Assignment 2: Fact Checking

Sandeep Kumar Kushwaha, sandeep.kushwaha@studio.unibo.it
Zarmina Ursino, zarmina.ursino@studio.unibo.it
Jyoti Yadav, Jyoti.yadav@studio.unibo.it

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Abstract

In recent years, disinformation and "fake news" have been spreading throughout the internet. In our assignment we focus on the task of Fact Extraction and Verification (FEVER) and its accompanied dataset. The task consists of validating whether the information in the documents supports or refutes a given claim.

Data pre-processing, Exploration & Encodings

The FEVER dataset includes 136094 claims & evidences with label (SUPPORTED, REFUTED) for training, validation and test are presented in *Fig.1*. We reduced the dataset by 90% to implement the experimentation while modelling, later we used the whole dataset for all the models & chose the best one.

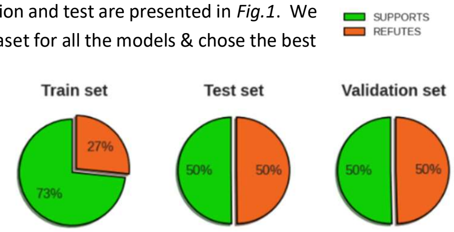

Figure 1, Distribution of data

We preprocessed the claims and the evidences using the following order of functions:

1. *preprocess_claim()*: removes last punctation symbols, we striped the whitespaces and converted the text to lower case
2. *preprocess_evidence()*: removes everything before the first tab character, parenthesis, everything between square brackets, everything after the last period, punctuation, and extra whitespaces. Moreover, we replaced tabs with spaces, and we converted the text to lower case.

CATEGORY	BEFORE	AFTER	%
EVIDENCES	35964	30856	14.20%
CLAIMS	31185	23229	25.51%

Table 1

We also made an analysis of our data before and after the preprocessing procedure: we calculated the number of tokens for the vocabulary of evidences and claims before and after and we calculated the reduction in percentage *[Figure 2 and 3]*. We plotted the distributions of evidence and claims sentences sizes using the seaborn *'distplot()'* function *[Figure 4 & 5].* As an extension, we tried using Lancaster, porter & snowball stemming (*stemming_process()* function) on the texts, but the results weren't so positive, so we commented it out. We applied binary encoding to the target label as we have mere two different targets. The features consisting of claims and evidence have been encoded using the GloVe embedding model.

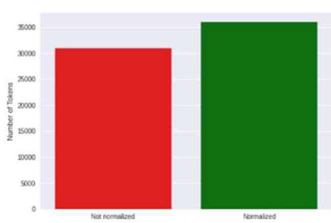

Figure 2

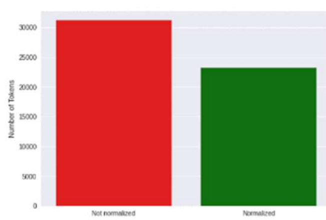

Figure 3

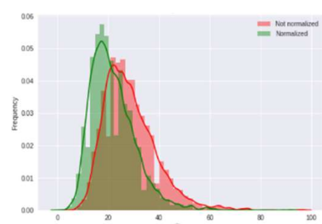

Figure 4

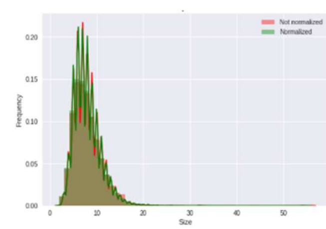

Figure 5

As the last step of exploration, we used visualization of the most frequent words by the WorldCloud python library only on the Training dataset. In *Figure 6* and *Figure 7* we showed respectively the 'WorldCloud' for Supported and Refuted Claims, where we can see that the most frequent words are the ones with bigger font size.

Figure 6

Figure 7

We downloaded pre-trained Glove embedding with the dimension of 100 and built the vocabularies and embedding matrices as we did in the first assignment regarding the POS tagging & we calculated the percentages of '*Out Of Vocabularies'* terms, whose percentage values are shown in *Figure 8*.

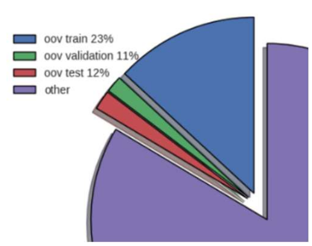

Figure 8

The dataset is handled using the class *DataLoaders* which implements the batch creation, padding and shuffling. A Base Model Class has been defined. It takes in input the '*input_layer_claim*' and '*input_layer_evidence*', two important entities, which are the *sentence_embedder* and the *embedding_merger* and a classifier layer, the Dense one. In this class the functions *call*(), *loss*(), *metrics*(), *weights*() and *summary*() have been implemented.

- *call()*: it returns as output the merged embedding between claims and evidences after having applied the Dense classifier layer
- *loss()*: we use for these models is sparse_categorical_crossentropy, which produces a category index of the most likely matching category.
- *metrics()*: It calculates the accuracy, f1-macro precision and recall in order to evaluate the model performance.
- *weights()*: this function copies the weights of the best model.

We shuffled the data for each epoch to have different data for each batch for both training and testing. A sentence embedding is a contextual representation of a sentence which is often created by transformation of word embeddings through a composition function. We created three different sentence embeddings:

- BOVEmbedding: we used the structural embeddings of words to create the '*Bag of Vectors*' representation for each sentence and we evaluate these sentence embeddings in each task.
- MLPEmbedding: we created '*Multi-layer perceptron*' class to obtain vector representation for a sentence by flattening out the model to have the same dimension on hidden features and by adding a Dense layer;
- RNNEmbedding: we implemented '*Recurrent Neural Network*' class, where we added a Bidirectional LSTM layer as it allows to learn the problem faster. This embedding has been used twice: the first embedding by taking the last step (whose instance is *RNNlast*) and setting a flag 'average' to False and the second by averaging all output states and setting the flag to True (whose instance is *RNNavg*).

