Argument Component Identification im Stile von Stab und Gurevych

Hugo Meinhof, 815220

March 2, 2024

Abstract

some abstract stuff goes here Stab und Gurevych[SG17]

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. on these annotations, they trained models to built and tested a pipeline. "However, 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." meaning that they didnt evaluate the entire pipeline, as they expect to get the same results as the models, running on gold data. This is a questionable practice, expecially considering that they're focussing on the accuracy, and not f1 here. cant find the accuracy referenced here "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 modeling 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 doesnt 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 surpass them. ELABORATE AND REFERENCE FIGURE – eplain what a span is

1.3 my previous work

test

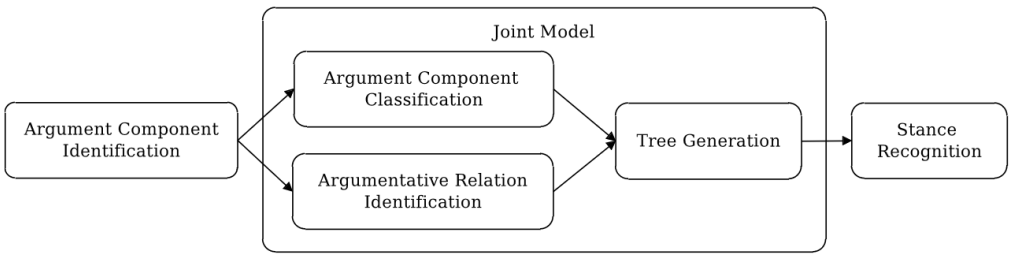

Figure 1: Architecture of the argumentation structure parser.

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. I discuss it in the section 2.7. The idea of this model is the exact same as with the simple model, except this time its 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’’

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. spans shorter than four are ignored. no further postprocessing. TODO. The resulting texts are then fed into the sep tok model, one again handled by a huggingface pipeline. Its outputs are used in conjuction 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 labled all tokens within it as "O", it is deleted.

2.7 problems

- B and I labels are cool and all, but what use are they, when a span of labels begins with I, or when a random word is dropped from a span - i get cuda errors when running NER at an even number of labels. hence i am using X_placeholder_X.

3 trainingsdaten

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.

quelle: https://aclanthology.org/J17-3005.pdf

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

über baselines schreiben!

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

die essays wurden tokenisiert und gespeichert wo sich spannen befinden, welche rolle sie haben, und welche tokens dazugehören. diese daten werden von renes script erstellt was war das. damit modeelle damit lernen können, muss ein dataset erstellt werden, welches die daten aufbereitet und dem modell verständlich sortiert. dies ist die größte aufgabe beim training. da alle traininerten modelle auf dem selben datensatz basieren, gibt es für alle zusammen ein gemeinsames dataset. viele extraktions, aufbereitungs, und matching schritte bleiben für alle modelle gleich. unterschiede gibt es im grunde nur im letzten aufbereitungsschritt. das wird von den verschiedenen configs des datasets gehandhabt. es gibt für jedes modell eine eigene config, die den selben namen trägt, wie das modell. da sich die modelle nur darin unterscheiden wie die trainingsdaten aufbereitet sind, bedeutet das auch, dass ein trainingsscript für alle modelle verwendet werden kann, in dem nur die config angepasst werden muss. ich habe zudem darauf geachtet, dass die configs die selben namen tragen wie die modelle, damit alles reibungslos abläuft bessere erklärung des trainings scripts. das war jedoch nicht schon immer so. angefangen habe ich mit je einem trainings script pro modell. das ist zwar auf der einen seite nicht so anpassbar wie ein sript für alle, welches über command line arguments angepasst werden kann, hat jedoch auf der anderen seite den klaren vorteil, dass so ein einzelnes modell erstmal trainiert und ausgetestet werden kann, ohne dabei andere im hinterkopf behalten zu müssen.

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

4 evaluation

5 training

Makro-f1	full labels	spans	simple	sep tok full labels	sep tok
4	0.579	0.898	0.769	0.749	0.843
5	0.631	0.905	0.782	0.801	0.849
6	0.716	0.908	0.790	0.816	0.858
7	0.740	0.910	0.796	0.824	0.864
8	0.752	0.911	0.800	0.832	0.864
9	0.757	0.912	0.801	0.837	0.865
10	0.759	0.912	0.801	0.838	0.867
11	0.761	0.912	0.803	0.833	0.866
12	0.763	0.914	0.803	0.837	0.865
13	0.763	0.913	0.802	0.844	0.867
14	0.767	0.915	0.801	0.840	0.867
15	0.766	0.914	0.801	0.841	0.868
16	0.766	0.915	0.803	0.846	0.870

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**
Meinhof	
spans	**0.912**

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	no-eval
Meinhof	
full_labels	0.759
simple	0.801
sep_tok_full_labels	0.838
sep_tok	**0.867**
full_pipe	*TO-DO*

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

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.