Identification and polarity classification of aspect terms

Niccolò Morabito
1808746
morabito.1808746@studenti.uniroma1.it

1 Introduction

Aspect-based Sentiment Analysis (ABSA) is the use of natural language processing to, given a sentence, identify its aspect terms and classify their polarity (assign a sentiment to each of them). The project wants to address a multi-task learning problem, therefore it is important to distinguish between the two tasks.

2 Aspect term identification (A)

For the first task, we expect the model to output zero, one or more targets, each of them having one or more words (e.g. in the sentence "We use the built in tools often, iTunes is open nearly every day and works great with my iPhone.", the identifier should extract both *iTunes* and *built in tools*).

According to this requirement, we do not have a fixed number of targets nor a fixed target length. Each word and, more specifically, each token, is potentially a target itself or a part of a target. Using a more sophisticated word embedding, we might directly represent the n-token terms (n could be also 21, i.e. the maximum length of a target in the dataset) with a single vector and reduce the number of predictions. This could also increase the performance both for the greater context given by the aggregation and for the lower chance to partially or completely miss the aspect term. With the chosen embeddings, on the other hand, we need to produce a binary classification for each token the sentence has been split into. In this way, a sentence with no target will correspond to all 0s classifications and more targets can be actually predicted. The banal idea to compose *multi-tokens* targets is to simply aggregate the positive predictions that are close (i.e. no negative predictions in the middle) [see example], but better and more sophisticated methods could improve the capacity to correctly identify a complete target. As the table 1 shows, this approach can generally work, even if the performance obviously decreases as the targets' number of words increases.

3 Aspect term polarity classification (B)

The second model should predict a sentiment (between positive, negative, conflict and neutral) for each token or, more precisely, for each target identified by model A. What is important is that a single prediction for the input text is not enough. Therefore, five classes are necessary: a prediction is going to be output for each word and a class 0 corresponds to non-target. Reducing this number to 4 would have ended up into having a sentiment for each token, which is not the case, and it would have created problems in the implementation (e.g., in order to make the loss function ignore certain predictions, it is necessary to specify the corresponding index).

Of course, it is important to avoid a *double* prediction of target/non-target. By default, indeed, predicting five classes would mean that also this model should learn to distinguish between targets and non-targets by assigning no sentiment to the latter. However, this behavior can be easily ruled out through some parameters of Pytorch classes or functions. In particular, the loss function is forced to completely ignore class 0 (non-target token with no sentiment) for loss computation in order to prevent gradient to depend on that class (which is also the majority one). Consequentially, the loss function is taking into account only the predictions of actual sentiments.

Nevertheless, this is not enough since it is also important to tell model B which words are targets and which not, otherwise it could anyway predict 0 for a target token or (less importantly) a sentiment for non-target tokens. This is why the true target terms (during training) and those predicted

by model A (for prediction) should be *applied* to the logits produced by the model B before computing the loss error: for all the non-target tokens (according to the dataset or the model A prediction), a huge number is set in the first position in order to "force" the corresponding word to be ignored by the sentiment classifier, while an opposite number is put in the same position for target tokens in order to take the second-best prediction (which will necessarily correspond to a sentiment).

This procedure is used to classify multiple targets. In addition to this, another idea must be found in order to deal with multi-tokens targets. Since each token could be classified in a different way, we need to choose a combining rule: a simple majority vote has been used for the greatest part of the attempts and the hyper-parameters tuning phases.

The ideas explained so far have been adapted to the technical and practical problems deriving from the implementation, which turned out to be the difficult and limiting part of the work.

4 Implementation

As anticipated, the ground truth and the prediction both for model A and B are represented by a tensor of n positions, where n is the number of tokens the sentence has been split into (or, more precisely, that number after adding the padding to make all the sentences in a batch of the same length). This idea adds a lot of limitations to the pre-processing part: in order to decode (i.e. get target terms from the prediction tensor or associate sentiments to the correct tokens), we need to map the target indices that are provided in the dataset with the index (or indices) of the *tokenized* input text containing the corresponding terms [see example]. For instance, using the `TreebankWordTokenizer` of NLTK library (nlt), it is possible to tokenize a text and get the spans of each token. However, in order to compare these spans with the start/end indices of a target, they must be updated accordingly to what is done during pre-processing the input text.

The previous homework has not shown big differences in the overall performance of the models trying several combinations of pre-processing. Obviously, each problem is different and this specific case could possibly be positively affected by some kind of pre-processing. However, the chosen embedding is the same (pre-trained **GloVe** (Pennington et al., 2014), Wikipedia 2014 version with 50 features per word), and it was already revealed that the percentage of recognized tokens, including punctuation, is anyway very high since the dimension of vocabulary is 400k. Therefore, the only possible improvement that pre-processing can lead to is the reduction of noise and of less important features but, according to a few pre-processing tests with discouraging results, this is not the case. These are the reasons why I finally opted, for the first attempts, not to pre-process the input text before providing it to the neural network.

