Two Stage Aspect-Based Sentiment Analysis: Multi-Target extraction and Polarity Classification

Simone Caldarella (224434)
University of Trento
simone.caldarella@studenti.unitn.it
Project Repository

Abstract

This document represents the report of the project developed for the MS.C. course *Natural Language Understanding* at the University of Trento. The main goal was to develop an *Aspect-Based Sentiment Analysis* pipeline, divided in two sub-task: *Multi-Target Extraction* and *Target Polarity Classification*.

I. Introduction

Aspect-Based Sentiment Analysis aims to classify the polarity of a set of aspects (or targets) in a sentence, thus it belongs to the category of token-level sentiment analysis. Moreover, the task is composed of two sub-tasks: multi-target extraction and polarity classification. The latter one is solved in a classification fashion, using as inputs the information carried by the tokens that are related to the classified one. The former one is, instead, more complex and can be addressed in several ways. A first line of approaches involves BIO-tagging techniques of the input tokens, while a second line of works proposes a span-based target extraction approach. In this report will be presented my implementation of the pipeline, which follows the work published by Hu et al. [1], but coded from scratch following the paper. However, since differences in performances appeared, in one of the following section will be debated some of the major issues in their approach, that I spotted while I was comparing the performances of their model with mine. To do this, since their code couldn't run completely on colab, I relied on an Azure instances equipped with two GPUs Tesla M60, offered by the University (University of Trento). The rest of my code was instead run on Colab.

The sections of the paper are structured as follows. Firstly, in the Method will be explained the implementation and the procedure used to develop the two sub-tasks. The pre-processing of the dataset will also be addressed here.

Secondly, in the Issues will be presented the major problems of the approach proposed by [1], with comments about them.

Finally, in the Implementation and results will be discussed some implementation details followed by the results obtained from the developed pipeline. The tests performed will also be presented in the last part of this section.

II. Method

The pipeline developed to solve the task is based on the approach proposed by [1] in 2019 and published by *Proceedings of ACL*. In the next sub-sections will be explained the pre-processing of the dataset and the two steps of the method.

A. Dataset

The dataset used is SemEval2014 [2] for ABSA, and specifically the *laptop reviews* domain, since it contains also the *restaurant reviews* domain. This dataset consists of over 3k English sentences extracted from customer reviews of laptops. Experienced human annotators tagged the aspect terms of the sentences (first sub-task) and their polarities (second sub-task). Regarding the split between training and testing, for the former we have 3045 sentences, while 800 for the testing. The original dataset is provided in an XML format in which, for each sentence, all the targets are annotated with theirs start and end positions. However the most used version is an unstructured txt, that presents for each line a sentence and its annotations word by word as *WORD=T-(POS|NEG|NEU)|O*.
For this pre-processing part, I wrote a parser to convert targets label into starts/ends position labels. As word embedder, I used the *BertTokenizer* module to obtain the $input\_ids$ of each word and the $attention_mask$ of the sentences. I also added the [PAD] tokens to match the max_length parameter value and the Bert special tokens [CLS] and [SEP]. The final dataset created was a list of quadruplets with the following elements:

- Original sentence expressed as a list of tokens;
- Tokenized sentence;
- Two tensor with length max_length that represents if each possible position is a start or not and if it's an end or not;
- A list that contains, for each target, the start position, the end position and its relative polarity.

In this first part, I faced with one of the first issues of the approached by Hu et al., regarding the sentences with no targets, which seem to be excluded from the computations.

B. Multi-Target Extraction

The first sub-task is solved with a novel approach employing a modified *Question-Answering* model, followed by a custom heuristic in order to allow multi-target extraction. The idea is to extract, for each predicted target, its span start and end positions. The reason behind this is that reduces the search space compared to standard BIO-classification approach. Moreover, since we are in multi-target fashion and we don't know a priori the actual number of targets in each sentence, we require the heuristic to select the most relevants targets inside the predicted ones. This appears to be the most critic and slow part of the approach, but is a step required only for testing/inference.

