Dauphine | PSL
UNIVERSITÉ PARIS

[NLP Project] - Predicting Closed Stack Overflow Questions

Adrien Golebiewski - Clément Preaut

Contents

Introduction

This article discusses a project aimed at developing a method for identifying which questions on the Stack Overflow website should be closed. Stack Overflow is a website where developers from all over the world ask and answer questions related to software development. With over 6,000 questions asked each day, it is difficult to analyze each question and determine whether it should be closed or not. To solve this problem, this article proposes using deep learning and neural network models to identify good candidates for closing.

1 Paper strategy

The basis of our work is an research paper from Levi Franklin titled "Predicting Closed Stack Overflow Questions". It approaches the problem as a sentence classification task. It utilizes a large dataset provided by Kaggle and Stack Overflow, consisting of questions, metadata about each question, and information on whether the question was left open or closed for a particular reason. The first task in developing a neural network model is to model the actual words that make up each of the questions. Word vectors are used to represent the words within each question and consist of numeric vectors of some dimension. These word vectors are a representation of how words appear in relation to each other in a large corpus of documents.

The project uses a pretrained set of word vectors, GloVe vectors, which is well regarded in the field. Additionally, a new set of vectors is trained using the Stack Overflow dataset, which incorporates technical terms specific to Stack Overflow that may not be present in the GloVe set. These two sets of word vectors are compared to determine which performs better.

The paper discusses two main neural network models considered for the project. The first model uses a system that averages the word vectors and uses them as the input to a traditional multi-level neural network. However, this has some shortcomings such as not taking into account sentence structure. The second model uses a convolutional neural network that passes over the words in each question to consider them in groups. Each filter, or convolution, outputs an estimate that is then pooled together, and the result is used to make a prediction on what class the input belongs to. This model does a better job of including sentence structure, but it is more complicated and time-consuming to run.

The article also discusses how to incorporate numeric features in addition to the word vectors. Each question includes some numeric features such as the user's reputation at the time of asking or how many open questions they have at the time of asking. The main method of including these features is to concatenate them onto the end of each word vector so that they can be included in the training. The project compares training with and without these features to determine which approach is more helpful.

To gauge the performance of the models, the article uses the log loss metric, which was used during the Kaggle competition. This allows for comparison with other competitors. Additionally, the loss of the models is compared to some baselines, including making a uniform prediction of each class as the most naïve baseline, as well as one based purely on the class distribution in the training data. Finally, Stack Overflow provides a machine learning model they created to compare against other submissions.

Our project aims to beat more rudimentary baselines and be near many of the other Kaggle submissions in performance. Overall, this project demonstrates the potential of using deep learning and neural network models to solve real-world problems in the software development industry.

2 Improvements

In order to update the method used in the original article, we want to use more recent models and compare them : the recurrent neural networks GRU and LSTM. This method was used on a large scale in NLP due to their ability to memorize information through the input sequence. We want to reproduce the data augmentation performed in the article by adding numeric features in the network and by using the Stack Overflow words embedding. As the paper is not associated with any code,

we implemented the experiments by hand. So it enables us to describe how we performed the data augmentation and run our experiments. We also describe the impact of the dimension of our training data, that is to say the maximum length of a word sequence sample, the number of samples and such.

Moreover, to simplify the experiments, we reduce the problem to predict two classes : close or open. In other words, we do not look for prediction of the close reason.

2.1 Choice of hyperparameters

We first run an experiment with basic Glove word embedding, so no data augmentation, in order to get an intuition over 100 epochs of what learning rate and batch size we should use. It is considered as our baseline.

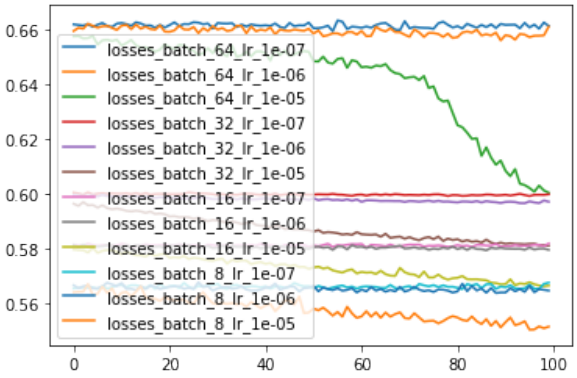

Figure 1: Grid search for GRU model.

