Explainable Sentence-Level Sentiment Analysis for Amazon Product Reviews

Andrea Rubeis
s290216

Lucia Innocenti
s279123

Shannon Mc Mahon
s289958

***Abstract*—In this project, we report the results of a ranking classification problem on a text dataset. The dataset contains text sentences, so an NLP Pipeline has been used for the scope: starting from preprocessing like stop-word removal and tokenization, moving to Word Embedding and finally using a BiLSTM with self-attention. Moreover, the concept of attention is also useful to perform another task: explainability. We kept track of the different attention values for each term in order to obtain a general value of importance for a given word in the classification task.**

I. Problem statement

Nowadays, more and more people are purchasing products online: in 2020 17.8% of worldwide retail sales came from e-commerce platforms.
One of the main differences with respect to traditional shopping is not being able to see the product in real life before buying it. Images and reviews from customers are in this sense vital to the success e-commerce has had.
This project focuses on the former, applying sentiment analysis to the reviews to the benefit of both consumers and producers. Starting from the Amazon Musical Instruments Reviews dataset, we worked on two aspects:

- Sentence classification, that takes as input the text of the review and tries to predict the score (a value from 1 to 5) the user will give to the product. This is done by using a BiLSTM with self attention.
- Explainability analysis, which is the goal of understanding which part of the sentence most influenced the result

Our implementation is available in the current repository.

II. Methodology

In this project we analyzed and reproduced the work done in [2].

A. Preprocessing

First of all we use Amazon Musical Instruments Reviews dataset, from which we only take the columns "reviewText" (containing the sentence reviews) and "Overall" (containing the ranking between 0-5) used as samples and targets respectively.

Before train the model we preprocess our sentences by firstly removing non-alphabetic terms; then, a stop-word removal phase is performed, using the list of more common stopwords from the English vocabulary. Given the such filtered reviews, the first step for an NLP process is produce a word embedding for each term in the sentence. In this project, we used a Bert Encoder [1], that is composed of:

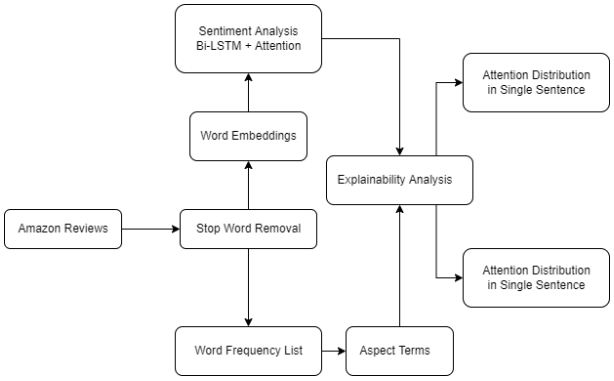

Fig. 1. NLP Pipeline

- *Tokenizer*: essentially it splits a phrase, sentence, paragraph, or an entire text document into smaller units, such as individual words or terms. Each of these smaller units are called tokens.
- *Embedder*: it returns, for each word, a features vector that represents the word in a n-dimensional space. Bert provides, for each word, 12 hidden-layer that represent the word during the Encoding step. In our project, we used different combination of part of these layers as word embedding vector.

B. Aspect terms

After the stop removal, to find the aspect terms we apply the TF-IDF and sorting in a decreasing order. The final aspect term is given by the top 160 most frequent nouns in the dataset. During the training process, for each word in the dataset is stored the attention value, and the final value is averaged over all the occurrences. This allow us to get, for each of the 160 more frequent terms, a value representing the importance that the word had during the whole classification process.

C. Model

The model proposed is the Self-Attention Bi-LSTM neural network [3] which is made by a BiLSTM layer, Attention layer and a final Softmax layer.

1) Bi-LSTM layer: The bidirectional LSTM is an extension of RNN able to solve the problem of vanishing gradient that typically occurs in standard RNNs. Looking at Fig. [2] the

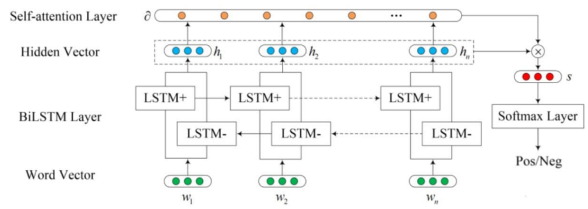

Fig. 2. Model Architecture

LSTM network receives as input the review represented by a word vector w where w_1, w_2, ..., w_n are the words already embedded with BERT and padded of the review of length n. While $h = \{h_1, h_2, ..., h_n\}$ represents the hidden vector.

