NLP Project report :
Semantic question matching

Corentin Guérendel, Vincent Gouteux, Vincent Henric, Clara Simmat

15 May 2020

Contents

1 Introduction

The goal of the project is to participate at the Kaggle competition: "Quora Questions Pairs". Quora is a question-and-answer website and more than 100 million people visit it each month. Therefore, same question can be asked multiple times. So, people will waste time to answer the same question (or to find the best answer). The aim of this challenge is to identify which of the provided pairs of questions contain two questions with the same meaning.

1.1 Data

You can find the dataset on: Kaggle. Here are the different features:

- id - the id of a training set question pair;
- qid1, qid2 - unique ids of each question (only available in train.csv);
- question1, question2 - the full text of each question;
- is_duplicate - the target variable, set to 1 if question1 and question2 have essentially the same meaning, and 0 otherwise.

2 Approaches

2.1 Cleaning

This dataset is a quite clean one. Regarding the task we wanted to perform, the cleaning can be different. Indeed so task need more pre processing that other, which will require only tokenization. The cleaning simply consists in removing stopwords, punctuation or unknown characters. We also make sure that every question is a string, in english. But, as we will see, the tokenization can do it most of the time.

2.2 Tokenization

Once we have removed every undesirable character or word we have to tokenize our data. Although tokenization seem very easy is it absolutely essential to perform machine learning tasks on text. A tokenizer removes the punctuation and converts a string in a sequence of integers. It associate each word to its index in a given dictionary. Bert tokenization has the specificity to consider sentences and tokenize them. It has special tokens [UNK],[PAD],[CLS],[SEP],[MASK] that can be very useful to perform stat of the art NLP.

We can see that, depending on the cleaning and the tokenization techniques, the results can be different. We did not investigate much the difference in results in the downstream tasks, but we could observe a sensible difference in the numbers of matches with pretrained embeddings. For many methods that reused pretrained embeddings, this can be important. In the notebook Benchmark

embeddings, we propose a small experiment. We compare three processes for cleaning and tokenization. We measure the number of unrecognized tokens in the pretrained embeddings of Google News.

The tokenizer that we try are summarized in the following table:

Method name	remove punctuation	parse xml	tokenize function	remove stopwords	lemming
tokenizer			nltk.tokenize		
tokenizer1	x	x	str.split()	x	x
tokenizer2	x	x	nltk.tokenize		

Table 1: Characteristics of the different preprocessing functions

The following plots show to what extend there can be a difference in the recognition of the tokens depending on the preprocessing, by word2vec. Of course, a best practice would be to retrain the embeddings, or use exactly the preprocessing used to learn the embeddings.