More specifically, the extraction model is a modified versione of Bert for Question Answering. This model is composed by a standard Bert model [3] as backbone, followed by a linear layer on top. As reported in the paper they propose to use a modified binary cross entropy to compute the loss:

$$\mathcal{L} = -\sum_{i=1}^{n+2} \mathbf{y}_i^s \log(\mathbf{p}_i^s) - \sum_{j=1}^{n+2} \mathbf{y}_j^e \log(\mathbf{p}_j^e)$$

The loss doesn't take in account the no-target class, which seems wrong, bu we will address this problem in the modified binary cross entropy subsection of the issues. Thus, the pipeline for the training of the first part can be summerized in the following steps:

1) Tokenized sentence (or batch of sentences) is given to Bert which produces a vector of embeddings with dimension $batch \times length \times hidden\_size$, with $hidden\_size$ (or h) $= 768$;
2) The linear layer is then feed with the embedding vector, $\mathbf{g}^s = \mathbf{w}_s \mathbf{h}^L$, producing the output $logits$ of dimension $batch \times length$;
3) Finally, after having applied the softmax sentence-wise, $\mathbf{p}^s = softmax(\mathbf{g}^s)$, the modified cross entropy is computed and the backpropagation is performed.

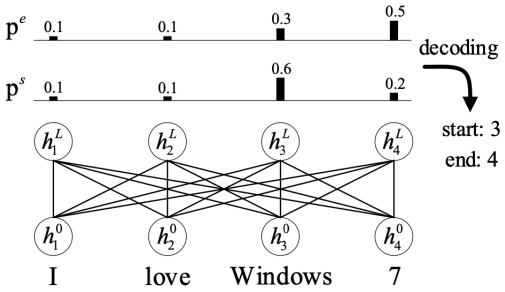

Fig. 1: Overview of the pipeline described

Regarding the inference/testing pipeline, an heuristic approach is applied. In standard QuestionAnswering approaches, where we are in a single-span extraction fashion, the sum of the logits value for all the $tuple(start\_pos, end\_pos)$ with $start\_pos < end\_pos$ is computed, and the best couple is the output. Here we cannot do the same keeping the best M spans, because inside these spans could be overlappings and too long targets. In order to deal with these problems, the heuristic in Figure 2 is implemented.

C. Polarity Classification

The second sub-task is instead solved using a modified Bert for Token Classification with local attention-like module on top. The pipeline is composed by a standard forward pass over with Bert model, followed by an iterative process over all the target spans. The loss function used to optimize is a categorical cross entropy with 3 classes (POS, NEG, NEU):

$$\mathcal{J} = -\sum_{i=1}^{k} \mathbf{y}_i^p \log(\mathbf{p}_i^p)$$

For each span, a weighted sum over all its tokens is computed to obtaining a summarized vector, on which is then predicted

Algorithm 1 Heuristic multi-span decoding

Input: $\mathbf{g}^s, \mathbf{g}^e, \gamma, K$
$\mathbf{g}^s$ denotes the score of start positions
$\mathbf{g}^e$ denotes the score of end positions
γ is a minimum score threshold
K is the maximum number of proposed targets