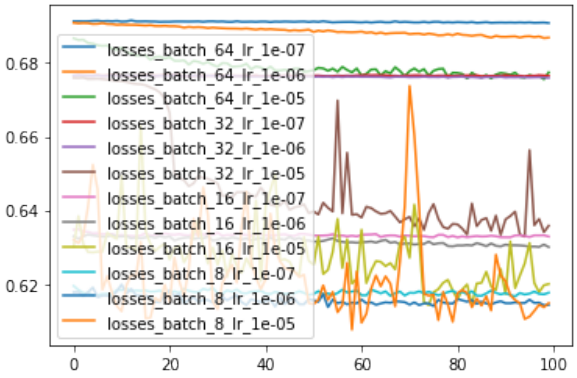

Figure 2: Grid search for LSTM model.

The results are visible in figure 1 and 2. On one hand choosing a batch size of 8 with a learning rate of 10^{-6} leads to the most stable and lowest loss. But the loss quickly converged. On the other hand, choosing a batch size of 64 with a learning rate of 10^{-6} enables one to observe the loss decrease during all the experiment. Even though the loss is higher, it lets us think that we should choose a relatively high amount of data for each batch, greater than 8, if we run the experiment on more than 100 epochs. To be sure to not overwhelm the RAM we first use a dataset of only $20,000$ sample containing at most 100 text tokens.

2.2 Stack Overflow embeddings

As in the paper, we import Stack Overflow words embeddings. We found in a public github the file "SO_vectors_200.bin" containing almost 1.8 million of string tokens associated with vectors of size 200. To save RAM, we reduce thanks to the PCA (principal component analysis) method the vector size to 50.

Stack Overflow vectors are much more numerous than GloVe embeddings which contain 400 000 tokens. Yet, we can surprisingly see common words present in GloVe and missing in StackOverflow embedding. Therefore we add every vector associated with missing words in our vocabulary. Later, if a token in a training sample is not present in our vocab, we simply associate it to a null vector of size 50.

We can first compare an experiment using LSTM with a learning rate of $1e-5$ and a batch size of 64 to what we got using only GloVe embedding. We see in figure 3 the advantage brought by Stack Overflow embedding by the loss at epoch 100 which reaches about 0.59 against 0.68 for the equivalent hyperparameters baseline experiment.

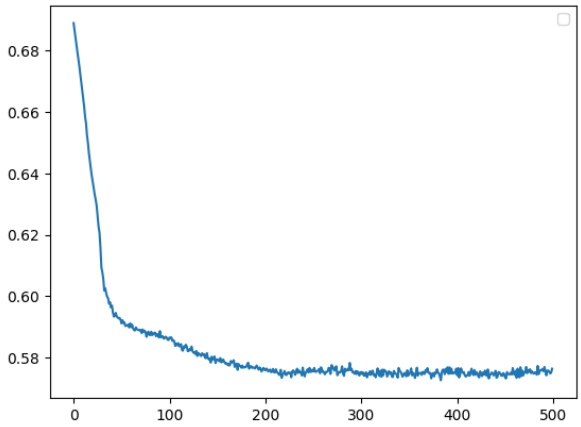

Figure 3: Experiment using LSTM and Stack Overflow embedding with learning rate 10^{-5} and batch size 64.

Then we compare GRU and LSTM with the same hyperparameters. On figure 4, we can see that GRU reaches a lower loss of 0.56 than LSTM at the end of 500 epochs.

2.3 Numeric features

As mentioned previously, several tests and improvements in the initial solution preprocessing have also been implemented.

Indeed, we wanted to further enrich the inputs of our deep learning models with numerical data. Some of these data come from the initial dataset (for example, the "score" given for each post). The others were created from the initial data by post processing and by a POS (Part Of Speech) approach.

We engineered features related to the parts of speech and sentiment of a post. In this context, parts of speech refer to the number of nouns, verbs, adverbs, and adjectives in the title and body of a post. Sentiment refers to the tone or attitude of the post as represented in the title or body of the post and takes on a value between less than zero. Where a sentiment is less than one, it is considered negative, a sentiment near zero being neutral, and a sentiment greater than zero being positive.

