NLP Project: Quora duplicate detection

Justin AYIVI, Samy FERRAT, Othmane ZARHALI and Sebastien CHAILLOU
Université Paris-Dauphine

I. Dataset presentation

The dataset is a question set from the Quora site, it is broken down into a train set of 404290 rows and a test set of 1048574 rows.
It is composed of 4 columns 'questionsid': Identifier of the pairs of columns, (qid1,qid2): Corresponding to the individual identifiers of the questions (only for the ream), ('question1','question2'): text corresponding to each question, 'is duplicate': Target variable that indicates whether the two questions have the same meaning.

II. Preprocessing

A. Text cleaning

To perform the dataset preprocessing, we first proceeded by "cleaning up" the text, i.e. we replaced contractions with their original forms using a predefined dictionary (for example "can't" with "cannot"), we also replaced common errors or different forms of writing the same thing (eg bestfriend with best friend), this then helps to avoid unnecessary confusion for the model.

B. Word2Vec

Concerning the Word Embedding, we used the Skip-Gram variant of the Word2Vec algorithm, the Word2Vec algorithm does not require any labeling, it is an unsupervised learning algorithm, the Skip-Gram variant seeks to predict the words of the context from a central word.

Indeed, the outputed train set and validation set are 2D tuples compound of:

- (Question1,Question2) vectors
- is_duplicate boolean

Here is a synthesized vision of the preprocessing step:

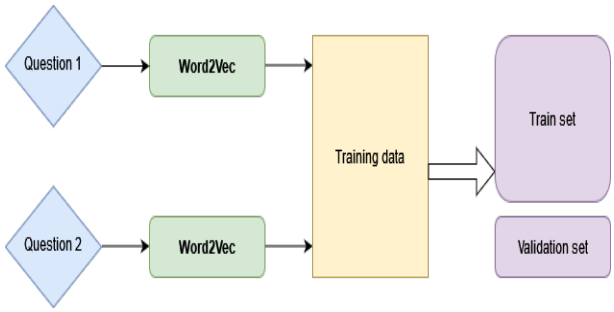

Fig. 1: Preprocessing step

III. Evaluated models

In the literature, many models have been tested with and without a RNN based (LSTM most of the time) attention models. For instance, in [3] we have the following performances:

Model Class	Model	Test Results Accuracy (%)	F-score
	Most frequent class	63.1	-
Linear	LR with Unigrams	75.4	63.8
	LR with Bigrams	79.5	70.6
	LR with Trigrams	80.8	71.8
	LR with Trigrams, tuned	80.1	71.5
	SVM with Unigrams	75.9	63.7
	SVM with Bigrams	79.9	70.5
	SVM with Trigrams	80.9	72.1
Tree-Based	Decision Tree	73.2	65.5
	Random Forest	75.7	66.9
	Gradient Boosting	75.0	66.5
Neural Network	CBOW	**83.4**	**77.8**
	LSTM	81.4	75.4
	LSTM + Attention	81.8	75.5
	BiLSTM	82.1	76.2
	BiLSTM + Attention	82.3	76.4

Fig. 2: Models evaluated in the [3]

A a result, the models that seem to perform well are those based on three main complexity levels :

- **Level 1**: feature encoding via a word embedding technique
- **Level 2**: Memory dependency, which is obvious for a language model
- **Level 3**: Context inclusion via attention models

In this project, we have performed a tailored language model that incorporates the level 1 and 2:

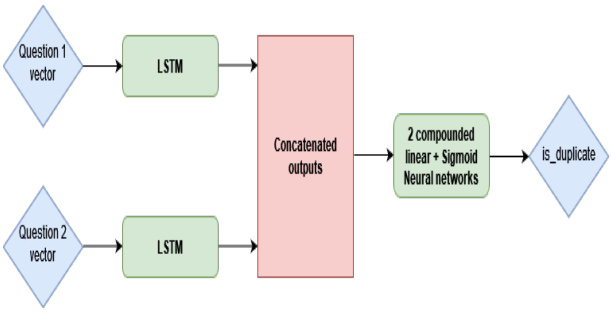

Fig. 3: Language model

IV. Code architecture

The code is decomposed into different entities summarized as follows:

