

NLP Project

Question answering system

Christophe Arendt, Sebastien Bordonnat, Maud Galmiche,
Marceau Guiot and Thibaut Nicot
30th of May, 2019

Guidelines

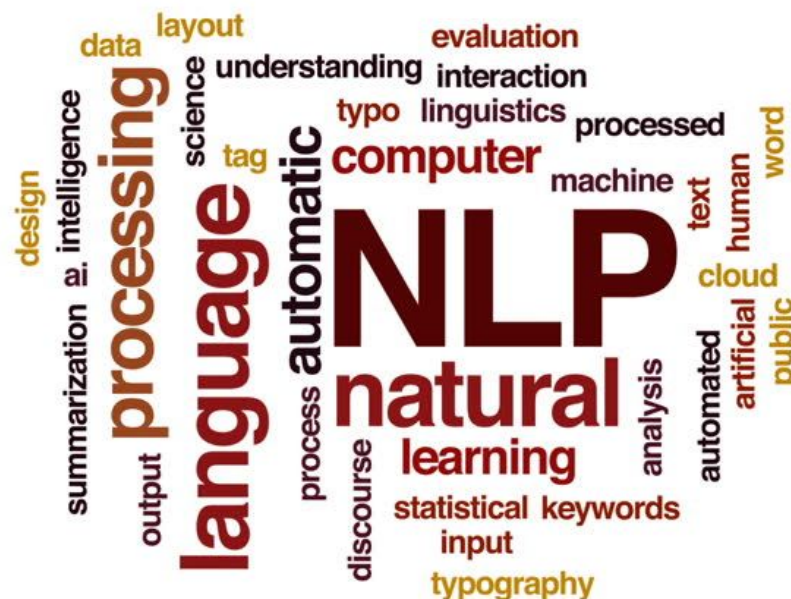
You will be asked to predict/find the best answer if any exists (see **is_best_answer**). Note that, there are many questions with no answer or no best answer. The best answer was usually the one that received the most votes. Of course, if the message was deleted, it's not the best answer.

The value of the field/column/variable "question_answer_or_comment" indicates whether the corresponding line is a Question (Q), an answer (A) to a question (use parent_id to retrieve the question to which the answer is given), or a comment (which is usually associated to an answer).

Please note that one question can have multiple answers (that's why you will be finding out the best one), and one answer can have multiple comments.

Advice: you don't need to use all the variables available. You can start by using only questions and answers (i.e., not comments), and then add comments to your models if they could improve the performance of your model.

Do not hesitate to let me know if you have any questions regarding this project.





Step summary

Guidelines	1
Step summary	2
Define problem	3
Data preparation	3
Importing drive:	3
Importing data:	3
With a customized method	4
Checking imported data	4
Convert time to date	4
Removing comments	4
Text cleaning	4
Stopwords:	4
Descriptive analytics	5
Data visualization	5
Models	8
Experiments	10
Results and conclusion	10
Next steps	11

Define problem

Task:

In this NLP project, we are asked to predict the best answer if any exists. In the dataset provided, there is not always a best answer for each question. Another issue is that there are some comments upon some answers. In the first time, we will build a model only upon answers.

How did we organise ourselves?

For this project, we decided to work on Google colab. In fact, as students, we did not have access to a GPU with our laptop so we decided to use the one freely provided by Google colab. To collaborate, we decided to write all our code on a notebook. Since this project has non-intent to be put in production, we think that would be the most convenient format to work on it.

Colab allowed us as a group to improve our coding skills in Python programming language but also develop Deep Learning applications using popular Python libraries such as TensorFlow, Torch and Flair. Furthermore, we used a development environment (Jupyter Notebook) that does not require any configuration. The feature that distinguishes Colab from other services is the access to a GPU graphics processor, which is completely free of charge! Furthermore, no installation is required.

Data preparation

- Importing drive:

In order to have access to the dataset provided, we mounted our drive containing the CSV file. This operation allows us to import files from the drive but also to write and save files on the google drive. We just have to copy the authorization code to access the drive.

- Importing data:

As explained before, the data export-forum_en.csv are some format issues. In spite of these issues, we try to force the importation using pandas to have a first glance at the data.

Even if at first sight, the importation seems to have worked, we can see after a quick analysis that we have faced some issues. In fact, after a few lines, the importation process shifts all the columns, damaging the data.

With a customized method

In the first time, we decided to work with the importation excluding the data corrupted with a simple filtered. Since our NLP professor M. Anh-Phuong decided to give us another method to import the whole dataset, we switched to his method.

Checking imported data

Once the CSV is saved, we can easily check that everything went well transforming our target variable into a set.