These engineered features have been included in our initial data set to enrich our data. A complete dataset containing these new data could thus be built, as a basis for our trainings.

In the reference paper, the convolutional approach tested is performed by concatenating the numeric features at the end of each word vector. However,in our study, numeric data were concatenated to the output obtained from the chosen learning method (lstm, gru) before applying a "dense" function.

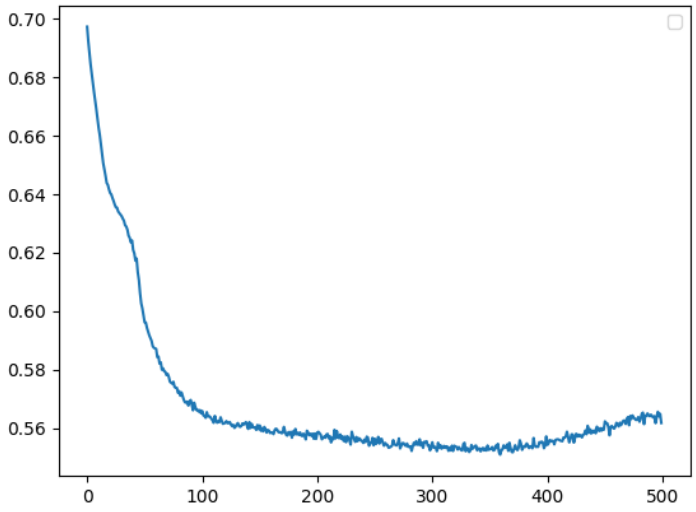

Figure 4: Experiment using GRU and Stack Overflow embedding with learning rate 10^{-5} and batch size 64.

Two comparisons were made to evaluate the contribution of numerical data in learning. The comparison will be based on the loss value after 500 epochs:

- the one between our GRU/LSTM + Stack Overflow embedding approaches with numeric features and without.

- the one between our GRU/LSTM + GloVe embedding approaches only with numeric features and without.

As illustrated in the two figures below, in both cases, we notice that our GRU/LSTM approaches with numeric features reaches a lower loss (around 0.57 without StackOverflow embedding and around 0.55 with) at the end of 500 epochs.

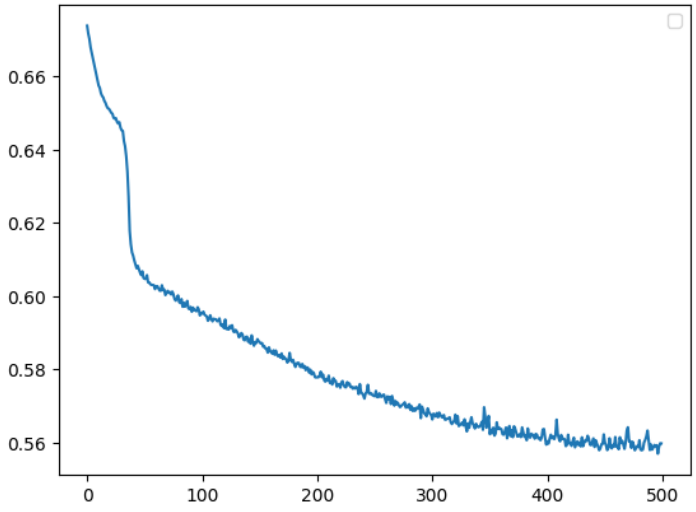

Figure 5: Experiment using GRU and Stack Overflow embedding with numerical features taking acount

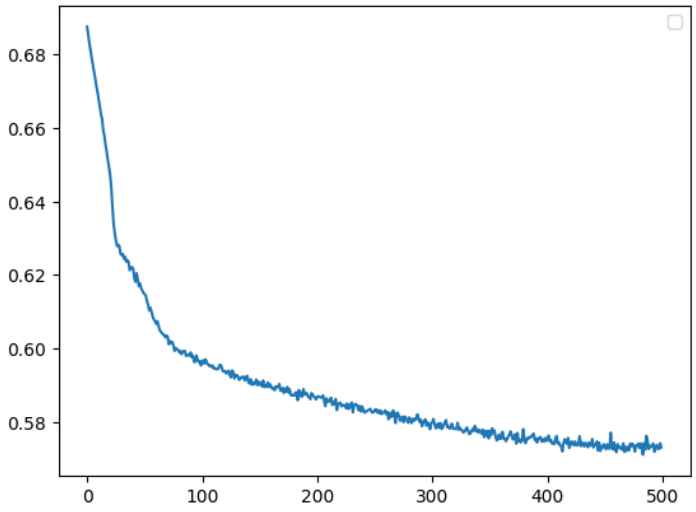

Figure 6: Experiment using LSTM without Stack Overflow embedding and with numerical features taking acount