Initialize $\mathbf{R}, \mathbf{U}, \mathbf{O} = \{\}, \{\}, \{\}$
Get top-M indices $\mathbf{S}, \mathbf{E}$ from $\mathbf{g}^s, \mathbf{g}^e$
for $\mathbf{s}_i$ in $\mathbf{S}$ **do**
  for $\mathbf{e}_j$ in $\mathbf{E}$ **do**
    if $\mathbf{s}_i \leq \mathbf{e}_j$ and $\mathbf{g}^s_{\mathbf{s}_i} + \mathbf{g}^e_{\mathbf{e}_j} \geq \gamma$ **then**
      $\mathbf{u}_l = \mathbf{g}^s_{\mathbf{s}_i} + \mathbf{g}^e_{\mathbf{e}_j} - (\mathbf{e}_j - \mathbf{s}_i + 1)$
      $\mathbf{r}_l = (\mathbf{s}_i, \mathbf{e}_j)$
      $\mathbf{R} = \mathbf{R} \cup \{\mathbf{r}_l\}, \mathbf{U} = \mathbf{U} \cup \{\mathbf{u}_l\}$
while $\mathbf{R} \neq \{\}$ and size$(\mathbf{O}) < K$ **do**
  $l = \arg\max \mathbf{U}$
  $\mathbf{O} = \mathbf{O} \cup \{\mathbf{r}_l\}; \mathbf{R} = \mathbf{R} - \{\mathbf{r}_l\}; \mathbf{U} = \mathbf{U} - \{\mathbf{u}_l\}$
  for $\mathbf{r}_k$ in $\mathbf{R}$ **do**
    if f1$(\mathbf{r}_l, \mathbf{r}_k) \neq 0$ **then**
      $\mathbf{R} = \mathbf{R} - \{\mathbf{r}_k\}; \mathbf{U} = \mathbf{U} - \{\mathbf{u}_k\}$
return $\mathbf{O}$

Fig. 2: Heuristic for multi-target extraction

the polarity. The weight is the local attention-like parameter and it's computed applying a $softmax$ over the weighted output of Bert. The sequence of operations carried out is as follows:

1) $\alpha = \text{softmax}(\mathbf{w}_\alpha \mathbf{h}^L_{\mathbf{s}_i:\mathbf{e}_j})$
2) $\mathbf{v} = \sum_{t=\mathbf{s}_i}^{\mathbf{e}_j} \alpha_{t-\mathbf{s}_i+1} \mathbf{h}^L_t$
3) $\mathbf{g}^p = \mathbf{W}_p \text{tanh}(\mathbf{W}_v \mathbf{v})$, $\mathbf{p}^p = \text{softmax}(\mathbf{g}^p)$

During inference, the polarity probability is computed for each candidate target span in the set O (target spans extracted), and the sentiment class that possesses the maximum value in $\mathbf{p}^p$ is chosen.

III. Issues with the proposed approach

In this section, all the issues related to the work by Hu et Al. are presented, followed by a brief comment about how they could impact the final performances and how I tried to address them in order to check if similar results could be obtained. While some of the problems are related with the reproducibility of the results, highlighting inconsistencies between the code and what is explained in their paper, other problems are instead related on choices that seems to be wrong for solving the task in a proper way. Numerical results related to the tests performed, will be presented in the next section.

A. Sentences with no targets

The first issue I encountered while trying to reproduce their work is the remotion of the sentences with no targets from the computation. Indeed, in their paper, they didn't mention how they treat them, but from their code, at SpanABSA Github Repo I actually found out that these kind of sentences are excluded, both in training and testing. I discover this problem by looking at their parser function, but was spotted also by another user

and added to the issues in their github (issue). Regarding the performances, the exclusion of the sentences with no targets causes an obvious increment in the results in the first sub-task.

B. Pretrained model with no source

Another relevant issue is related to the actual pre-trained model they used. The model, which should be a pre-trained $Bert-base-uncased$ model can be download from a onedrive link they added in their github, but no source about how it was pre-trained are available. Moreover, during the testing, I found out that the performances are quite different compared to those obtained using the HuggingFace $Bert-base-uncased$ pre-trained model [4]. This difference could be due also to the fact that they use an older version of Bert w.r.t. the one available from the HuggingFace library.

C. Diffent version of Bert

As stated before, the version of Bert they use appear to be different, if compared with the one provided by HuggingFace library. I discover this when I try to test their pre-trained model and found out that no weights were imported using the BertModel class in Transformers library (the library provided by HuggingFace team). Thus, in order to guarantee reproducibility, I imported the code of that version of Bert and added it in my code. From the comparison of the two model the older one appear to be slightly better for this task, but this could be dued also to the different pre-trained model source.