2) LSTM Cell: The LSTM blocks in Fig. [2] represent the LSTM cells which are made by the following main components: *cell state* in which aims to memorize the most important words of the review, the *forgetting gate* which delete the important words collected found in the previous review at time $t-1$, the *input gate* that update the current cell state at time t and finally the *output gate* which returns the logits obtained by the word processed. In our case we have a customized LSTM cell different from Pytorch default one and it is defined according to the following gates:

- *Forget gate*

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + V_f c_{t-1} + b_f) \tag{1}$$

- *Input gate*

$$\begin{cases} i_t = \sigma(W_t x_t + U_i h_{t-1} + V_i c_{t-1} + b_i) \\ c_{in_t} = \tanh(W_c x_t + U_c h_{t-1} + V_c c_{t-1} + b_c) \\ c_t = f_t \cdot c_{t-1} + i_t \cdot c_{in_t} \end{cases} \tag{2}$$

- *Output gate*

$$\begin{cases} o_t = \sigma(W_o x_t + U_o h_{t-1} + V_o c_{t-1} + b_o) \\ h_t = o_t \cdot \tanh(c_t) \end{cases} \tag{3}$$

Where:

- c_{t-1}, h_{t-1}, x_t: they are the inputs of out BiLSTM network where x_t is the word embedded version obtained with BERT and h_{t-1} c_{t-1} are zero vectors at the beginning.
- c_{t-1}: cell state at the last moment.
- W_*: connecting weights between x and $*$
- U_*: connecting weights between h_{t-1} and $*$.
- V_*: connecting weights between c_{t-1} and $*$.
- b_*: bias term.

The Bi-LSTM consists in two indipendent LSTMs which accumulate the information from both forward and backward direction of the review obtaining $h_{t,f}$ and $h_{t,b}$. Finally merge together the information coming from both directions to obtain the final hidden vector $h_t = [h_{t,f}, h_{t,b}]$.

3) Attention Layer: The aim of the attention layer is to detect the words among the sentence that contribute more to the semantic of the sentence by assigning them higher weights. To do this, the final hidden layer h_t obtained by the Bi-LSTM layer is passed through the attention layer in which the following computations are computed:

$$u_t = \tanh(W_a h_t + b_a) \tag{4}$$

$$\delta_t = \frac{\exp({u_t}^T u_w)}{\sum_t \exp({u_t}^T u_w)} \tag{5}$$

$$s = \sum_t \delta_t h_t \tag{6}$$

First we generate a new hidden representation u_t by applying a Multilayer Perceptron to the final hidden layer h_t obtained by the Bi-LSTM layer. The context vector u_w represents a high dimensional representation to measure the relevance that each word has inside the sentence and it is randomly initialized and learnt together with u_t during the training process. The sentiment s is finally obtained after applying a weighted mean of the hidden vector through the softmax function.

4) Softmax layer: The results obtained by the attention layer are then transformed by applying dropout, a fully connected layer, the ReLU activation function and dropout again. We obtain the final logits vector o of length $c = 5$ equal to the number of classes. According to the formula [7] y_{pred} is obtained by applying a softmax layer as a classifier to the logits o and take the index -i.e the class- belonging to the max probability predicted.

$$y_{pred} = \arg\max(softmax(o)) \tag{7}$$

D. Loss Function

The final loss is obtained by summing up the Mean Square Error (MSE) loss and the Cross Entropy (CE) loss both scaled by a factor of 0.5.

$$L = \frac{1}{2} MSE + \frac{1}{2} CSE \tag{8}$$

With:

$$MSE = \sqrt{\frac{1}{N} \sum_i (y_i - y_{pred,i})^2}, \tag{9}$$

$$CSE = -\frac{1}{N} \sum_i y_i \log(y_{pred,i}) \tag{10}$$

E. Evaluation Metrics

The evaluation metrics used to measure the performances of our model are:

- Accuracy: ratio of comments correctly classified.

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{11}$$

- Precision: ratio of positive comments correclty labelled to all positevely labelled comments.

$$precision = \frac{TP}{TP + FP} \tag{12}$$

- Recall: ratio of negative comments correctly labelled to all negatively labelled comments.

$$recall = \frac{TN}{TN + FN} \tag{13}$$

- F1 score: weighted average of precision and recall.

$$F_1 = \frac{2 * precision * recall}{precision + recall} \tag{14}$$

Where TP are the true positive so number of positive elements correctly classified, FP are the false positive which are the negative reviews wrongly classified, TN are the true negative meaning the number of negative comments correctly classified and finally FN are the false negative, so the number of positive reviews wrongly classified.

III. Experiments

As far as experiments are concerned we used the following hyperparameters: learning rate $1e^{-3}$, dropout rate 0.4 and hidden dimension 256. As our optimizer we chose Adam, and our dataloader made use of a custom collate function which performed padding on the reviews since the dataloader requires all elements to have equal length (i.e. if for example we had 3 reviews with 2, 6 and 9 words each respectively, then we would pad the first two to length 9). Here we set number of workers to 4 and batch size to 128.