To create the Embedding Mergers, first to merge multiple inputs we implemented the Parent class 'Merge ()', which accepts the boolean flag parameter *cosine*. To define the claim/Evidence merging strategies we created three children's' classes:

- Concatenation: It is the concatenation of the sentence embedding. Here we exploited the NumPy *concatenate()* method.
- Sum: It is the sum of the sentence embedding. Here we used the Add() layer.
- Mean: It is the average of the sentence embeddings. Here we used the Average() layer.

The hyperparameters used are '*batch_size*', *max_epochs*, '*learning_rate*', '*latent_dim'* (the number of nodes used as input of the generator) and the model applied. After analysing we considered followings as fixed: *latent_dim* = 64, *max_epochs* = 100 and *sequence* = 216. We tuned the *batch_size* hyperparameter value from 64 to 216, but we observed that the best values obtained of accuracy and f1-macro was when we consider *batch_size*= 216. Then we created two lists *sentences_emb_list and embedding_mergers_list,* whose elements are respectively the instances of the sentence embeddings and the mergers classes we described above. We implemented a nested for loop over the lists to fit the models for the training and validation set with the different mergers. In the *Table 2* we showed the last validation epochs' results of these combined architectures, in terms of loss, accuracy, f1-macro, precision and recall metrics.

	Loss	Accuracy	F1-macro	Precision	Recall
RNNEmbedding-last with Concatenation	1.151309609413147	0.6927374005317688	0.678957998752594	0.7238426208496094	0.6892158389091492
RNNEmbedding-last with Sum	0.9662585854530334	0.6438547372817993	0.6150432229042053	0.6943643689155579	0.641322910785675
RNNEmbedding-last with Mean	1.3434748649597168	0.6634078025817871	0.6387308239936829	0.721346378326416	0.6601753830909729
RNNEmbedding-Avg with Concatenation	1.0327926874160767	0.6578212380409241	0.6266567707061768	0.7034015655517578	0.6508037447929382
RNNEmbedding-Avg with Sum	1.0072684288024902	0.5782122611999512	0.49903127551078796	0.7101278305053711	0.573319137096405
RNNEmbedding-Avg with Mean	0.8971171975135803	0.505586564540863	0.33898258209228516	0.3350427448749542	0.4991636574268341
BOV with Concatenation	0.9618930816650391	0.50698322057724	0.33846405148506165	0.2563733756542206	0.5345678909876545
BOV with Sum	1.0887292623519897	0.50698322057724	0.33685705065727234	0.25414609909057617	0.5032345677654345
BOV with Mean	0.9065410494804382	0.50698322057724	0.3345162570476532	0.25191885232925415	0.5432345677654345
MLP with Concatenation	1.2017585039138794	0.6368715167045593	0.5934720635414124	0.7138814926147461	0.6336462497711182
MLP with Sum	0.9953599572181702	0.6061452627182007	0.550208747386932 4	0.7088988423347473	0.6031424403190613
MLP with Mean	1.2057470083236694	0.5377094745635986	0.40559327602386475	0.666241466999054	0.5287136435508728

Table 2, Results for Validation Set for the Baseline Models

For the extension task we decided to use the parameter *cosine*=True on the best performing model. We trained the model with the Mean, Sum, concatenation, and additional cosine similarity feature. Later it was observed that it didn't bring the major improvements after carefully observing the validation loss, accuracy and f1-macro values. So, we can conclude that the baseline model without cosine similarity performs better. On carefully analyzing all the results for the models we concluded that the two models which fairly performed well are **RNNEmbedding-last with Concatenation** & **RNNEmbedding-last with Mean** (both highlighted in green, *Table2*). They have the accuracy of 69% and 66% and F1 macro was 67% and 63% on the validation data. While evaluating the results on the test set it was observed that accuracy and F1-macro scores were around 65% and 63% respectively.

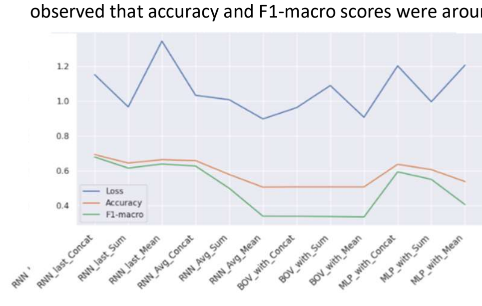

Figure 9 . Analysing the best model based on the loss, accuracy and epoch

We observed that the models BOV and MLP performed poorly on the training and the validation set they reached the accuracy and f1 -macro of 50% - 60% and 33% - 55%. The model **RNNEmbedding-last** and **the RNNEmbedding-last with Mean** were able to provide us with better solutions. In a nutshell the **RNNEmbedding-last** outperformed every model.

For getting the better results and improvements we could have:
1. Added more preprocessing steps like handling the number of foreign words;
2. Tried with different neural architectures like addition of attention.

References

Using Fact Checking: A survey, https://www.mdpi.com/2411-9660/5/3/42/pdf
The Fourth Workshop on Fact Extraction and Verification, https://fever.ai/
FEVER: a large-scale dataset for Fact Extraction and VERification, https://www.researchgate.net/publication/323771039_FEVER_a_large-scale_dataset_for_Fact_Extraction_and_VERification
Fact Extraction and VERification (FEVER) Challenge, https://competitions.codalab.org/competitions/18814
Custom metrics in Keras and how simple they are to use in tensorflow2.2, https://towardsdatascience.com/custom-metrics-in-keras-and-how-simple-they-are-to-use-in-tensorflow2-2-6d079c2ca279