D. Different optimizer

In the paper by Hu et Al., a standard Adam optimizer is reported to be used. Actually, by looking through the code the optimized used is a modified version of it, called BertAdam. The main differences (reported here BertAdam w.r.t. Vanilla Adam are:

- BertAdam implements weight decay fix,
- BertAdam doesn't compensate for bias as in the regular Adam optimizer;
- BertAdam incorporate a specific Scheduler to change dinamically the lr during the training steps;

E. Modified binary cross entropy

As reported in the paper (and in the formula above), a modified binary cross entropy is employed to compute the loss. However there are two problems. The minor one is that they didn't report that, after the summarization of the negative log likelihoods related to starts and ends, the final value is divided before by the number of start/end positions in the sentence and then by a factor of 2. This is done in order to compute the mean between all the target positions and the mean of between the start and end cross entropies. The more relevant one is related to the form of the modified binary cross entropy they implemented, which doesn't take in account of no-targets loss. I implemented both solutions and find out that the standard binary cross entropy doesn't work well. In my opinion, this is due to the fact that, since the targets (class 1) are far less than no-targets (class 0) and since cross-entropy suffers a lot by class unbalancing, the binary CE results to be too strong, thus leading to non convergence of the training. We have to recall that, since we are fine-tuning a foundation model we move inside a pre-defined and smaller search space (w.r.t. non pre-trained model) and so we have to apply some tricks in order to optimize the model. In this case, use the modified binary cross entropy translates to optimize only the targets, since when we multiplying the labels for the log likelihood, we are in fact multiplying by 0 all the loss values that belong to the no-target class.

F. Threshold value not reported

Finally, a less relevant point, but still a problem in reproducibility is that no threshold for the multi-target extraction heuristic is reported. The threshold is used to decide if a couple that represents a span should be keeped or dropped. In the paper is only stated that the threshold was hardcoded for each dataset they used, keeping the one that maximize the f1, for that dataset. Luckily, from the code can be seen that for a dataset with similar statistics of the laptop14 one, called total_rest, a threshold of 7.5 is used. Thus, I was able to find the best threshold for the laptop14 near that one, with a value of 9.

IV. Implementation and Results

In this sections all the relevant details of the implementation will be presented, followed by the tests performed in order to assess all the performances variations with different parameters and modules.

A. Testing module

In order to make all the testing phase easier I developed several workspace modules in which almost all hyperparameters presented in the code can be quickly changed. The data-structure used to retain all the parameters is a nested dictionary. The parameters that could be controlled are the following:

- **Optimizer**: Name (type of optimizer), Learning Rate, Scheduler, Warmup (in percentage of step);
- **Model**: Gamma (Threshold), K (Number of maximum targets extractable, M (number of top targets extracted before Heuristic), Model_class (Model used), Pretrained_weights, Layers (dict to manually select grad flag for each bert layer), All_params_grad (flag used to set all the param_grad True);
- **Data**: Max_length (maximum length of sentences, used for padding), train_split, pretrained tokenizer, tokenizer model'
- **Training**: batch_size, epochs, reproducibility (which change the BertModel to the older version)
- **Evaluation**: paths, gamma_range (if first task), number of restarts.

Moreover the $workspace$ functions are called from the $main$ with a string parameter that allow the user to choose between the modalities: te_train, te_eval, pc_train, pc_eval, te_pc_pipe.

B. Hyperparameters

For the training of the Target extraction model, the optimizer used is AdamW with Linear scheduler with warmup of 0.1 and learning rate equal to 3e-4. The batch size is 32, the epochs are 3 and the best results are obtained with a training split of 1 (no split) This is a consequence of the small dataset used. Adding a split to integrate validation set we can train for more epochs applying the early stopping in order to avoid overfitting, but this leads to worser performances than using no validation set. Regarding the tokenization, BertTokenizer is used and the max_length parameter is fixed to 96. During inference K is equal to 10, M to 20 and gamma leads to the best results is 8.5. For the second task, the same hyperparameters are used for the training of the first are used, except for the batch_size which is equal to 16.