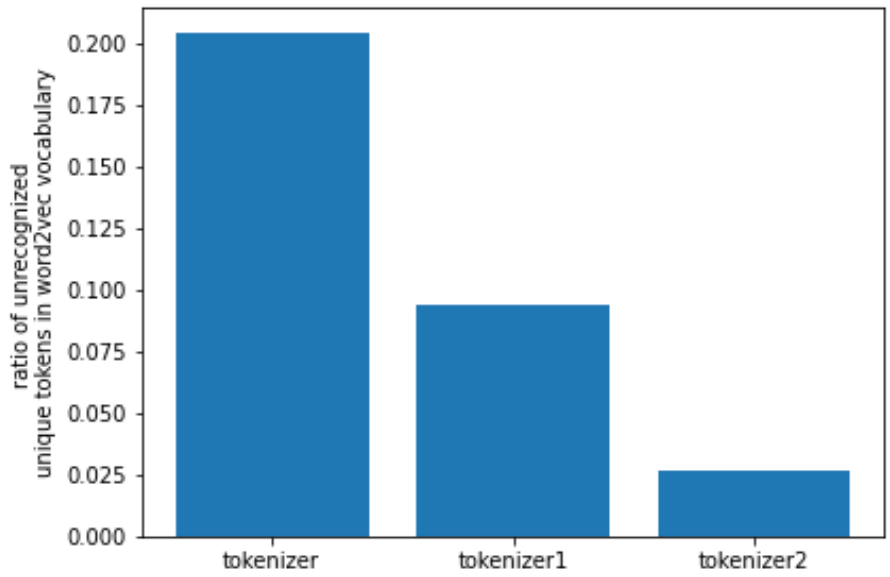

Figure 1: Ratio of unrecognized unique tokens in word2vec vocabulary

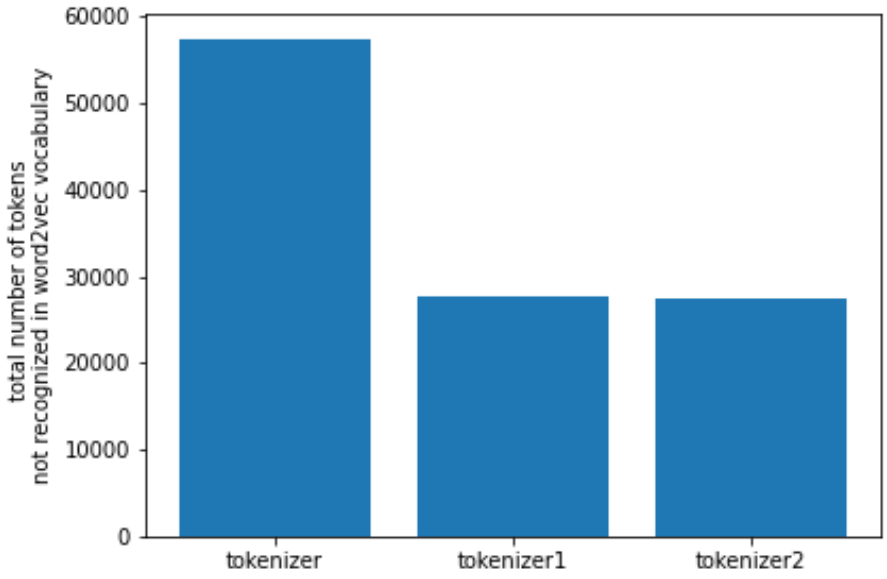

Figure 2: Total

2.3 Not so naive features

To be able to compare our results with naive approaches, we computed very naive and basics features together with Fuzzy features. First, the very basic features included length-based features and string-based features:

- Length of question1 L1 (respectively question2 L2)
- Difference between L1 and L2
- Character length of question1 (respectively question 2) without spaces
- #words in question1 (respectively question2)
- #common words in question1 and question2

Fuzzy features are answers to the problem of finding strings that match an approximate pattern rather than an exact match. The closeness of a match is defined by the number of primitive operations necessary to convert the string into an exact match. These features were created using the *fuzzywuzzy* package available for Python. We included in our analysis all fuzzy ratios in this list:

- QRatio
- WRatio
- Partial, partial token set ratio and partial token sort ratio
- Token set ratio and token sort ratio

2.4 Bag of words approach

We simply use two methods based on word counts:

1. simple word count
2. tf-idf features

Once obtained, we perform a singular value decomposition to get features for the questions, in other words some kind of sentence embeddings.

2.5 Sentence embeddings

We tried different sentence embeddings. The idea was to do some transfer learning on pretrained models, and see to what extend these models could be useful for our use case. In the described models, we did not try to retrain, only to use on the shelf models.

The models we used are the following;

- Doc2Vec
- Sent2Vec
- Sentence BERT (SBERT)
- SIF embeddings

These methods have various characteristics, that we will briefly describe thereafter. We chose these approach because of their communicated efficiency, and the easiness of re-using open source libraries. See [1] for a detailed literature review on sentence embeddings. All of these embeddings are unsupervised embeddings.

