Improving on Stab and Gurevychs Argument Component Identification and Classification

Hugo Meinhof, 815220

March 7, 2024

Abstract

In this article I am presenting new models for classifying and identifying argument components, as defined by Stab and Gurevych[SG17]. I propose a new model for identifying and classifying in one step, as well as models for doing so separately, in a pipeline. I show that my models for the pipeline outperform the baselines, as well as Stab and Gurevychs previous models, on their persuasive essays corpus.

1 Introduction

1.1 Previous Work: Stab and Gurevych 2017

In 2017, Stab and Gurevych[SG17] revisited their own work on argumentation mining from 2014. Dissatisfied with the existing corpora at the time, they decided to make their own, which they extended in 2017. On that corpus of persuasive essays, they annotated argument components and their argumentation structures. With these annotations, they trained models to built and test a pipeline. "However, as demonstrated by Levy et al. (2014) and Goudas et al. (2014), the identification of argument components is more complex in other text genres than it is in persuasive essays. Another potential issue of the pipeline architecture is that wrongly classified major claims will decrease the accuracy of the model because they are not integrated in the joint modelling approach. For this reason, it is worthwhile to experiment in future work with structured machine learning methods that incorporate several tasks in one model (Moens 2013)." I have trained such model, full_labels, which sadly doesn't reach the f1 of the SuG ILP model yet.

1.2 Goal of my work

For this paper I have attempted to surpass the models for argument component identification and classification, as shown in Figure 1. In general, the identification regards the question of where an argument component, like MajorClaim, will be located, without knowing what type it will be. Then, the classification task is the labeling of what kind of argument component we are dealing with. Stab and Gurevych trained models for each task, and my goal was to improve on them. All of the code I wrote can be found on github at "Theoreticallyhugo/argument_mining_SuG". The models are available on Hugging Face, at "Theoreticallyhugo/longformer-", followed by the respective name of the model. ELABORATE AND REFERENCE FIGURE – eplain what a span is

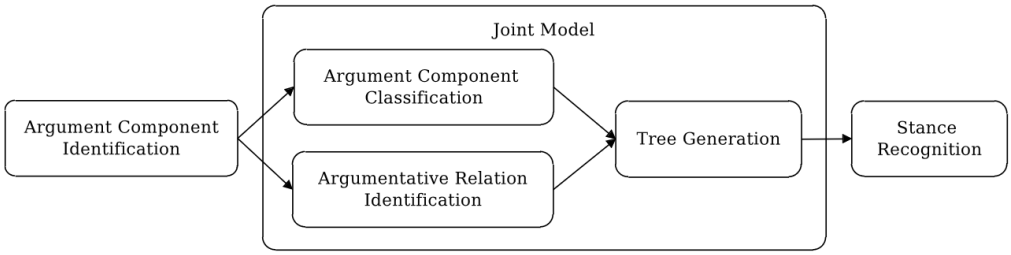

Figure 1: Architecture of the argumentation structure parser

1.3 My previous work

test

2 Finding Argumentation Components

I have trained and compared multiple models and methods for both identifying and classifying argumentation components. From that process five different models emerged in various stages of simplification and mutual support. All of them are finetuned versions of longformer[BPC20], the long document transformer.

2.1 Full labels

Just like Stab and Gurevych recommended in 2017, I tried to train a model that combined the tasks of argument component identification and classification. For that I used the labels "B-MajorClaim", "I-MajorClaim", "B-Claim", "I-Claim","B-Premise", "I-Premise", and "O". The first Part of each label carries information for the argument component identification, whilst the second part does so for the classification. Full labels performs worse than Stab and Gurevychs ILP-balanced model[SG17]. However this is not an apples to apples comparison, as there is no 5-fold cross-validated evaluation data available for the pipeline, with the ILP model, meaning that it may perform worse than the full labels model.