Even if everything seems well during the importation, we are still facing a few issues. In fact, the two columns date and last_answer_date are in second elapsed since the 1st January 1970. In order to have an appropriate view of this data, we transform these two columns in the proper date.

- **Convert time to date**

In order to know in which the time is represented we search on the web the forum related to the first question: [click-here](#). Once this is done, we just have to be sure that every column is in the right format.

- **Removing comments**

As explained before, we start by building a model without considering the comments.

- **Text cleaning**

Stopwords:

One major issue we are facing with the dataset provided is the fact that this one is multilingual. This caused trouble for both cleaning text and the embedding. In fact, the library *nltk* provided some stopwords to clean the text but those are specific to one language. What can be done here is to build a model to detect the language of each question. Another less time consuming solution is to concatenate the list of stopwords for each language. This [Git-hub](#) link provides a JSON file gathering stopwords from multiple languages.

Here we are using a set instead of a list to decrease the computing time. Since we have now defined our stopwords we can define a function to clean the text. First, we have to choose the variable on which we will apply the function.

Since we want the information on both the question and the message, we decide here to concatenate both columns. Once this done, we will be able to get rid off the title of each question since it will also be contained in the message.

The data provided might certainly come from scrapping. That is why we need to clean all the HTML tags first. Once this is done we can replace all the punctuations characters with spaces and convert all the words to lowercase. At the end, we can apply our tokenizer on the text. Here we decided to use the *TreebankWordTokenizer*. At the end of the day, we check of course if the word is a stopword before adding it to our message.

As explained before, we start with concatenating both columns message and title. Once the concatenation is done, we can apply our function to clean the text.

Descriptive analytics

- **Data visualization**

One of the most important things to do here is to check how balanced the labels of the target variables are. Of course, as we have expected, the best answers are really scarce in our dataset. We have to keep that in our mind when will analyse the results of our model.

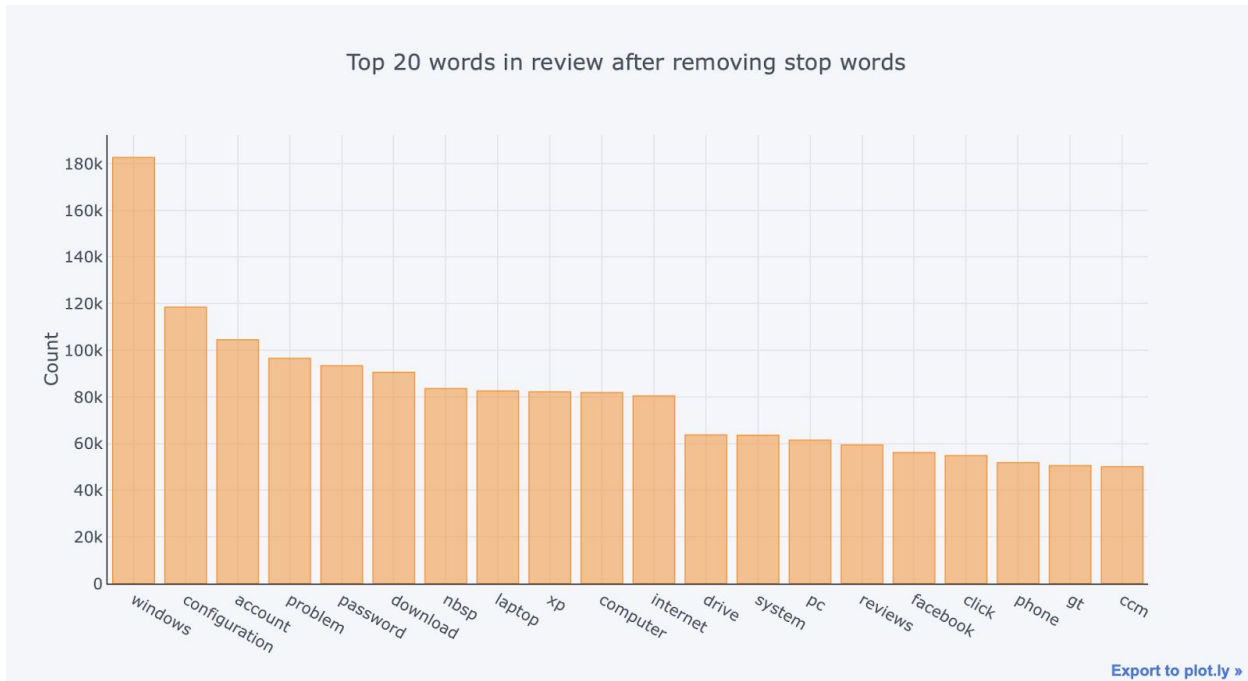


In order to see which are the most common words, we build a bag of words. Once this bag is built, we can sort it in order to have the most frequent words.