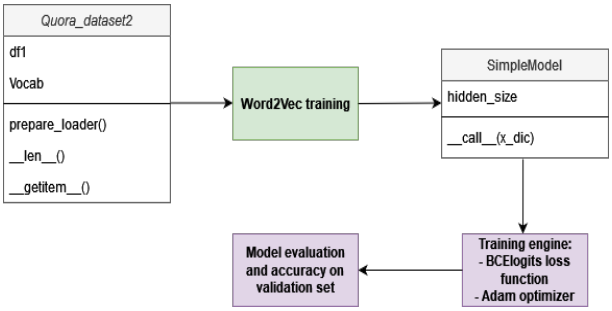

Fig. 4: Language model

The easiest part was to implement the Word2Vec model thanks to the documentation provided by the Gensim website and to some notebooks which showed us how to train a model on our dataset. We focused on the train set provided by Kaggle because we wanted to monitor the accuracy rather than rely on the score given after a submission but even if we further divided this set into train, valid and test subsets, we trained the Word2Vec model on the whole set because the hidden part is not the questions but their duplicate nature.

On Kaggle, we found that the most voted notebooks were variations of recurrent networks implemented with Tensorflow. On the other hand, Pytorch gained traction more recently and it isn't well represented in this competition which is 5 years old. We found inspiration in [3] but it uses Glove pre-trained Glove representation and it was more challenging for us to use our trained Word2Vec model.

One of the difficult part was to elaborate a custom loader as we had to handle two different questions and convert into lists of vectors with our representation. Fortunately the Dataloader of Pytorch provides a feature "collate_fn" that allowed us to elaborate our customization while keeping a decent loading speed. We fixed a maximum for the sequence lenght because most questions contain around 20 words while some contain more than 100 and we didn't want vector representations full of zeros.

As showed in other notebooks, we understood the interest of concatenating the outputs of the LSTM layers associated with the two questions as Pytorch doesn't have difficulty in back-propagating the gradient.

Two of us have a GPU on their computer and we were able to speed up our code with CUDA. We also managed to run our code on Google colab with the subtle nuance that we had to update the version gensim. The data is imported from our google drive as the file is too big for github (more than 25Mb).

V. Results

Using the previous language model, we reached an accuracy on validation set of 72.10%. This accuracy can be enhanced taking into account the following tips:

- Training on a larger dataset
- incorporating an attention model
- Training using an elaborated optimization algorithm
- Adding complexity in the neural networks (depth and highly non linear activation functions)

An important part of the modeling is the hyperparameters tuning. It allows to improve the performances of the model and most often comes after the choice of an architecture that works rather well and a good choice of the optimizer. Given the complexity of the task, we explored different methods.

Based on the literature, we initially considered the following sizes for the different hyperparameters: a size of 32 for the batch size, a number of epochs of 30, a depth of the hidden layers of size 50, a binary cross entropy loss function and an lr = 0.01. With this configuration and the initial model we reach a validation score of accuracy: 0.7394. By changing the lr from 0.01 to 0.001, the accuracy is now 0.7610. By adding a dropout, the loss decreases slightly but the accuracy deteriorates a little bit to 0.7464. We also explored the different activation functions which did not seem convincing either with regard to the accuracy except for the activation function of the Relu type. Finally, we added a fourth fully connected layers after the two lstm layers and this allowed us to considerably reduce the global loss of the model on the train by presenting an accuracy of 0.7455 on the validation set. The fact that the accuracy doesn't increase much is a sign of overfitting. In summary, we obtain the following performances.

Model	validation accuracy	test accuracy
2 LSTM + 3 FC + Dropout	0.7824	0.7822
2 LSTM + 4 FC + Dropout	0.7455	0.7522

***p.s**: the code available here: https://github.com/SebastienChaillou/Quora_questions_NLP*

REFERENCES

[1] Lakshay, S., Graesser, L., Nangia, N., and Evci, U. *Natural Language Understanding with the Quora Question Pairs Dataset*

[2] Lakshay, Rehurek, R., Sojka, P. *Gensim–python framework for vector space modelling*, NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic, 3(2). 2011

[3] CHAFEKAR T. *Quora Question Pairs Bi-LSTM Pytorch* 2020