the paper links. Once we collected the csv of paper links, we used a script to download PDFs from the script, which gave us around 100 papers. To select the PDFs to use, we collected the most influential papers which

Preprint. Under review.

were most cited, as these had both, the proof of concept and content quality, as well as widespread use and applications in other papers, which mainly followed these papers. Prioritizing these main papers would help our model understand the big picture better, as most other papers follow the ideas based on these papers. We also added some popular and good quality papers like the original BERT paper to make sure we were capturing the essence of the sample space and not missing any key papers due to an inadequate query. Two of the papers added by this process were the BERT and BERTology papers. Use of such a partly curated dataset enhanced the model's few-shot learning capabilities, increasing F1 score by around 0.05 and allowed us to leverage our small team size by prioritizing good quality data over large quantities of data.

2.2 Extraction of text from raw data

To extract text data from raw PDFs, there were two main steps that we followed, first we extracted the text using the scipdf parser. We change its configuration to run on an AWS EC2 instance. Next, we implemented pre-processing functions to clean the PDFs parsed by scipdf. Specifically, we removed any URLs contaminating the data and removed reference numbers from the text. We then wrote the content to a text file, generating one text file per paper. Each text file was paragraph formatted, separating paragraph by newline pairs. We thus built a set of 11 text files from the most expressive papers we could choose, some of which included the BERT, GPT-3, Attention, RoBERTa papers. We further split this into a train validation and test sets and annotated the same, as mentioned in the further sections.

2.3 Tokenization

We used Allen-AIs SciBERT Scivocab tokenizer since this was most relevant given our goal, model selection and target vocabulary / paper content and corresponding SciBERT cased tokenizer for the SciBERT cased model. Functions of this tokenizer are: Encode the text data into encodings, tokenizing (splitting strings in sub-word token strings), converting tokens strings to ids and back, and encoding/decoding (i.e., tokenizing and converting to integers), adding new tokens to the vocabulary in a way that is independent of the underlying structure, managing special tokens (like mask, beginning-of-sentence, etc.): adding them, assigning them to attributes in the tokenizer for easy access and making sure they are not split during tokenization.

2.4 Data split and annotation

We further split the data into three parts, train, validation and test data. We performed a 70 - 20 - 10 train - val - test split. Test data taken was 10 percent since we had access to additional held out test data from the SciNER test set and the total quantity of test data seemed empirically sufficient to express the generalization capabilities of the model on two different sets of test data. We also picked a paper different from the BERT based papers, the XLNet paper for testing, to ensure that we were testing on significantly unseen data that the model would not have learned inadvertently from its training.

For annotation, we annotated the generated 11 text files by tagging them with entity names using Label Studio, which We picked text file format for annotation as it was the most straightforward and easy way that aligned the most with the CoNLL format, and thus the model inputs. We picked the most broad and general papers for annotation. The reason being that we wanted our model to learn best from even small data, and hence we followed the best quality papers in terms of clear goals and language, proof of concept, sound comparisons and meticulous experiments. We used Label Studio for our annotation purposes given it's graphical user interface and ease of annotation.

We tagged 7 different entities for this task: MethodName, HyperparameterName, HyperparameterValue, MetricName, MetricValue, TaskName, DatasetName. The annotations were done such that entities directly related to the premise of the scientific text being annotated.

To ensure that our model was not just a good language model performing NER but a good scientific NER model, we used a model trained on scientific texts, the SciBERT. Hence we did not use non-annotated data explicitly, but used a model that had seen a variety of non-annotated data and nuances in technical writing that branched from regular articles that BERT models are pretrained on. The model thus had context on scientific papers and technical writing due to its original pretraining on scientific text.

Finally, the dataset was created in the CoNLL format for every text file from Label Studio and we ran a script that combined these to form a full dataset that we further split into the train, validation and test sets.