Here the top 20 of the most used word in the answers:

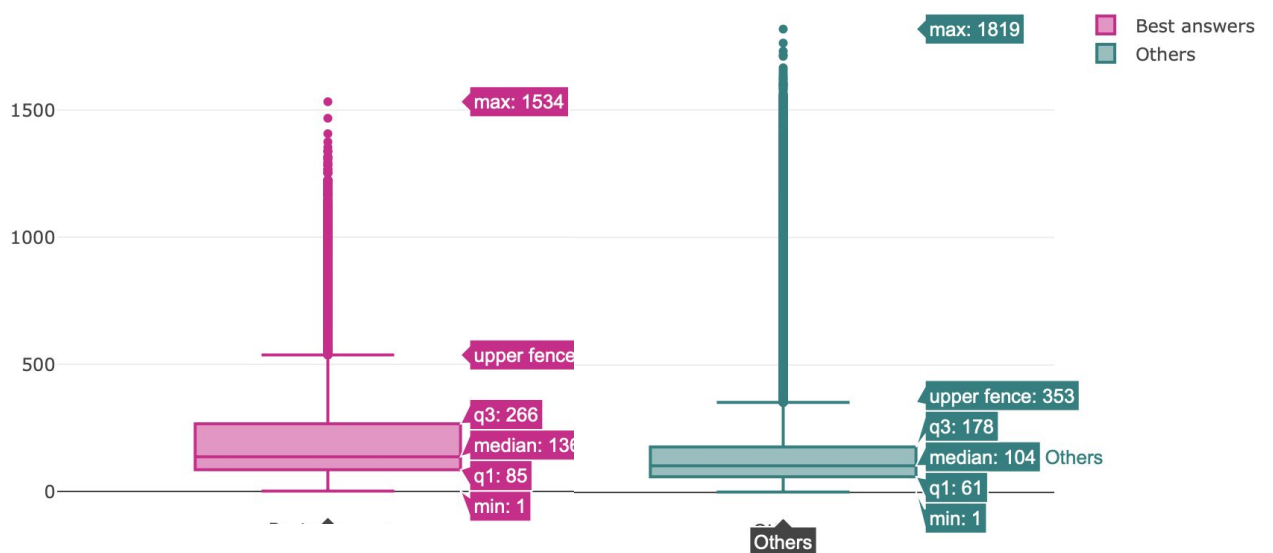
- | | | |
|-----------------|------------|------------|
| - Windows | - Laptop | - Reviews |
| - Configuration | - XP | - Facebook |
| - Account | - Computer | - Click |
| - Problem | - Internet | - Phone |
| - Password | - Drive | - GT |
| - Download | - System | - Ccm |
| - Nbsp | - PC | |

Since we are facing some trouble when we want to display our plot, we build a little function to force colab to authorize plot to show in the active cell.



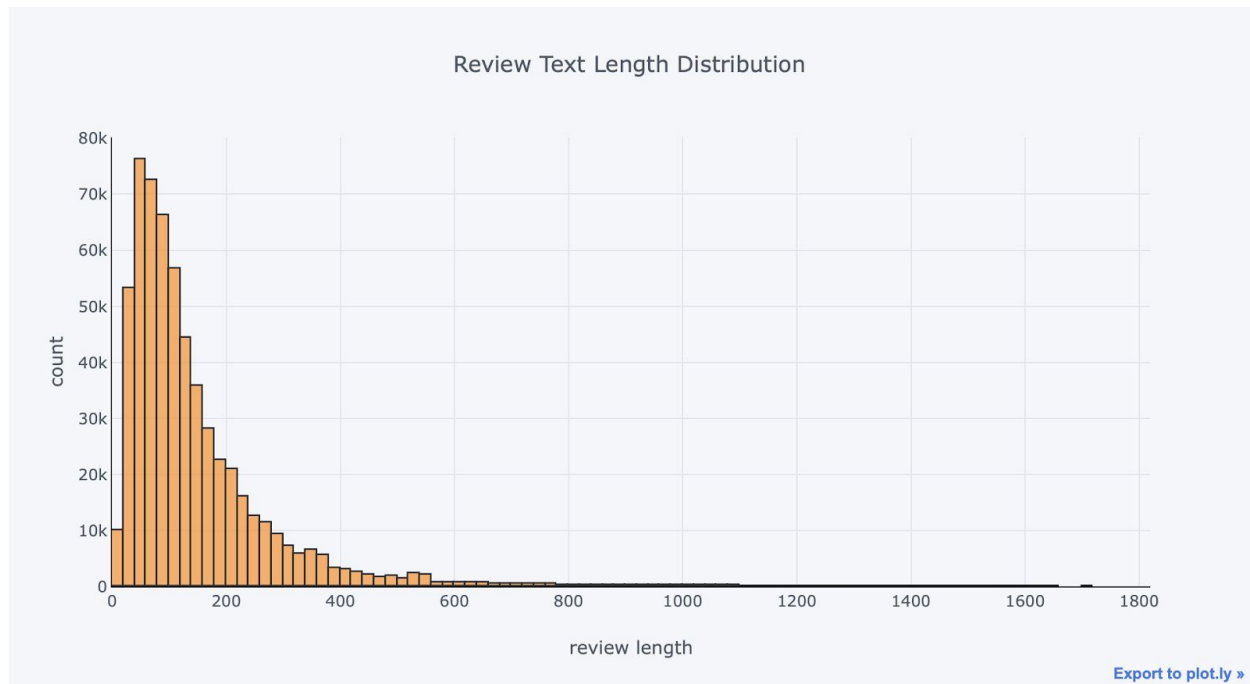
Of course, here we see that the most common words are English and related to IT.