For all of these methods, we tried to use logistic regression and a simple feedforward neural network on top of the pretrained sentence embeddings. We compare all the results for the training and validation sets.

2.5.1 Doc2Vec

In Doc2Vec [2], also known as Paragraph Vectors, we generalize the learning of word-embedding from the word2vec model. For each sentence, we have a given embedding for each of the words and for the sentence. In the Distributed Memory Model of Paragraph Vectors (PV-DM), the embeddings of the words of the context window and the embedding of the sentence are concatenated, and we try to predict the word which has the given context in the given sentence. The optimization is of the same type as for word2vec. In the Distributed Bag of Words version (PV-DBOW), we try to predict words of a window of the sentence, from the sentence embedding. At inference, word embeddings are held fixed, and some gradient descent steps are performed to find the appropriate sentence embedding.

2.5.2 Sent2Vec

Sent2Vec [3] is another attempt to construct sentence embeddings, and can be seen as an extension of the Continuous Bag of Words model. In short, instead of a fixed window of words as a context, we use a whole sentence. The sentence embedding is obtained as the average of the word embeddings of the words inside the sentence.

2.5.3 SBERT

BERT approach, that we saw in class, is based on attention [4]. It appears that the embeddings given by BERT are not that performant for transfer learning and to be used for other tasks than the task it has been trained for. Sentence BERT is a remedy to this, and claims to give multi-purpose embeddings based on BERT architecture [5]. In SBERT, BERT model is simply learnt as the input of a siamese net with a cosine similarity, to learn similarity between two sentences. For different purposes, especially sentence similarity, using such embeddings looks like a good start, and we did give it a try. To measure the performances of the sentence BERT encoding we simply compute all sentences embeddings and classify them by applying a cosine similarity and a certain threshold. The training phase enable us to determine the threshold that gives us the best accuracy.

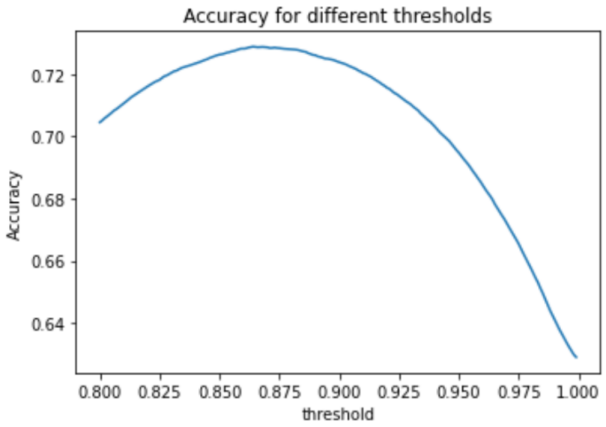

Figure 3: Accuracy for different thresholds

2.5.4 SIF embeddings

SIF embeddings [6] is simply based on a smart linear combination of word embeddings of a sentence or a document. Instead of going for the naive averaging, we tried this implementation, where the weights are of the form:

$$\frac{a}{a + p(w)}$$

where a is a parameter and $p(w)$ is the word frequency, or an estimation. This formula stands for SIF (smooth inverse frequency). We directly used a standard value for a as recommended in the paper. It has the advantages of being very simple, very fast compared to encoder-decoder schemes, and reasonably good.

We use the implementation of O. Borchers [7].

2.6 Training of neural networks

2.6.1 Siamese Net based on attention

Attention has shown great effectiveness in natural language processing and is characterized by its simplicity. The attention mechanism consists in matrix and vectors dot product in order to detect links and influence between words. We decided to implement a Siamese neural network with an "attention layer". And the results are quite impressive despite its simplicity.

- **Preprocessing** : We will clean and pre-process the raw data with the tools given by Keras. The keras Tokenizer is very simple, it removes punctuation and converts a sentence in a sequence of integers (each integer being the index of a token in a dictionary). We then pad sequences so that all inputs have the same shape.
- **Model** : The model is quite heavy as embedding, bidirectional and LSTM layers have a lot of parameters. However the architecture is really simple as we can see on the figure below.