3 Modeling

We tested three different models across our modeling trials. The baseline architecture was BERT-base, but we saw performance gains by using Allen-AI SciBERT, a BERT model pretrained on scientific data. We initially experimented with SciBERT-cased as cased models are usually the preferred choice for Named Entity Recognition. We then switched to SciBERT-uncased to compare the performance with the cased version. Further, we used the BERT-base-cased model to have a level-ground comparison with SciBERT-cased. We observed better performance with the use of SciBERT, increasing F1 score over BERT-cased by 0.035 on our XLNet-based test dataset and 0.076 for the test SciNER dataset. We see improvements by using the cased model, which is expected behavior on the NER task, as cased models generally perform better when doing NER. For finetuning these models, we use a model for token classification from these pretrained models and finetune on the CoNLL dataset.

3.1 Model choice

The model choice of SciBERT was based on two primary reasons:

1. Obtaining large-scale annotated data for our task was very challenging, expensive and prone to error, especially for a small team size. This model helped a leverage small datasets to achieve comparable results to a model trained on a larger dataset that would typically be used for our task.
2. To leverage the pretraining task of SciBERT on the SciVocab dataset, specifically comprised of a sample of 1.14M papers from Semantic Scholar, and thus enhance the model's generalization capabilities on scientific text more specific to this text than Wikipedia and BookCorpus text that BERT is traditionally pretrained on.

Given this rationale, we experimented with the SciBERT, and observed improved results.

3.2 Model Dataloader

We use an NER dataset that serves encodings and labels to a consumable format by the NER model. Collation during batching is done by a data collator from HuggingFace. We use a dataloader that takes an index, and outputs encodings and labels for the corresponding training example. These can then be collated into batches and then served to the model.

3.3 Training setup

We use the following set of hyperparameters for training the NER model: batch size 32 across train and validation sets, number of epochs as 25, weight decay of 0.01, learning rate of 2e-5. We started with a learning rate of 1e-5, a low value picked specifically because we were finetuning an already pretrained model. We later adjusted this to 2e-5 because we saw speed gains and better convergence.

We use the HuggingFace Trainer API for its well suited compatibility with our use-case, and this can be customized as required, for example, in future work if we are looking to add weighted loss functions to handle the class imbalance. The SciBERT-cased model gave the following raw metrics during training:

4 Model Testing

4.1 Preliminary testing on models

We compare the F1 scores of the three models for a preliminary analysis on performance. We observe that the SciBERT-cased model outperforms the uncased model and the BERT model.

Metric	Value
Epochs	25
Training accuracy	0.976
Training F1 score	0.479
Validation accuracy	0.955
Validation F1 score	0.455

Table 1: SciBERT raw training statistics

Model F1 score summary			
	BERT-base-cased	SciBERT-uncased	SciBERT-cased
Custom test set	0.441	0.494	0.476
SciNER test set	0.322	0.3997	0.435

Table 2: NER Model performance statistics

4.2 Significance testing on models

To have a more concrete comparison and to see strengths and weaknesses of each model, we performed a significance test on the three models. To start with straightforward metrics, the F1 scores revealed that the SciBERT-cased model performed best, followed by SciBERT-uncased, followed by BERT-cased. Further, the F1 scores on our own custom XLNet dataset revealed that SciBERT-uncased had the best performance, followed by SciBERT-cased and then by BERT-base-cased

This indicates that even with cased version use of BERT, both variants of SciBERT were a better option on held out data. This can be positively attributed to the pretraining of SciBERT on scientific data, thus proving its more apt use for the Scientific NER task.

Figure 1: Overall Performance on SciNER (F1 scores)

Figure 2: Overall Performance on Custom dataset (F1 scores)

We observe the patterns in F1 scores to compare and contrast the models in terms of their strengths and weaknesses. Although these noticeably differ by a small margin, we evaluate these results to see which model performs best in most criteria, and when to use which model.

We saw several patterns in the significance testing on our custom test data, but unambiguously, with an unknown task, it is almost certain that either of the SciBERT versions would perform sufficiently well as we did not observe any drift or erratic performance. Owing to the nuances observed, We draw a comprehensive comparison by use cases between the model architectures based on these metrics.

