Natural Language Processing - Assignment #2

Lorenzo Pratesi, Martina Rossini, Riccardo Foschi, Vairo Di Pasquale
{lorenzo.pratesi2, martina.rossini3, riccardo.foschi4, vairo.dipasquale}@studio.unibo.it
Artificial Intelligence Master's Degree

Alma Mater Studiorum

Abstract

This work is centered on an emerging NLP task, formally known as Fact Checking. We first analyzed and prepared the given dataset, then trained and evaluated seven neural architectures chosen from the assignment specifications. Our best model obtained an accuracy and a "voting" accuracy both of 75% on the test set. Finally, we also trained and evaluated an architecture which includes an attention mechanism, obtaining a standard test accuracy of 78% and a "voting" test accuracy of 79%.

1 Introduction

Fact Checking is an emerging NLP technology which aims to asses the truthfulness of a claim. More precisely, in this work we try to predict whether a set of evidences confirm or refute a claim. The data we use during our experiments comes from the FEVER dataset. The approach we take when solving this problem is based on neural architectures: we start from the token embeddings, use them to compute a sentence representation for both claim and evidence and then combine those in a single vector, which is passed to a classificator. Note that, in addition to the required baseline models, we also developed a final architecture making use of an attention mechanism.

Section 2 shows how we pre-process the initial corpus and also how we create our embedding matrix. Indeed, since training the embedding layer from scratch implies learning a huge amount of parameters, we choose to rely on the pre-computed GloVe embeddings. Section 3 briefly describes the models we implemented and how they were trained/evaluated. Finally, in Section 4 we show our results and briefly discuss our errors.

2 Data Preparation

After splitting the datasets, we started to look at the data. We discovered that the classes are unevenly distributed only in the training set. We also saw that the sentences needed to be cleaned before the tokenization phase. Next, we created a custom vocabulary containing all the tokens associated with the GloVe embeddings of dimension 300 and all the other tokens we found in the train set. For the creation of the embedding matrix, we concatenated the index of each word that belongs to the GloVe's vocabulary to the corresponding pre-trained embedding. Instead, for the other words, we decided to assign to them a random embedding, where each element has the lower and upper bound as, respectively, the element-wise minimum and maximum of the embeddings of GloVe. For the OOV words found at inference time, we decided to assign them an unique random vector computed as described above. Next, we transformed each sentence in sequences of identifiers, with respect to the vocabulary. Finally, we zero-padded each sequence with a fixed maximum length (each claim/evidence of each dataset has its own), in order to be able to create batches during training.

3 Models' Description

As required by the assignment, we defined the four different architectures, the three merging strategies and the cosine similarity extension. For the embedding implementation, we used the Keras Embedding Layer. All the architectures process the embeddings of the claims and the evidences using the same encoder, but in a separate stream. The outputs of the encoder are two sentence embeddings, one for the claim and one for the evidence. After merging the two vectors, we pass the result to the classificator, a Keras Dense layer with an architecture-dependent input shape and a single output node, representing the prediction. The hyperparameters of the architectures are the following:

1. BiLSTM Last state: hidden units of the LSTM layer set to 256
2. BiLSTM Average states: hidden units of the LSTM layer set to 256
3. MLP: FC layer with an output shape of 256
4. Bag of Vectors: the output shape of the encoder is the same as the embedding shape (300)

We first decided to train all the architectures using concatenation as merging strategy. Then we selected the most performing method to construct a sentence embedding and tried it with the other two merging strategies. Again, after picking the most performing model, we tried it with the cosine similarity extension.

Finally, as extra work, we decided to implement our custom extension, based on an attention mechanism. In particular, we encode the claim/evidence embeddings using a BiLSTM, then we compute the Luong-style (1) attention using the claim sentences as queries and the evidence one as keys and values. Next, we merge the claims with the attention's output via concatenation and finally feed them to the classifier.

4 Result / Error Analysis

The results we obtained for validation accuracy and F1 score during our experiments are shown in Table 1, while the test scores are displayed in Table 2. Note that, we use a couple of naming conventions when building the aforementioned tables: first, next to each of the required models' name we use [C], [S] or [M], to indicate respectively "concatenate", "sum" or "mean" as the used merging strategy. Moreover, we called "voting scores" the scores we obtained by grouping together all the pairs `(claim, evidence)` with the same claim and then taking a majority vote. Note that the best model, according to both validation and test results, is the one with the attention mechanism; however, if we consider only the baseline neural models required by the assignment, the best results are obtained by the `LSTM last [C]` model, with `LSTM last cosine [C]` that comes as a close second.
Figure 1 shows the confusion matrix we obtained by evaluating our best model (i.e.: the one with the attention mechanism) on the test set. As we can clearly see, the model misclassifies mostly the class "REFUTES", which is also the minority class in the training set. We tried to mitigate this problem by assigning a weight to each class during train, but it did not have the expected results. Finally, during our experiments we found that all our models suffer from overfitting, a problem which we tried to solve by employing regularization techniques. However, we decided to dropped them and simply use early stopping in order to halt the train as soon as the overfitting starts. Indeed, neither weight decay nor dropout had the desired effect in our case, something which may be also caused by the difference in class distribution between train and validation/test datasets.

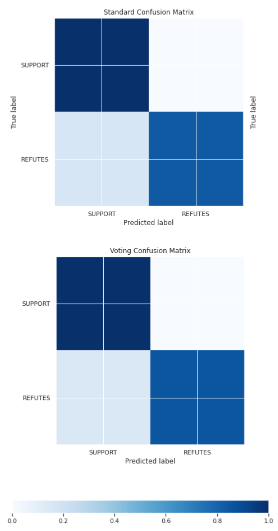

Figure 1: Best Model's Confusion Matrix

	Acc	Voting Acc	F1 voting
LSTM last [C]	77%	77%	77%
LSTM avg [C]	76%	76%	76%
MLP [C]	72%	72%	72%
Bag-Of-Vectors [C]	63%	63%	63%
LSTM last [S]	76%	75%	75%
LSTM last [M]	76%	76%	75%
LSTM last cosine [C]	77%	77%	77%
Attention	**80%**	**80%**	**80%**

Table 1: Models evaluation on the validation set

	Acc	Voting Acc	F1 voting
LSTM last [C]	75%	75%	75%
Attention	**78%**	**79%**	**79%**

Table 2: Best models evaluation on the test set

5 Future work

Our work could be further improved by employing more advanced neural architectures, like transformers. Moreover, during the EDA phase we noticed that the class distribution in the training and evaluation datasets is not the same, which may limit our model's ability to perform well on test data. Indeed, this problem is also known as "data mismatch" problem: we could try to mitigate this by either re-sampling the minority class or employing some data augmentation techniques like back translation. Finally, we could also try more sophisticated ways to handle the OOV when generating our embedding matrix (e.g.: using subwords models like fastText).

References

[1] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation, 2015.