2.2 Spans

Once again, just like Stab and Gurevych in 2017, I came to realise that splitting up the argument component identification and classification could yield overall improvements. The spans model takes care of the first part of the pipeline, the identification. It uses the labels "B", "I", and "O". Reducing the number of labels from seven to three, and limiting the task to identifying the spans, greatly improved the models performance, compared to full labels.

2.3 Simple

With the performance increase of the spans model in mind, I created the simple model. It uses the labels "MajorClaim", "Claim", "Premise", "X_placeholder_X", and "O".The X_placeholder_X label however is never annotated in the entire dataset, and hence never used. It is discussed in the section 2.7.2.
The idea of this model is the exact same as with the simple model, except this time it's for the classification task instead of identification. The lack of information regarding where spans are, would be made up for in the post processing. There, for each span that has been found

by the spans model, the pipeline would check what labels were annotated. The most commonly annotated label within the span would then be applied to the entire span.

```
‘‘we’’, ‘‘should’’, ‘‘attach’’, ‘‘more’’, ‘‘importance’’, ‘‘to’’, ‘‘cooperation’’
‘‘O’’, ‘‘Claim’’, ‘‘Claim’’, ‘‘Premise’’, ‘‘Premise’’, ‘‘Claim’’, ‘‘Claim’’
```

This synthetic example shows a span without its context. Within it, Claim is the most commonly annotated label, and so the post processing would consider the entire span to be of the type “Claim”.

2.4 Sep tok full labels

This model takes advantage of another way in which the spans model can support a model that follows. Not only can the information on where spans are be used during post processing, but also before the inference or training. The preprocessing for this model places class and separator tokens in the text data, in order to signal where spans are. They are learned during pretraining and defined in the huggingface documentation of longformer as follows:

> cls_token: <s>
> “The classifier token which is used when doing sequence classification (classification of the whole sequence instead of per-token classification). It is the first token of the sequence when built with special tokens.” [Bel20]
>
> sep_token: </s>
> “The separator token, which is used when building a sequence from multiple sequences, e.g. two sequences for sequence classification or for a text and a question for question answering. It is also used as the last token of a sequence built with special tokens.” [Bel20]

So, the model uses the same labels as the full labels model, “B-MajorClaim”, “I-MajorClaim”, “B-Claim”, “I-Claim”,“B-Premise”, “I-Premise”, and “O”, but differentiates itself through the text it is being fed. This means that for this model, input looks like in this synthetic example:

```
‘‘<s>’’, ‘‘we’’, ‘‘should’’, ‘‘attach’’, ‘‘more’’, ‘‘</s>’’,
‘‘O’’, ‘‘B-Claim’’, ‘‘I-Claim’’, ‘‘I-Claim’’, ‘‘I-Claim’’, ‘‘O’’
```

Since injected cls and sep_token increase the total number of tokens, they are assigned the “O” label, in order to keep the labels aligned.

2.5 Sep tok

Both the models simple (2.3) and sep tok full labels (2.4) showed improvements over the initial full labels model (2.1). Therefore, this model was created, taking advantage of both ideas. It uses the labels “MajorClaim”, “Claim”, “Premise”, “X_placeholder_X”, and “O”, just like simple, and processes texts with insertet separation tokens, like sep tok full labels. It comes to no surprise then, that this model performs better than simple and sep tok full labels.

2.6 Pipeline

The pipeline handles argument component identification and classification individually, and uses the best model for each task. Running the pipeline starts with inference on the input strings, with

the spans model. The inference is handled by the huggingface pipeline object. Its output is then used to inject class and separation tokens into the input strings. Due to the problems described in 2.7.1, the following assumptions are made in postprocessing. For any span that begins with "I" instead of "B", the first "I" label is considered the begin. Any span that is shorter than four tokens is considered an uninterpretable leftover, and hence ignored. spans shorter than four are ignored. no further postprocessing. TODO. The resulting texts are then fed into the sep tok model, once again handled by a huggingface pipeline. Its outputs are used in conjunction with the inputs. First, the cls and sep tokens are used to identify the spans within each text. Then, all tokens that lie within each span are found, counted, the most frequent for each identified and then applied to the entire span. If a span is identified, but not labeled, because sep tok labeled all tokens within it as "O", it is deleted.