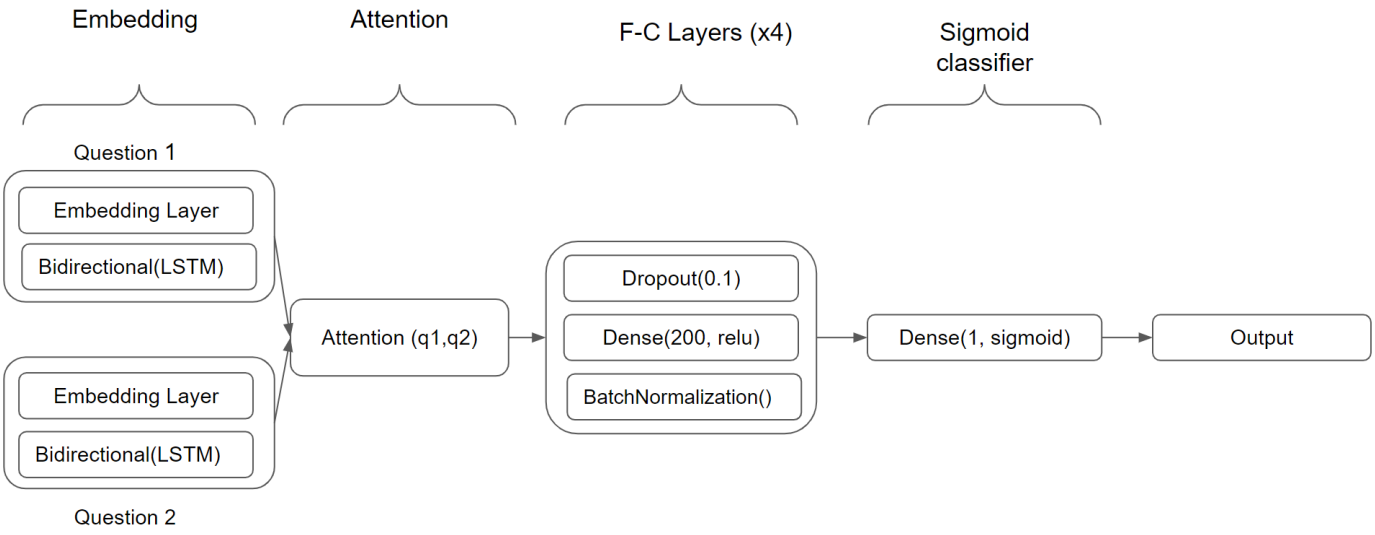

Figure 4: Architecture of the network

As it is a Siamese network we embed the questions separately and then concatenate the embeddings. We then have an attention layer connected to a Fully-Connected network and finally a sigmoid classifier.The embeddings used for the embeddings layers are Glove pre trained embeddings.

2.6.2 Manhattan LSTM (MaLSTM)

Another approach with siamese net is the MaLSTM, put forward in [8]. The idea is to compute the Manhattan distance of the outputs of LSTMs. Then, taking the opposite of the exponential, we make sure to have values between 0 and 1. We can easily evaluate the output and train the model. We notice that a MSE loss gives better results in terms of accuracy than a binary crossentropy loss. The disadvantage is that we do not have an optimized and relevant binary crossentropy loss.