Model A and model B are both composed by a first word embedding layer which, as already said, encode a sentence using GloVe. After that, an LSTM processes the input text from left to right (and from right to left in case it is bidirectional) and outputs a new representation of each word that depends on the preceding words. Each state of the LSTM (corresponding to a token of the original input) is finally passed to zero or more linear layers (alternated by a Rectified Linear Unit activation function (Brownlee, 2019)) that produce the final classification (one for each token, in both cases). A dropout with different values (including 0) has been applied both after embedding and on the output states of the LSTM for regularization and for preventing overfitting. This mechanism randomly zeroes some of the elements with the specified probability using samples from a Bernoulli distribution during training. (Hinton et al., 2012)

The little but important differences between the two architectures depend on the expected predictions:

- for the first model, we expect a binary classification for each token; therefore, Binary Cross-Entropy is a suitable loss function and it is necessary to apply the Sigmoid function to the output logits in order to scale them in the range (0,1) and finally round the results;
- for the second model, on the other hand, a standard Cross-Entropy Loss has been used and, since the model should predict a single sentiment for each token, the predictions are obtained using `argmax` operator.

5 Hyper-parameters tuning and other approaches

In order to improve the two models described above, several hyper-parameters have been tuned training different models according to the various combinations that they produced. The results achieved by the best models are summarized in

the table 3. Looking at them, it is clear that, even if the aspect-term identifier could and should be improved a lot, the real bottleneck of the overall ABSA module is the aspect-term-polarity classifier, which is just above the 30% threshold with regard to the f1 score (macro). As a matter of fact, the task which turned out to be the most difficult and whose results were the worst is the polarity classification

That is why tuning focused mainly on this one in order to obtain the best performance from it and, in particular, to maximize the f1 macro score. Results from different choices are averaged in the table 2. Other minor attempts were manually carried out for the first one.

During the training of a model, one callback saved the best model according to the f1 macro score on the validation set, while another one early-stopped the training after a certain amount of times in order to prevent overfitting (taking into account the same metric).

The achieved metrics show that the model B did not benefit from HP tuning as expected. This is easily explainable looking at the metrics per class in the table 4.

The number of false positives for the `Conflict` class is almost 100%. This depends on the fact that this class is seriously under-represented: only about 2% of the targets in the dataset (both training and validation) are labeled with a conflictual sentiment, therefore it is impossible to obtain performance comparable with the other classes, especially the positive and the negative ones. That is why the only method which has been tested to address class imbalance was to compute loss assigning different weights to the classes: also these weights have been tuned, and the resulting metrics (best results of each combination) can be observed in table 5 and in the figures 1 and 2. Unfortunately, it is evident that any combination of weights does not improve predictions of the conflict class (instead, only the number of FP increases) but it tends to worsen the other classes' performance.

The last big attempt that has been tested is about word embedding and pre-processing. GloVe is a context-independent word embedding, meaning that there is at most one single vector for each word that combines all the different senses. To make the predictions depend on the possible meanings of the words, we need something more complex and accurate like the **Bidirectional Encoder Representations from Transformers (BERT)** (Devlin et al., 2019). This could significantly affect the performance of those models that try to extract and classify the aspect terms from text, because the resulting features are potentially able to represent the corresponding context (Devlin and Chang, 2018; Kovaleva et al., 2019).

For this reason, I re-implemented both the models substituting GloVe with BERT. This affects also the encoding and decoding of targets and their sentiment because the BERT tokenizer is going to split the sentence in a different way with respect to the `TreebankWordTokenizer` previously used: it applies some kind of lemmatization and pre-processing by dividing a word into different components (ber).

The resulting models are significantly more complex, and the training phase can only be more computationally expensive. In order to reduce the overall time of training, the model is left set in the evaluation mode, therefore its weights are not updated according to the sentences in the training set. Nevertheless, the time needed for training a single model is in the order of hours and a comprehensive and rigorous hyper-parameters tuning was impossible to be carried out as it was done for the first two models.

However, it probably would not have helped since all the attempts that were manually made did appear as almost equivalent (in the case of aspect term identification) or even worse than before (for aspect term polarity classification).

6 Results and conclusions

All the numerical results discussed in this report must be considered temporary because they are obtained on the validation set and the model should be tested on new and unseen data in order to consider them reliable. However, as we already pointed out, the aspect-term-polarity classifier does not reach sufficient results. All the potential solutions that have been tested have shown bad or no results.

