One Million Posts Corpus

Jens Becker and **Julius Plehn** and **Oliver Pola**
Language Technology Group
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

Abstract

The purpose of this term paper is to summarize the development effort put into the creation of a multi-label neural network for text classification. The paper covers the employed preprocessing steps of the initial corpus, integration of an embedding layer and the actual deep learning network. Finally, an evaluation is given. We conclude that using a multi-model supports the training process and thus, improves the model predictions.

1 Introduction

Extensive work has been done on NLP tasks concerning the English language. To contribute to the knowledge about German language, similar work has been repeated on large collections of German text. Schabus et al. (2017) produced and analyzed the *One Million Posts Corpus* described in the next chapter in more detail. Later Schabus and Skowron (2018) improved their results using other methods.

Our work will use that corpus to try and reproduce some of the original work. The focus lies on applying the recurrent deep learning method LSTM (Hochreiter and Schmidhuber, 1997), which was also evaluated by the original authors.

The corpus contains labelled data for multiple categories. As the previous work did apply classifiers for each category on its own, we try to contribute another approach, using all categories at once.

Our code is publicly available on GitHub[1].

[1]Our code available at https://github.com/oliver-pola/OneMillionPostsCorpus/

	Labeled	Does apply		We apply	
Sentiment Negative	3599	1691	47%		
Sentiment Neutral	3599	1865	52%		
Sentiment Positive	3599	43	1%		
Off Topic	3599	580	16%		
Inappropriate	3599	303	8%		
Discriminating	3599	282	8%		
Possibly Feedback	6038	1301	22%	72	2%
Personal Stories	9336	1625	17%	47	1%
Arguments Used	3599	1022	28%		

Table 1: Categories of posts and their distribution according to Schabus et al. (2017)

2 One Million Posts Corpus

The corpus' data[2] originates from the website of the Austrian newspaper *DER STANDARD*[3] and represents User generated Posts. The dataset contains posts that are labelled according to nine distinct categories. The labeling process on these posts was performed by forum moderators. The posts have been written during a time period from 2015-2016 and contain 1 million unlabelled posts as well as 11,773 hand-labelled posts.

2.1 Categories

In Table 1 an overview of included categories is shown. The meaning of those categories is explained by Schabus et al. (2017). In the first column is shown that for example 3599 posts were labbelled in the *Off Topic* category. The second column shows that in this case 580 posts where indeed *Off Topic*, resulting in 16% of the labelled posts of this specific category.

As previously mentioned there are 11,773 labelled posts. This means after labelling 3599 posts in each category, there were another 2439 posts labeled in the *Possibly Feedback* and yet another

[2]Corpus available at https://ofai.github.io/million-post-corpus/

[3]DER STANDARD https://www.derstandard.at

5737 posts in the *Personal Stories* category only. These additional posts were strongly biased towards positive labels, so that the category overall applies to 22% / 17% of posts, whereas in a random sample like the 3599 posts it is rather 2% / 1%. Besides this questionable labelling method, the additional posts can't be used in our method for technical reasons. As we want to build a model that is capable of identifying multiple categories at once we use only posts that are annotated as 0 or 1 for each category. So only those 3599 posts can be used in our approach.

Looking at the percentage where the categories do apply, it becomes already clear that most of them will be problematic. Having only a few positive labels will encourage the model to always predict negative, as this will result in a 99% accuracy when the positive labels are 1% only, like in the *Sentiment Positive* category. With that small of a dataset for training it is also not applicable to omit some of the negative posts to train with an artificial balance around 50% each.

Also there is the *Off Topic* category with probably enough positives, but technically we will consider single posts only, out of their context among other posts belonging to certain articles. But the latter would be necessary to detect something as off-topic.

To summarize, we expect to produce useful results in the *Sentiment Negative*, *Sentiment Neutral* and *Arguments Used* categories only.

3 Embedding

Embedding is the task to transform the textual input, our posts, into some numerical vectors that can be used for calculation within a neural network. We use Word2Vec (Mikolov et al., 2013), where the high dimensionality of the resulting vector tries to encode the semantic of words. Words that often appear in similar context are considered to have similar meaning and appear in close distance in the resulting vector space. Word2Vec is a learning task on its own that we do not perform ourselves. Instead, we use a pretrained German model[4] that uses a vocabulary of 1,309,281 words, each embedded into a vector of size 300. The model is loaded and applied by Gensim[5] (Řehůřek and Sojka, 2010).

[4]German Word2Vec model available at https://deepset.ai/german-word-embeddings

[5]Gensim https://radimrehurek.com/gensim/

Every post is therefore split into words and each word is then embedded into a vector. Having multiple words in a post results in a list of vectors or rather a matrix. Words not found in the vocabulary will be skipped in the process. Since a neural network expects a fixed input size (at least within one batch), the matrix is padded to the size 80×300, cutting off too long posts and filling short ones with zeroes at the end. We chose the number 80 based on the maximum length of posts in the training data to be 161 and the average 37 words, also having memory and processing time in mind.