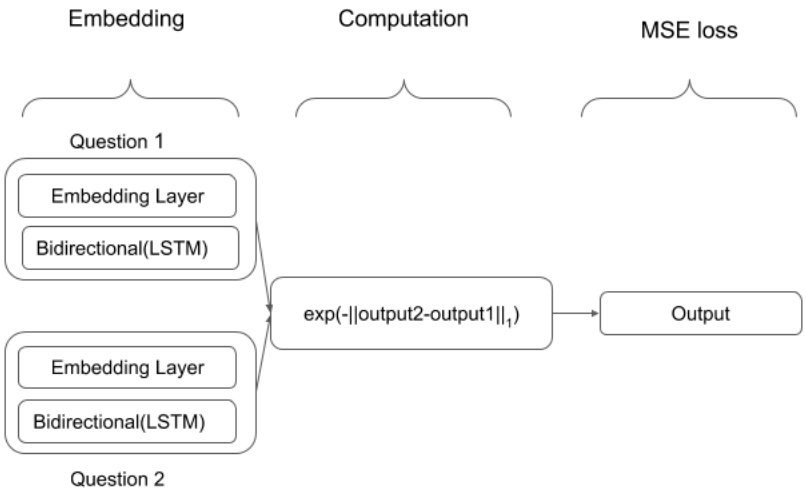

Figure 5: Architecture of the network

2.7 Mixing it all

We try in this section to mix a few approaches, in an ensembling model. We want to see whether we can obtain better results.

We combine features from sentence embeddings and basic and not so naive features. For the sentence embeddings features, we use only the cosine similarity.

Otherwise we could use the prediction for the models built on top of all of the sentence embeddings, as features. These results are not reported. The features are then fed into a gradient boosting algorithm and evaluated. With the cosine similarity, we do not attain performances of a neural network built on top of the embeddings. With the prediction probabilities taken from a neural network out of the embeddings, we obtain the best performance.

3 Results

3.1 Metrics and results

We can see a slight imbalance in the classes. We used the binary crossentropy as the loss for the optimization, and we also monitored the accuracy. Note that, because of the imbalance in the dataset, the naive and immediate baselines would be:

1. For the loss, to give to all pairs of sentences a probability corresponding to the frequency of the classes; it gives a baseline of 0.66

2. For the accuracy, to always predict that the sentences are not duplicates, as it is the main class; it gives a baseline of 0.63

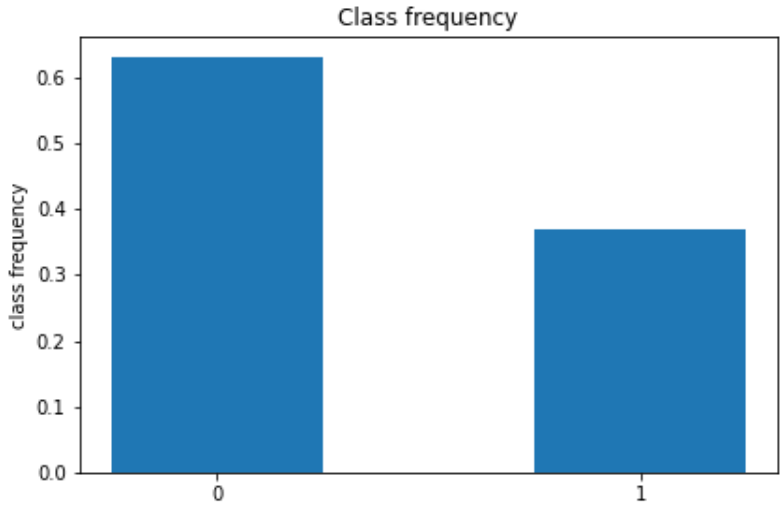

Figure 6: Class frequency

Because the provided test set does not have any labels, and is of different composition (automatic generation vs real questions in the training set), we rather considered our results on a validation set extracted from the training set. We make sure to keep the same frequency for the classes.