Future works should focus on improving its performance, in particular addressing class imbalance by collecting new data or using other techniques for data augmentation. Moreover, new attempts with pre-processing (removing stop-words, adding POS, etc.) or contextual embeddings could be carried out to improve also the identifier's results. However, the automatic process performed by BERT seemed to be useless (of course, a rigorous and wider tun-

ing may contradict this assumption), that is why the whole architecture probably needs to be redesigned: another possible approach could be to split the input text into more sentences (for instance, basing on the punctuation), each of one containing only one target, and make a single prediction for each of them. Other ideas would try to address the problem in a smarter way.

References

Bert documentation.

Nltk tokenizer package.

Jason Brownlee. 2019. A gentle introduction to the rectified linear unit (relu).

Jacob Devlin and Ming-Wei Chang. 2018. Open sourcing bert: State-of-the-art pre-training for natural language processing.

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

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors.

Olga Kovaleva, Alexey Romanov, Anna Rogers, and Anna Rumshisky. 2019. Revealing the dark secrets of BERT. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4365–4374.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

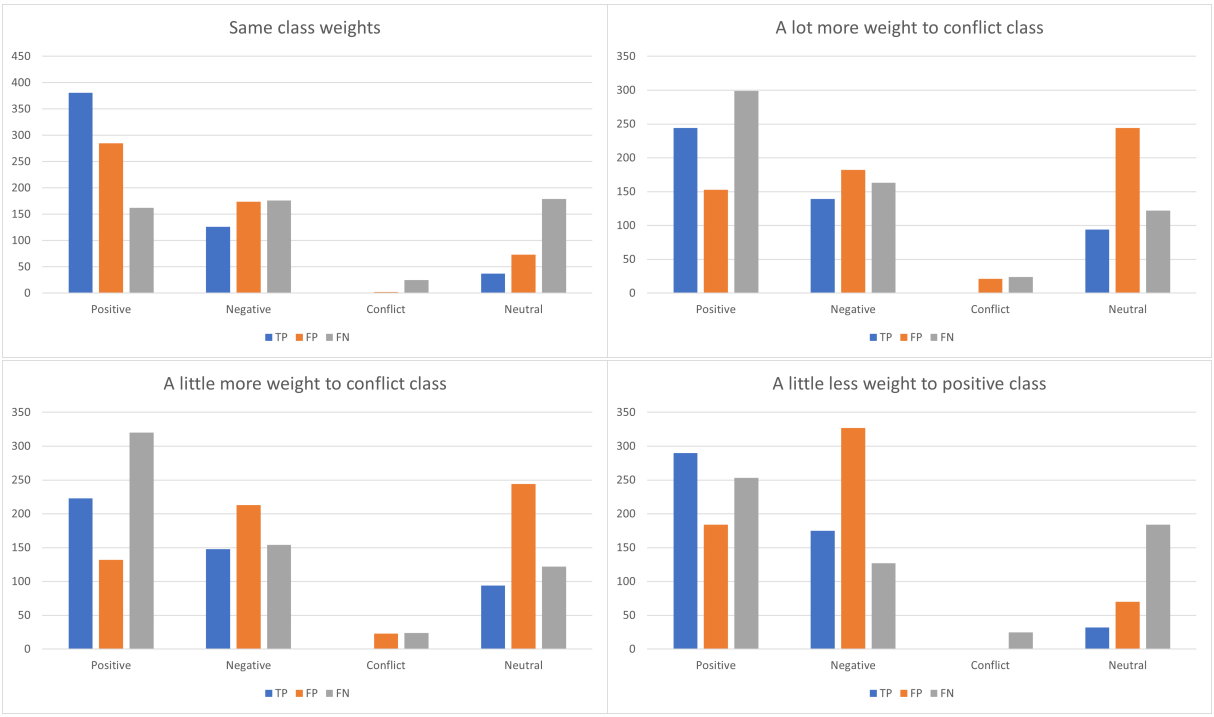

Figure 1: Comparison of metrics per class for each weight configuration in model B.