The standard approach to applying such an embedding seems to be generating an embedding matrix, that contains the vectors for all the words in the vocabulary and then have the input be the indices of the words. This matrix is then fed into the first layer of the network. With this approach we had some problems due to memory limitations on some systems. Also feeding the embedding matrix into the network and using Tensorflow on a GPU means to load that embedding matrix into the limited GPU memory. Many rows in that matrix will not be used in the current batch, some of them will in fact never be used.

Instead we considered the embedding as a preprocessing step, using Gensim to go from words to vectors directly. The input of the network will then be vectors already and no embedding matrix needs to be stored in GPU memory. Since we also only do the preprocessing once and then repeatedly feed the vectors into the network, the embedding is also done only once instead of embedding the same post again in each epoch. Also this opens up the ability to free the memory of the embedding model after preprocessing is done. That reduced the memory needs of our model and we could use less advanced hardware for training, only dealing with processing time.

4 Deep Learning

This chapter shows the implementation of our network model, justifies the chosen hyper parameters and describes the training.

4.1 Model

The model has been implemented using Tensorflow 2 and Keras. Because it tends to overfit several dropouts are applied. As can be seen in Figure 1 the first layer uses a dropout rate of 20%. The following LSTM layer uses 128 units while

also applying a dropout rate of 40%. The LSTM is wrapped by a bidirectional layer which improved the performance by using information of the past, as well of the future. Finally, two dense layers are used, where the last layer maps the output of the network to the nine categories shown earlier.

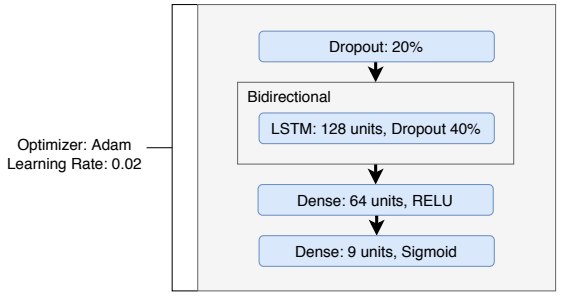

Figure 1: Applied deep learning network

This describes our final model, we call *Multi-Model* where we use all the categories combined to a vector of size 9. To reproduce the original work, we first implemented a classifier that was considering a single category only. This model we call *Single-Model* and it is similar to Figure 1 except having only one unit in the final classification layer.

4.2 Hyper Parameters

The hyper parameters of our model are justified by a search over a range of values, shown in Figure 2. We completed a full training with each set of parameters. Finally the best model measured by F_1 score on the validation set is applied. A first impression was that out model performs well in the *Arguments Used* category, so we concentrated to optimize for that category.

The training results are evaluated on two different systems and averaged over a total of six runs. Figure 2 (top) excludes data we've collected for 32 and 192 LSTM units for a better overwiev, whereas Figure 2 (bottom) is limited to a selection of dense units around the maximum. The bottom curves show that there is a noticable jump in the networks ability when increasing from 96 to 128 LSTM units. We have chosen 128 LSTM units (having this bidirectional results in 256 trainable parameters) and 64 dense units, because we consider that peak as more pronounced compared to the 160 LSTM curve, it had a smaller standard deviation and we preferred to have fewer parameters.

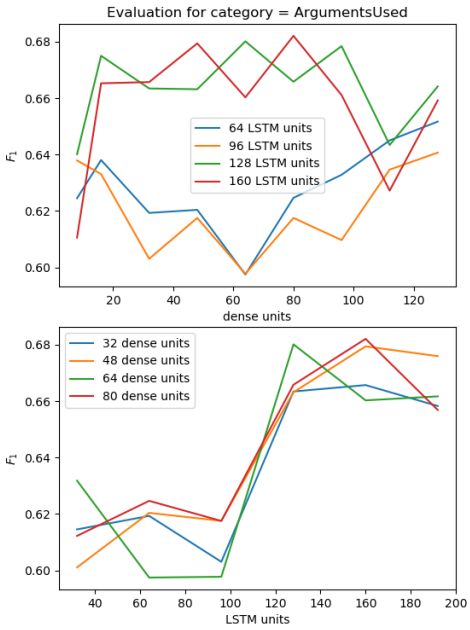

Figure 2: Search for best model based on different hyper parameters

4.3 Training

The dataset of 3599 fully labelled posts is split into a training set of 3059 posts and a validation set of 540 posts (15%).

The training itself is supervised by using automatic learning rate adaptation (*ReduceLROnPlateau*) and *EarlyStopping*, which helps to reduce the overall training time. Figure 3 shows the progress during training and the early stopping method applied, as training ends when validation loss starts to increase.

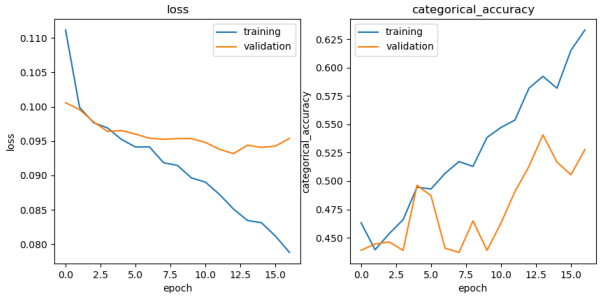

Figure 3: Progress of loss function (left) and accuracy (right) during training

The *categorical accuracy* is shown in Figure 3 as summary over all categories. In fact it is the average of the accuracies in each category, weighted by the rate of positive labels. Later only accuracies for single categories, extracted from a vector of predictions, are analyzed.