	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
Basics&Fuzzy-LR	0.56	0.66	0.55	0.66
SIF-Logistic	0.6	0.69	0.59	0.69
Doc2Vec-Logistic	0.54	0.71	0.54	0.71
Sent2Vec-Logistic	0.56	0.72	0.56	0.72
Basics&Fuzzy-xgboost	0.49	0.72	0.49	0.72
Cosine-Bert	x	0.72	x	0.73
SIF-NN	0.49	0.75	0.51	0.74
ensembling-cosine	0.40	0.79	0.41	0.79
SBERT-Logistic	0.41	0.81	0.42	0.80
MaLSTM	x	0.91	x	0.82
Siamese Net Attention	0.32	0.85	0.39	0.82
Sent2Vec-NN	0.35	0.84	0.38	**0.83**
Doc2Vec-NN	**0.34**	**0.84**	**0.37**	**0.83**
SBERT-NN	**0.34**	**0.84**	**0.37**	**0.83**
ensembling-NN	**0.27**	**0.88**	**0.32**	**0.85**

Table 2: Loss and Accuracy for different methods

For the sentence embeddings, we obtain reasonably good performances. They clearly outperform more naive methods.

Figure 7: Performances of a logistic regression on sentence embeddings

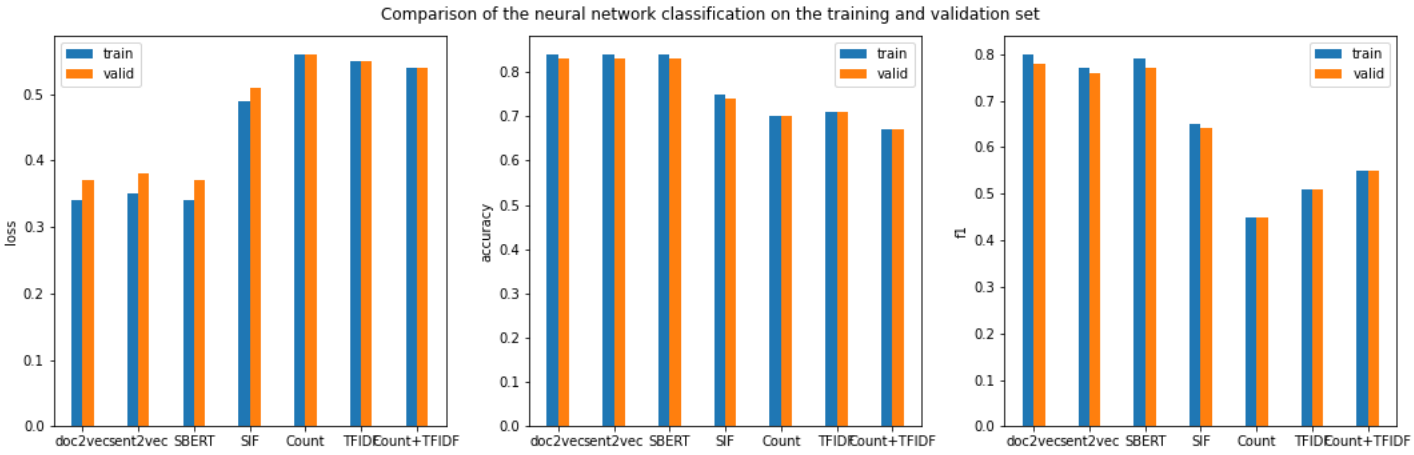

Figure 8: Performances of a logistic regression on sentence embeddings

We can clearly see that the embeddings carry by themselves a fair amount of information, because a simple logistic regression has a reasonable performance, especially for SBERT. Simple feedforward neural network help leverage this information, and enable to achieve good performances for the three pure sentence embedding techniques. SIF embeddings, based on a linear combination of word embeddings, also scores fairly well, but does not enable to retain as much information as we have lost the word order in the sentence. This can be of great importance to understand finely the meaning of the questions.