2.7 Problems

2.7.1 Argument component identification

Distinguishing between "B" and "I" labels is supposed to allow proper identification of where spans begin and end. However, there are multiple ambiguous situations, which complicate the process.
If, for example, a span starts out with an "I" label, it is unclear whether this truly is where the span begins. Possibly this "I" label is correct, but should be preceeded by a "B" label, but it could also be the case that it isn't correct, and that the "I" should be a "B". There also are situations where after a span, there is one "O", followed by a couple of "I" labels. Here, a third possibilty is added, being that this gap should be filled with an "I", making it one long span.

Problem 1:

```
‘‘we’’, ‘‘should’’, ‘‘attach’’, ‘‘more’’, ‘‘importance’’, ‘‘to’’, ‘‘cooperation’’
‘‘O’’,  ‘‘I’’,      ‘‘I’’,      ‘‘I’’,     ‘‘I’’,          ‘‘I’’,  ‘‘I’’
```

Possible resolutions:

"O" label is begin, span starts one token earlier

```
‘‘we’’, ‘‘should’’, ‘‘attach’’, ‘‘more’’, ‘‘importance’’, ‘‘to’’, ‘‘cooperation’’
‘‘B’’,  ‘‘I’’,      ‘‘I’’,      ‘‘I’’,     ‘‘I’’,          ‘‘I’’,  ‘‘I’’
```

"O" label is correct, and first inside label should be begin instead

```
‘‘we’’, ‘‘should’’, ‘‘attach’’, ‘‘more’’, ‘‘importance’’, ‘‘to’’, ‘‘cooperation’’
‘‘O’’,  ‘‘B’’,      ‘‘I’’,      ‘‘I’’,     ‘‘I’’,          ‘‘I’’,  ‘‘I’’
```

Problem 2:

```
‘‘we’’, ‘‘should’’, ‘‘attach’’, ‘‘more’’, ‘‘importance’’, ‘‘to’’, ‘‘cooperation’’
‘‘B’’,  ‘‘I’’,      ‘‘I’’,      ‘‘O’’,     ‘‘I’’,          ‘‘I’’,  ‘‘I’’
```

Possible resolutions:

Skipped label is begin, making two spans, no gap

```
‘‘we’’, ‘‘should’’, ‘‘attach’’, ‘‘more’’, ‘‘importance’’, ‘‘to’’, ‘‘cooperation’’
‘‘B’’,  ‘‘I’’,      ‘‘I’’,      ‘‘B’’,     ‘‘I’’,          ‘‘I’’,  ‘‘I’’
```

Skipped label is inside, making one span, no gap

```
‘‘we’’, ‘‘should’’, ‘‘attach’’, ‘‘more’’, ‘‘importance’’, ‘‘to’’, ‘‘cooperation’’
‘‘B’’,  ‘‘I’’,      ‘‘I’’,      ‘‘I’’,     ‘‘I’’,          ‘‘I’’,  ‘‘I’’
```

Skipped label is correct and should be followed by begin, making two spans, with gap

```
‘‘we’’, ‘‘should’’, ‘‘attach’’, ‘‘more’’, ‘‘importance’’, ‘‘to’’, ‘‘cooperation’’
‘‘B’’,  ‘‘I’’,      ‘‘I’’,      ‘‘O’’,    ‘‘B’’,          ‘‘I’’,  ‘‘I’’
```

Additionally there is the issue of rogue labels, meaning that there sometimes are one, two, or three inside-label-long spans at seemingly random positions. For now, any span shorter than three is ignored, but considering the second problem, this could delete important parts or even entire spans, because they are split.

2.7.2 The “X_placeholder_X” label

For some reason CUDA kept crashing on the backwards pass, when the training for NER of longformer ran with an even number of labels. Therefore, the “X_placeholder_X” label was introduced. It is never annotated and never part of the models output, but fixes the issue with CUDA.