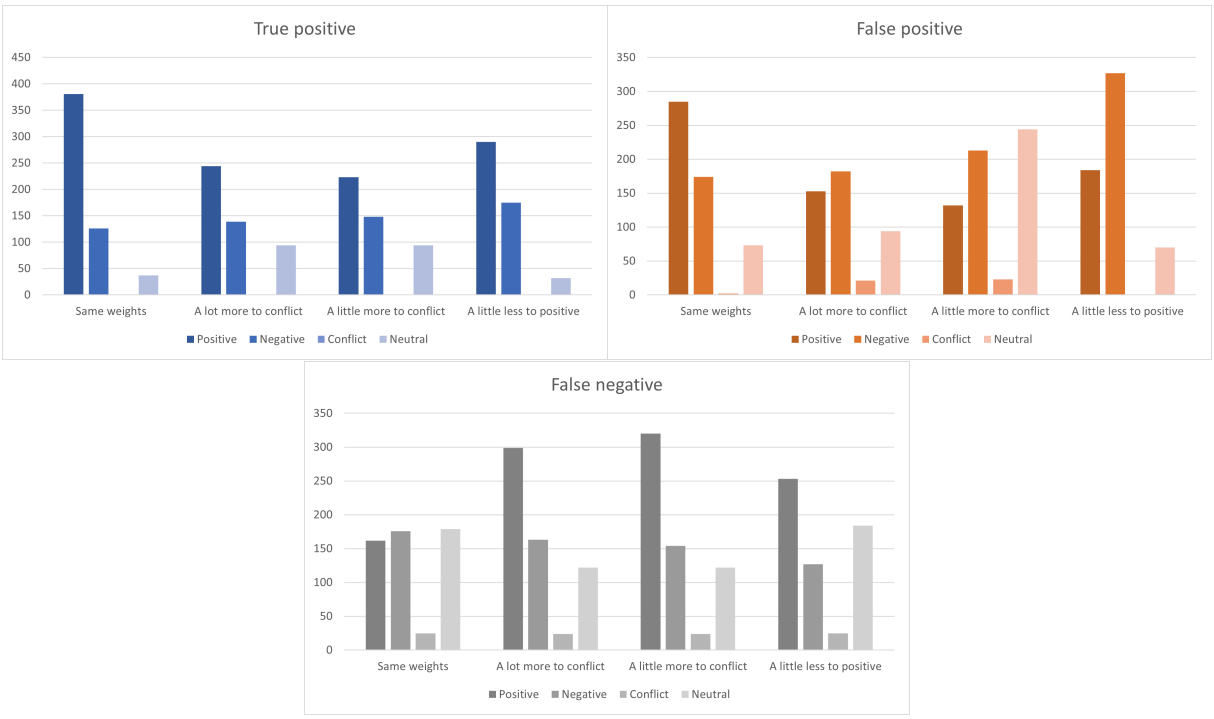

Figure 2: True positive, false positive and false negative values for each weight configuration in model B.

	Correct predictions	Total number	Recall
Mono-word targets	587	763	76.93%
Bi-word targets	106	230	46.09%
Tri-word targets	10	66	15.15%
n-word targets	0	35	-%

Table 1: Correct prediction rate of model A for targets' lengths (on the validation set).

	# of LSTM layers			Linear hidden layers		Dropout value		
	1	2	3	(50, 20, 10)	(300, 500, 100)	0	.1	.3
Average f1 score	32.23	31.52	26.21	26.14	31.65	33.12	31.17	26.82

	Bidirectional	
	True	False
Average f1 score	30.5	28.02

Table 2: Average macro f1 score for each possible value of each hyper-parameter in model B. The f1-scores are achieved during training on the validation set, but they are obtained averaging the results for each batch and they are different from the overall results on the total validation set.

	TP	FP	FN	Precision (macro)	Recall (macro)	F1 (macro)
Aspect-term identification	689	429	393	61.63	63.68	62.64
Aspect-term pol. classification	478	600	608	34.24	34.62	33.67
Overall ABSA module	340	778	746	21.77	22.33	21.82

Table 3: Final results for model A, model B and model A+B on the validation set.

Sentiment	Precision	Recall	F1
Positive	64.21	33.70	44.20
Negative	41.97	49.34	45.36
Neutral	26.26	48.15	33.99
Conflict	2.38	4.0	2.99

Table 4: Results of the best model B per class on the validation set.

	Configuration 1				Configuration 2				Configuration 3				Configuration 4			
	w	TP	FP	FN	w	TP	FP	FN	w	TP	FP	FN	w	TP	FP	FN
Positive	1	381	285	162	1	244	153	299	2	223	132	320	1	290	184	253
Negative	1	126	174	176	3	139	182	163	3	148	213	154	2	175	327	127
Conflict	1	0	2	25	10	1	21	24	5	1	23	24	2	0	0	25
Neutral	1	37	73	179	5	94	244	122	4	94	244	122	2	32	70	184

Table 5: Metrics per class of the best model B for each choice of weights (configuration). In the first column there are the weights of each class and then the metrics obtained by the best model for that configuration on the validation set.

Sentence	*Other installed features, such as certain printer software, are also most attractive.*
Targets	features, printer software
Tokenized sentence	[*other*, *installed*, *features*, *such*, *as*, *certain*, *printer*, *software*, *are*, *also*, *most*, *attractive*]
Targets vector	[0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0]

Example of how a sentence is encoded for the aspect-term identifier. In the case of the classifier, the encoding is the same except that a target token is labeled with one out the four possible sentiments.

Sentence	*Besides the great look, it is a great machine.*
Target text indices	[18, 22]
Target token index	3

Example of conversion from indices of the target in the text to index of the target in the tokenized sentence.