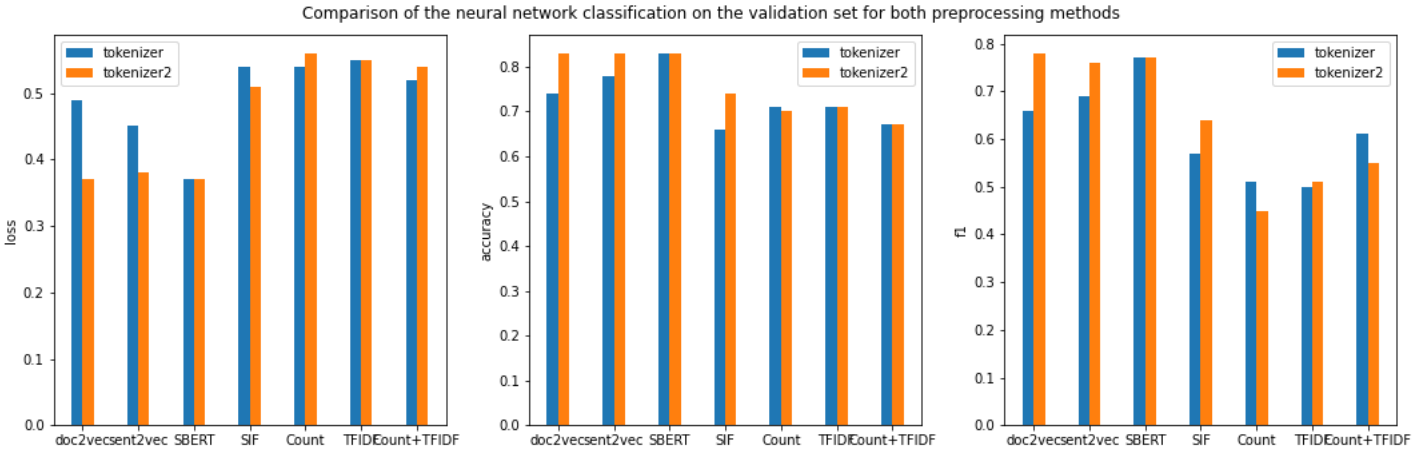

Figure 9: Comparison of performances of neural network models of two differently preprocessed tokens

Also, we can see that a better preprocessing help achieve better results, as we can see in Figure 9

3.2 Critics and improvements

In our project, we focused on pure NLP aspects of the kaggle problem. But we could also have taken into account the left aside features, in other words the identifier of the questions. Some questions have been asked multiple times, and we could expect a question that comes multiple times to be part of duplicates. More generally, the sparse graph structure of the identifiers (there is a link between two identifiers if they are associated in one row) could help uncover such type of information.

Also, we did not use any syntactic information of the sentences. But we could also well imagine that difference in structures could mean different meanings for the questions.

Also, we did use only a validation set. We have reasons to believe that the provided test set will not be exactly of the same distributions (some sentences are artificially created, and not extracted from Quora) and the labels were not provided.

References

[1] S. Palachy, "Document embedding techniques." Available at {https://towardsdatascience.com/document-embedding-techniques-fed3e7a6a25d}.

[2] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Proceedings of the 31st International Conference on Machine Learning* (E. P. Xing and T. Jebara, eds.), vol. 32 of *Proceedings of Machine Learning Research*, (Bejing, China), pp. 1188–1196, PMLR, 22–24 Jun 2014.

[3] M. Pagliardini, P. Gupta, and M. Jaggi, "Unsupervised learning of sentence embeddings using compositional n-gram features," in *NAACL-HLT*, 2018.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *ArXiv*, vol. abs/1706.03762, 2017.

[5] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *EMNLP/IJCNLP*, 2019.

[6] S. Arora, Y. Liang, and T. Ma, "A simple but tough-to-beat baseline for sentence embeddings," in *ICLR*, 2017.

[7] O. Borchers, "Fast sentence embeddings." https://github.com/oborchers/Fast_Sentence_Embeddings, 2019.

[8] Z. Imtiaz, M. Umer, M. Ahmad, S. Ullah, G. S. Choi, and A. Mehmood, "Duplicate questions pair detection using siamese malstm," *IEEE Access*, vol. 8, pp. 21932–21942, 2020.