Figures 3 and 4 show the significance tests done on the SciNER dataset and the custom dataset respectively. Tables 3 and 4 attempt to summarize the findings from these significance tests in terms

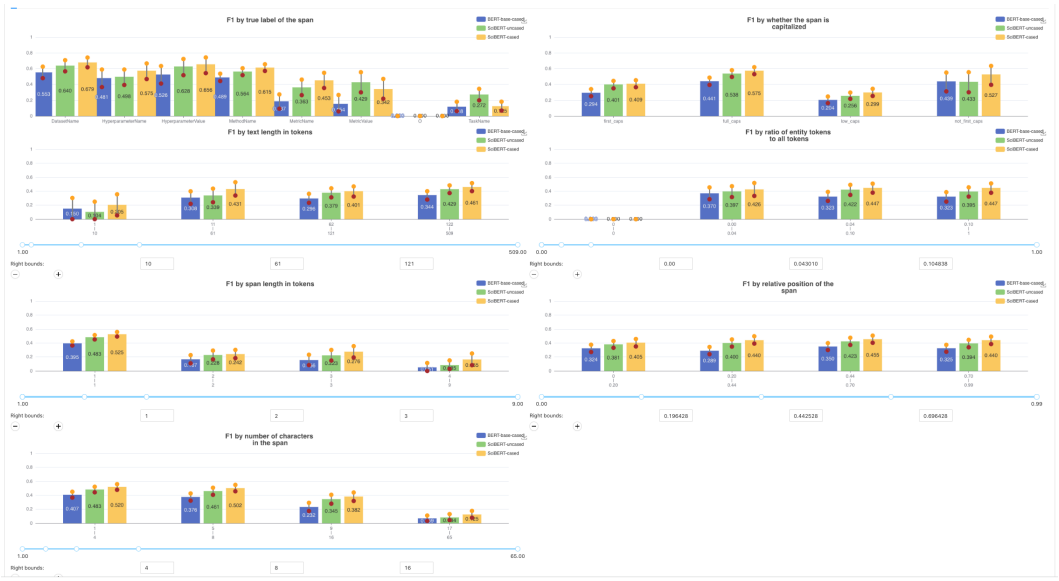

Figure 3: In-depth significance testing on SciNER

F1 score by:	Best performance	Intermediate performance	Worst performance
True label of span	SciBERT-cased	SciBERT-uncased	BERT-base-cased
First letter of span capitalized	SciBERT-cased	SciBERT-uncased	BERT-base-cased
All letters of span capitalized	SciBERT-cased	SciBERT-uncased	BERT-base-cased
Text length in tokens	SciBERT-cased	SciBERT-uncased	BERT-base-cased
Ratio of entity tokens to all tokens	SciBERT-cased	SciBERT-uncased	BERT-base-cased
span length in tokens	SciBERT-cased	SciBERT-uncased	BERT-base-cased
Relative position of span	SciBERT-cased	SciBERT-uncased	BERT-base-cased
Number of characters in span	SciBERT-cased	SciBERT-uncased	BERT-base-cased