3 Training data

The foundation of the training data for this project consists of 402 student essays annotated by Stab and Gurevych. For each essay, the dataset includes information on where the core assertion of the text, otherwise known as the MajorClaim, supporting assertments, or the Claims, and the premises, that back up the thesis with statistics and other evidence, are located. The texts used were sourced from essayforum.com and selected randomly. No further information is known about the authors, but it is conceivable that English may not be their first language and they are likely learning it in an educational context. Stab and Gurevych subsequently annotated the essays with tokens and recorded the locations of spans.

Claim: semantische klasse, die eine behauptung aufstellt; stützt majorclaim MajorClaim: kernbehauptung(en) des textes prämissen: unterstützung/ untermauerung der Claims (zb statistiken)

“For finding the best-performing models, we conduct model selection on our training data using 5-fold cross-validation”

check this shit The essays were tokenized and stored where spans are located, what role they have, and which tokens belong to them. These data are created by Rene’s script, what was that. In order for models to learn from this, a dataset must be created that processes the data and sorts it comprehensibly to the model. This is the greatest task in training. Since all trained models are based on the same dataset, there is a common dataset for all of them. Many extraction, processing, and matching steps remain the same for all models. Differences basically only exist in the last processing step. This is handled by the different configs of the dataset. There is a separate config for each model, which bears the same name as the model. Since the models only differ in how the training data are processed, this also means that a training script can be used for all models, in which only the config needs to be adjusted. I also made sure that the configs bear the same names as the models, so that everything runs smoothly, providing a better explanation of the training scripts. However, this hasn’t always been the case. I started with a separate training script for each model. While this is not as adaptable as a script for all, which can be adjusted via command line arguments, it has the clear advantage that a single model can be trained and tested first, without having to keep others in mind.

The corpus consists of 402 essays, downloaded, and randomly selected, from essayforum.com.

4 Evaluation

For reproduceability, every training and inference was seeded. Furthermore, all evaluation was 5-fold cross-validated, just like Stab and Gurevych did with their models, on the same data. However all evaluation is done on the raw inference, without any postprocessing.

With a Makro-f1 score of 0.77, the full labels model, running both the argument component identification and classification in one go, already performs quite well. Considering that it's a fast and simple single step solution, it certainly has its perks. It also needs to be mentioned, that the model surpasses the majority and heuristic baselines for both argument component identification and classification as presented by Stab and Gurevych. From epoch 7 onward, we get to see very little improvements. Epoch 14 is getting the best results, marking the beginning of a mostly flat curve, followed by very little change in either direction. Interestingly there seems to be no overfitting within the 20 epochs of training.

For the argument component identification task, Stab and Gurevych were happy to announce that their models performance came very close to their human annotators. I am equally happy to announce that the spans model has surpassed both their models, as well as human performance (see table 2). Since Table 8 and Table C.1 in [SG17] report different scores for CRF all features, both are included in Table 2.

The model takes only three epochs to perform very well. Considering scores, there isn't really a point in training beyond five epochs, but from epoch 15 onward, the curve is almost flat. Once again there are no signs of overfitting within the 20 Epochs of training. DOES IT BEAT SOTA?

Out of the three models for argument component classification, the simple model performs the worst. Nevertheless it achieves better scores than the baselines, the full labels model, and comes somewhat close to the performance of Stab and Gurevychs ILP-balanced model. Comparing simple to their standalone model for classification, SVM all features, it surpasses the scores, as well as the baselines.

In epochs six through 14, the model gets very similar scores, only to be improved upon by about 0.01 five epochs later. Therefore, an f1 of 8.811 can be claimed, but 0.799 at epoch six would be more sensible.

Here too, the scores eventually settle, without really worsening again, indicating that there is no overfitting yet.

The sep tok full labels model is the first to outperform Stab and Gurevychs ILP-balanced, without being the best in the group yet. Its scores fluctuate throughout the epochs, with the most significant improvements having taken place till epoch eleven and eventual stabilising towards epoch 18.