When creating our dictionary of embedded words with BERT we averaged out the last four tokens, for a total number of features of 768. To avoid having to recompute the embeddings on each run of the code we made use of Pickle, which serializing and de-serializing Python object structures. This saved us on average 50 min of time overall. This approach leads to an accuracy and recall of approximately 67% on the test set with a run time of about 20 minutes with a Tesla T4 GPU.

Furthermore, as previously discussed our model does not rely on default Bi-LSTM RNN offered by PyTorch, since the gates are differ slightly in their definition. Consequently we defined a custom net and LSTM Cell. We stress how in the backpropagation phase reversing the input granted us better results.

Finally, an important aspect to underline when considering our results is that the dataset is strongly unbalanced, specifically on a total of 10224 reviews approximately 67% have 5 stars, 20% 4 stars, 7.5% 3 stars, 3.5% 2 stars and 3% 1 star. It should no surpise us then that the accuracy metric for this problem does not exceed 67%. Since there is a strong imbalance towards the class 5 stars we can expect our net to reflect such imbalance. This can be seen in the confusion matrix available in figure 3, We underline that because of this unbalanced dataset we should not rely too much on the accuracy, but more so on the recall. In fact if we were given for example a dataset

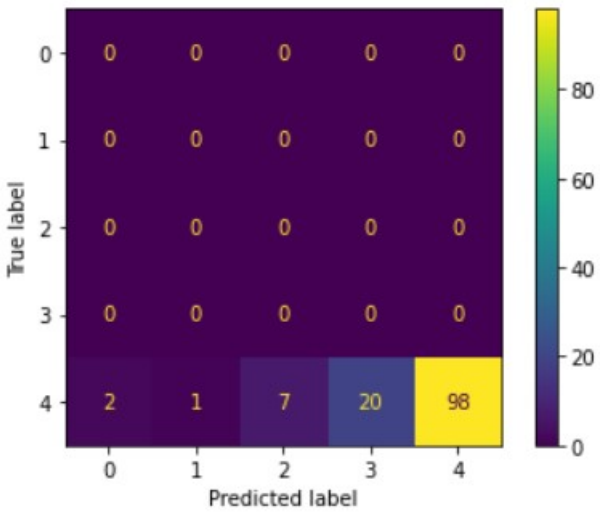

Fig. 3.

with only two classes A and B, where A accounts for 95% of the dataset while B for the remaining 5% we could reach an accuracy of 95% by simply predicting class A every time. A more proper calibrated method may instead achieve lower accuracy but have a substantially higher recall (true positive rate), which is really the metric we should be optimizing for. In any case, we will dive into a few ways to solve the inbalance in the dataset in the *Extensions* section.

IV. Extensions

In this section we focus on a few variations we made with respect to the original implementation.

Firstly, for the creation of our dictionary we introduce a second approach: if in the previously discussed case we average out the last four tokens, here we concatenate them (which results in a four times the number of features, 3072). In terms of results we did not notice a significant difference, since on average both approaches lead to an accuracy of 67%. However, as far as training is concerned using the average approach is less time consuming due to the reduced number of features.

Secondly, since we are dealing with a ranking classification problem which can be also seen as one of regression we computed the MSE loss in two different ways: the first compares true and predicted labels (e.g. 4 vs 3) while the second relies on probabilities and one hot encoding (e.g. if our expected label is 1 we will be comparing the one hot-encoded vector [1, 0, 0, 0, 0] with a probability vector obtained by applying a softmax to the linear output of our model, for example [0.6, 0.1, 0.005, 0.25, 0]).

Lastly, in the net we added a dropout, ReLU and fully connected layer which are not contemplated in the original version.

All the above changes did not lead to any significant improvements, and we believe this may be due to the strong dataset unbalance witnessed. Specifically for all the following combi-

nations we obtained an accuracy and recall of approximately 67% on the test dataset:

- concatenate pickle with loss = CE + MSE
- average pickle with loss = CE + MSE
- average pickle with loss = CE + MSE with one-hot enconding
- average pickle with loss = BCE + MSE

V. Conclusions

We have experimented that the use of BERT coupled with a custom Bi-LSTM RNN yields appreciable result for the task of sentiment analysis on the Amazon Musical Instrument Reviews dataset.

Next steps for our study might consist in addressing the imbalance of the dataset directly. One possible approach could be oversampling. Specifically, we could create new reviews for the classes which are less populated in order to rebalance the class frequencies. This could be achieved by starting from the existing reviews and creating "copies" of them, in which words are substituted by synonyms based on a probability (i.e. define a probability threshold and substitute a word with one of its synonyms only if the probability exceeds the threshold).

References

[1] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

[2] X. Li, X. Sun, Z. Xu, Y. Zhou, "Explainable Sentence-Level Sentiment Analysis for Amazon Product Reviews", November 2021

[3] Z. Hamed, B. Garcia-Zapirain, Sentiment Classification Using a Single-Layered BiLSTM Model