	True Pos	True Neg	False Pos	False Neg	Accuracy	Precision	Recall	F_1
Sentiment Negative	167	120	173	80	0.53	0.49	0.68	0.57
	139	205	83	113	0.64	0.63	0.55	0.59
Sentiment Neutral	178	138	111	113	0.59	0.62	0.61	0.61
	199	141	120	80	0.63	0.62	0.71	0.67
Sentiment Positive	0	529	0	11	0.98	0	0	0
	0	531	0	9	0.98	0	0	0
Off Topic	0	458	0	82	0.85	0	0	0
	4	451	3	82	0.84	0.57	0.05	0.09
Inappropriate	0	497	0	43	0.92	0	0	0
	1	487	4	48	0.90	0.20	0.02	0.04
Discriminating	0	493	0	47	0.91	0	0	0
	0	491	1	48	0.91	0	0	0
Possibly Feedback	0	525	0	15	0.97	0	0	0
	0	532	0	8	0.99	0	0	0
Personal Stories	0	538	0	2	1.00	0	0	0
	0	533	0	7	0.99	0	0	0
Arguments Used	85	349	32	74	0.80	0.73	0.53	0.62
	107	332	43	58	0.81	0.71	0.65	0.68

Table 2: Comparison of the performance of the single- (top row in each category) vs the multi-model (bottom row)

5 Results

In this paragraph the performance of the models evaluated by Schabus et al. (2017) and us are compared. While in Table 2 all measurements of the single-model as well as the multi-model of every category is shown, a few categories are highlighted.

In Table 3 the performance of our model classifying the category *Sentiment Negative* is shown. As the corpus for this category is nearly equally balanced this is a good starting point for a comparison. While the precision of our multi-model is higher, the recall is much lower than the one described in the original paper. This therefore results in a lower F_1 score.

	Accuracy	Precision	Recall	F_1
Schabus et al. (best)		0.5842	0.7197	0.6137
Schabus et al. (LSTM)		0.5349	0.7197	0.6137
Our Single-Model	0.5315	0.4912	0.6761	0.5690
Our Multi-Model	0.6370	0.6261	0.5516	0.5865

Table 3: Comparison: Sentiment Negative

Table 4 is an example of the consequences of unbalanced data. While the accuracy is at around 99% the precision, recall and F_1 score is zero. As the corpus only contains 1% of positively labelled data for this category the model learns to predict always zero. But the LSTM model in the original work had the exact same problem, while other models lead to non-zero but still unsatisfying scores.

	Accuracy	Precision	Recall	F_1
Schabus et al. (best)		0.2353	0.4651	0.1333
Schabus et al. (LSTM)		0	0	0
Our Single-Model	0.9796	0	0	0
Our Multi-Model	0.9833	0	0	0

Table 4: Comparison: Sentiment Positive

Table 5 shows the performance on the category *Arguments Used*. In this case both of our LSTM models outperform the original model, as can be seen by comparing the F_1 score. This is especially noteworthy as this category is only applied positively in 28% of the cases.

	Accuracy	Precision	Recall	F_1
Schabus et al. (best)		0.6105	0.6614	0.6098
Schabus et al. (LSTM)		0.5685	0.6458	0.6047
Our Single-Model	0.8037	0.7265	0.5346	0.6159
Our Multi-Model	0.8130	0.7133	0.6485	0.6794

Table 5: Comparison: Arguments Used

6 Conclusion

As expected, our results show that predictions based on highly imbalanced labeled data is hard. Having enough data, some method should be applied that does not reflect the natural ratio. The biased addition of labels in this corpus may not be a good general approach for statistics, but useful for this task.

Considering our single category model, one could reason that all the hidden layers are used to generate features specialized to predict that specific category. In the multi model on the other hand the network has to generate some global features and only a few specialized ones per category. Considering a specific category only, this should result in a better performance of the single model. But in our case the multi model shows better scores in general. The cause may be that high specialization leads to overfitting that we could not prevent by regularization methods. We like to conclude that learning multiple targets at once does help to generate balanced features.

References

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9:1735–80.

Tomas Mikolov, G.s Corrado, Kai Chen, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. pages 1–12.

Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA. http://is.muni.cz/publication/884893/en.

Dietmar Schabus and Marcin Skowron. 2018. Academic-industrial perspective on the development and deployment of a moderation system for a newspaper website. In *Proceedings of the 11th International Conference on Language Resources and Evaluation (LREC)*, pages 1602–1605, Miyazaki, Japan.

Dietmar Schabus, Marcin Skowron, and Martin Trapp. 2017. One million posts: A data set of german online discussions. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 1241–1244, Tokyo, Japan.