These results confirm the ones obtained in the reference paper : the numerical data extracted by features engineering make the model more robust. However, the contribution of these numeric features is not very significant (decrease of 0.04 for the loss) and does not bring a real added value in our different tests. It would have been interesting to evaluate the contribution of these numeric features by the "features importance" technique which assigns a score to input features based on how useful they are at predicting a target variable. This work could better specify the impact of these features and thus reconsider our feature engineering step.

2.4 Dimension of data

We ran another experiment with a learning rate of $1e-5$ and a batch size of 64 but with $50,000$ samples containing at most 150 text tokens. The results of the training are visible on figure 7 below. It reveals the importance of the quantity of the data to be processed to increase the quality of the result, because loss during the training reaches about 0.55 against about 0.58 with the previous settings.

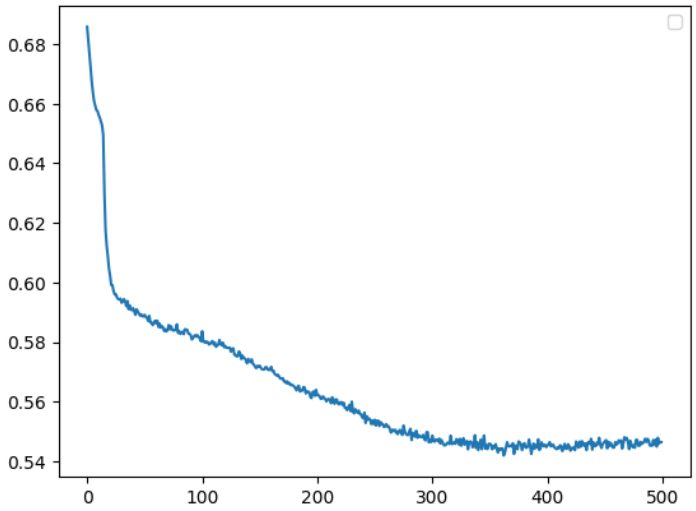

Figure 7: Experiment using LSTM and Stack Overflow embedding with $50,000$ samples and sequance maximum length of 150. The learning rate is 10^{-5} and batch size is 64.

3 Conclusion

We did not manage to reach the same results as in the paper but we could assume that with more epoch and better hyperparameters grid search, we could reach even more interesting results. Same reflexion if a more developed approach of numeric features had been realized.

The experiments can more be considered as fonctional application of deep neural model and proof of the benefit of data augmentation of Stack Overflow embedding and numerical features. We can notice that nowadays, GRU and LSTM are deprecated and replaced by Transformers. Our reference paper, dating back more than 10 years, did not take into account these latest state-of-the-art techniques. We could then continue the experiments with current state of the art. For example, it would have been interesting to applying BERT method to just the review text, and ignoring all of the other features.

The simplest way to handle these non-text features would have been to convert them into text and concatenate them with the review text. Being a state of the art method, one could assume that it outperforms the results obtained through this study.

Due to RAM limitations, we were limited in the word embedding size and the text sequence and so overall in the experimentations tests. This project was demanding and time-consuming but helped us to really dive into a real NLP project.

4 References

[1] Levi Franklin. Predicting Closed Stack Overflow Questions - Stanford University, 2013

[2] Predict Closed Questions on Stack Overflow. https://www.kaggle.com/competitions/predict-closed-questions-on-stack-overflow/code, 2013.

[3] Pradeep Kumar Makkena. PREDICTING CLOSED VERSUS OPEN QUESTIONS USING MACHINE LEARNING FOR IMPROVING COMMUNITY QUESTION ANSWERING WEBSITES, 2017.

[4] Yixing Lao, Chenwei Xie. Why is My Question Closed? Predicting and Visualizing Question Status on Stack Overflow

[5] StackOverflow Word2Vec, https://github.com/vefstathiou/SO$_w ord2vec$