Table 3: In-depth significance testing on SciNER

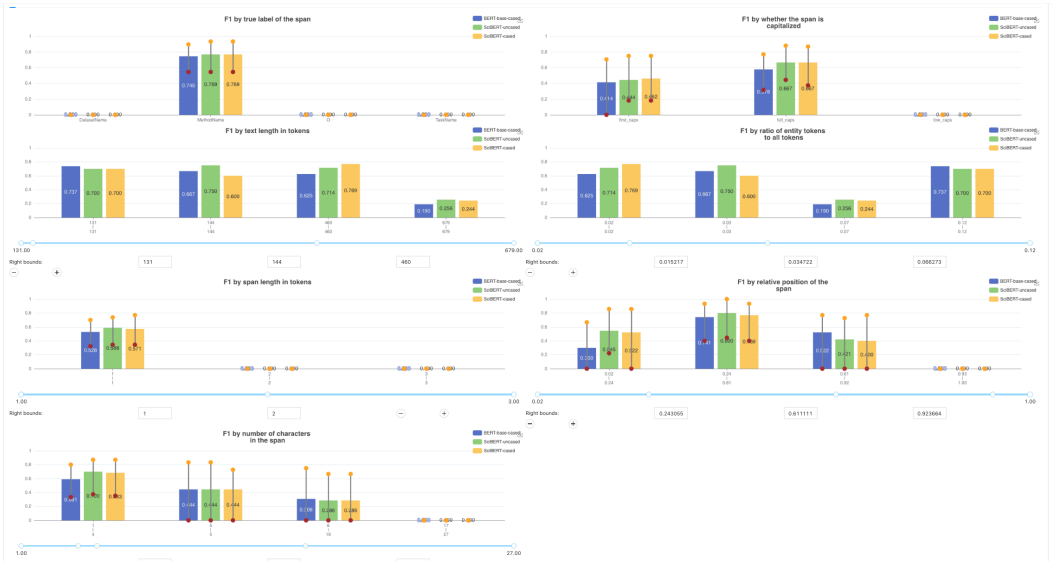

Figure 4: In-depth significance testing on Custom dataset

of order of performance of models and decide which model was better on specifically which kind of text for the NER task.

F1 score by use case:	Best model
First letter of span capitalized	SciBERT-cased
Low ratio of entity tokens to all tokens	SciBERT-cased
Shorter text lengths	BERT-base-cased
Longer text lengths	SciBERT-cased
Short span length in tokens	SciBERT-uncased
Large relative position of span	BERT-base-cased
Small number of characters in span	SciBERT-uncased

Table 4: In-depth significance testing on Custom dataset

4.3 p-values

In null-hypothesis significance testing, the p-value is the probability of obtaining test results at least as extreme as the result actually observed, under the assumption that the null hypothesis is correct. A very small p-value means that such an extreme observed outcome would be very unlikely under the null hypothesis. We found the p-values by doing a pairwise comparison of the 3 models used. The p-value comparisons are illustrated in Table 5. Please note that the 0.0000 values are indicative of very low p-values as the Explainaboard testing platform does not detect below a very small thrshold. $p < 0.05$ is often used as a threshold for statistical significance.

The p-value for SciBERT was repeatedly better than BERT-base on the custom dataset. There was not a significant p-value on the SciNER dataset. Among the two SciBERT architectures, the uncased version most often had the higher p-value, except on SciNER where the cased version had marginally high p-value. In essence, it seems from our tests that SciBERT-uncased is a reliable architecture to go for in most scenarios.

Model p-value summary		
Model pair	SciNER test set	Custom test set
BERT-base-cased v/s SciBERT-uncased	0.0000	SciBERT better by 0.1910
BERT-base-cased v/s SciBERT-cased	0.0000	SciBERT better by 0.2130
SciBERT-uncased v/s SciBERT-cased	Cased better by 0.0100	Uncased better by 0.2300

Table 5: Pairwise p-value comparison

5 Conclusion

From our experiments, we learned important lessons with the creation of an end to end NLP system performing Named Entity Recognition. We observed the technical considerations, constraints and nuances in creating a labelled dataset from scratch. We further saw the effect of pretraining on scientific text helps improve accuracy metrics, and that cased models would be better suited for the NER task. Given the amount of data and resources available, we successfully performed different methods to improve learning capabilities of the model from limited data. In future work, it would be helpful to resolve class imbalance by use of a simple weighting function that can perform weighting based on inverse frequency of labels. This can be passed to the training function and NLL losses to make the system more accurate and agnostic to class imbalances. We hope this project adds value to NER tasks done in a specific manner and on NER tasks on scientific data, and our methodologies add value to academic and research work done in this area.