Sep tok is the best performing model so far, showing better scores than the baselines, other longformer based, and Stab and Gurevychs models (see table 3). It gains the most performance until about epoch eight, with fluctuation, but eventual improvements until epoch 16. This is also the only model that shows performance dips after that epoch, meaning that there is no flattening of the curve, as observed with the other models.

"Our identification model yields good accuracy and an α_U of 0.958 for identifying argument components. Therefore, it is unlikely that identification errors will significantly influence the outcome of the downstream models when applied to persuasive essays." [SG17]

Following the reasoning of Stab and Gurevych, I am assuming that the pipelines performance is very close to that of the downstream model, both for my, as well as their models.

Makro-f1	full labels	spans	simple	sep tok full labels	sep tok
01	0.378	0.798	0.668	0.525	0.754
02	0.522	0.889	0.743	0.670	0.810
03	0.635	0.902	0.777	0.795	0.840
04	0.730	0.908	0.788	0.816	0.854
05	0.743	0.912	0.792	0.818	0.863
06	0.752	0.909	0.799	0.830	0.861
07	0.757	0.911	0.796	0.829	0.867
08	0.759	0.911	0.798	0.833	0.870
09	0.759	0.912	0.796	0.841	0.868
10	0.763	0.914	0.801	0.841	0.871
11	0.765	0.911	0.800	0.845	0.872
12	0.766	0.913	0.800	0.841	0.866
13	0.768	0.912	0.803	0.839	0.870
14	**0.770**	0.913	0.800	0.842	0.873
15	0.767	**0.915**	0.806	0.845	0.874
16	0.770	0.914	0.807	0.844	**0.876**
17	0.771	0.914	0.809	0.845	0.874
18	0.770	0.914	0.809	**0.846**	0.871
19	0.771	0.915	**0.811**	0.846	0.875
20	0.770	0.915	0.810	0.847	0.874

Table 1: 5-fold cross-validation of the macro-f1

Argument Component Identification	Makro-f1
Stab und Gurevytch	
Human upper bound	0.886
Baseline majority	0.259
Baseline heuristic	0.628
CRF all features	**0.849 or 0.867**
Meinhof	
spans	**0.915**

Table 2: Argument Component Identification (5-fold cross-validation as in Table C.1 of [SG17])

Argument Component Classification	Makro-f1
Stab und Gurevych	
Baseline majority	0.257
Baseline heuristic	0.724
SVM all features	**0.773**
ILP-balanced	**0.823**
ILP and CRF pipeline	no-eval
Meinhof	
full_labels	0.770
simple	0.811
sep_tok_full_labels	0.846
sep_tok	**0.876**
full_pipe	*TO-DO*

Table 3: Argument Component Classification (5-fold cross-validation as in Table C.2 of [SG17])

5 Training

All training was done using longformer[BPC20] and the Hugging Face transformers library. The dataset is a Hugging Face dataset and the code for the training and dataset is public, as well as the model weights. The corpus has been made by Stab and Gurevych [SG17], is publicly available and provided by the University Darmstadt. However, I am not allowed to publish, retransmit, display, redistribute, reproduce or commercially exploit the data in any form, as stated by the license agreement. Therefore the dataset requires acquiring the corpus separately.
The models are evaluated at each epoch for 20 epochs, five times each for the 5-fold cross-validation. All evaluation files are saved in the respective models directory. A separate script collects all evaluation files, calculates the cross-validation and saves the scores to the meta_data directory for human evaluation. An additional script compiles the macro-f1 scores into the code to build a LaTeX table.
Training has been run on a single Nvidia 4090.

References

[Bel20] Iz Beltagy. *Hugging Face Longformer Documentation*, 2020.

[BPC20] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer, 2020.

[SG17] Christian Stab and Iryna Gurevych. Parsing argumentation structures in persuasive essays. *Computational Linguistics*, 43(3):619–659, September 2017.