C. Evaluation methods

In order to perform the evaluations on the task, the metrics that should be used can be found in [2]. For the first task, a modified version of precision and recall are computed, in order to obtain the f1. Specifically we have to define the element to compute the metrics:

- S: set of all targets retrieved by the model;
- G: set of all the correct targets;
- C: set of all the targets in common, thus correct (TP);

moreover a target is considered in common if and only if all the tokens inside the target span predicted match with all the tokens inside the corrected target span.

Thus the metrics are computed as follows:

$$P = \frac{|C|}{|S|}, R = \frac{|C|}{|G|}$$

and then the F1 is computed in the standard way:

$$F1 = \frac{2 \cdot P \cdot R}{P + R}$$

In order to evaulate the second task, the accuracy is taken in account. Since we are in a multi-class fashion, with POS, NEG and NEU classes, given the confusion matrix M, the accuracy is computed as:

$$ACC = \frac{\sum_{i=0}^{C} M_{i,i}}{\sum_{i=0}^{C} \sum_{j=0}^{C} M_{i,j}}$$

where C is the number of classes, thus $C = 3$.

Finally the policy to evaulate the jointed task is similar to the first task, excpet for the commons/corrects which should also match the predicted polarity. We could also say that performances for the jointed pipeline are expected to be equal to $F1 \cdot Acc$, since we have to multiply the number of commons for the accuracy at the second stage.

D. Baselines

In order to better assess the results obtained from the method, I relied on the baselines reported in the paper of the original dataset [2]. For the first sub-task an F1 of 35.64% is reported, while for the second sub-task an Acc of 51.37% is reported, with a majority of results equal to 52.14%. Other baselines can be found in the original paper [1], where the method is compared with two BIO-tagging approaches, the first one uses CRF as decoder, while the second one proposes a DE-CNN to perform the tagging.

E. Tests performed

In this subsections are reported the most interesting tests performed. Most of the tests that were performed for debugging or understanding purposes (such as trying different optimizer or using DistilBert instead of Bert) are not reported here due to very small changing in performances. Moreover, in the following results are reported only the hyperparams performances the most.

1) Targets extraction:

- F1: 53.1%, lr = 3e-4, epochs = 4, no-targets excluded;
- F1: 30.1%, lr = 3e-4, epochs = 4, no-targets included;
- F1: 53.4%, lr = 2e-5, epochs = 3, no-targets excluded, older Bert (the one from the original SpanABSA repo)

2) Polarity classification: :

- Acc: 68.7%, lr = 3e-5, epochs = 10;

3) Jointed task:

- F1 pipe: 44.9%, params used are the best for the previous tasks.

The final $F1$ is sligthy more than the one expected (36% with $F1 \cdot Acc$) because the corrected targets extracted from the first step represent only a subset of all the correct targets, and so the accuracy of the second stage on that subset is higher that 68.7%.

V. Conclusions

In conclusions, overall results are accettable. However it could be interesting to try also different approaches. Expecially, despite the fact that in the work by Hu et al. is said that BIO-tagging approach introduce a bigger search space, it seems to be still more reliable and effective. For the sake of completeness, a really interesting approach is the one introduce by Yang et al. [5], that reaches the state of the art for the task. The idea behind this approach is to use dependencies in order to cluster together similar sentiments and also token that could be in the same target. Moreover, a sort of mix between the extraction of the span and the tagging is performed, with astonishing results (86% accuracy on the jointed task on laptop dataset).

References

[1] M. Hu, Y. Peng, Z. Huang, D. Li, and Y. Lv, "Open-domain targeted sentiment analysis via span-based extraction and classification," 2019.

[2] M. Pontiki, D. Galanis, J. Pavlopoulos, H. Papageorgiou, I. Androutsopoulos, and S. Manandhar, "Semeval-2014 task 4: Aspect based sentiment analysis," in *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*. Association for Computational Linguistics and Dublin City University, 2014.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[4] T. W. et Al., "Huggingface's transformers: State-of-the-art natural language processing," 2020.

[5] H. Yang, B. Zeng, M. Xu, and T. Wang, "Back to reality: Leveraging pattern-driven modeling to enable affordable sentiment dependency learning," 2021.