Analysing the distribution of the length of the answers according to if its best answer or not can also be interesting for us. In fact, we can expect, that long and detailed answers are more likely to be labelled as best answers.



The results show that the length does not seem to have a significant impact on the label. In fact, both distributions for the best answers and other answers are really similar.

In order to prevent memory issue when will do the embedding on our message, we check the length of the messages. In fact, a message that is too long can lead to a lack of RAM and can shut down our Google colab session.



Models

Fast text format :

In order to train our model, we have to put our data in the Fast Text format. Fast Text is an open-source, free, lightweight library that allows users to learn text representation and text classifiers. Here, we will not train our model through Fast Text. We will build our model with the library Flair (presented below). Since Flair is using Fast Text corpuses, we need to adopt this format.

Flair is a python library created by Zalando research team. This library draws a very precise state of the art in NLP and permits to reach a good score on regular problems.

- **A powerful NLP library:** Flair allows us to apply the state-of-the-art natural language processing to our text

- **Multilingual:** Flair supports the multilingual text. This is very useful in our case since the data are multilingual
- **A text embedding library:** Flair allows us to use and combine different embeddings. In this project, we used Flair embedding combined with BERT embedding.
- **A Pytorch NLP Framework:** the framework is directly built on Pytorch

Flair provides different type of embedding.


- Classic Word Embeddings
- Character Embeddings
- Byte Pair Embeddings
- Stacked Embeddings
- Flair Embeddings
- Bert Embeddings

In this project, we decided to combine BERT and Flair embedding. We decided to use Document Embeddings instead of Word Embedding. Word Embeddings give us one embedding for individual words unlike document embeddings give us embeddings for an entire text.

What is BERT?

BERT (Bidirectional Encoder Representations from Transformers) makes use of Transformer, an attention mechanism that learns contextual relations between words in a text. It is based on two innovations that we will briefly present below:

- The use of the Transformer architecture which constitutes a recent and efficient alternative to RNNs to achieve a deep bidirectional pre-training, terms to which we will come back later. This is BERT's main innovation.
- The use of two new tasks for pre-training, one at the word level and the other at the sentence level.



BERT's performance defines a new state of the art for many NLP problems and this without requiring any specific adjustment to a particular task! On the difficult problem of the "Answering Question" BERT even surpasses humans!

Document embeddings methods:


Different methods are available for document Embeddings :

- **Pooling:** This method calculates a pooling operation overall word embeddings in a documents. Here we take the *mean* off all the words in the answer. We can also use the operations *min* or *max*.
- **RNN :** Flair also provides a method based on RNN. The default RNN is a GRU-type but we can also use LSTM.

In our project we use a document embedding pooling with Flair Embeddings combined with BERT Embeddings.

Once our document embedding is done we can train the model. We have to choose the parameters of our models :

- **Learning rate:** The amount of change to the model during each step of this search process, or the step size. The learning rate controls how quickly or slowly a neural network model learns a problem. To set up the learning rate we will use the function `find_learning_rate` provided by flair.
- **Mini-batch size:** batch size the number of samples to work through before updating the internal model parameters
- **Hidden size:** number of hidden layer in the LSTM
- **Maximum of epoch:** epoch defines the number times that the learning algorithm will work through the entire training dataset
- **Embeddings:** we can train our model with different embeddings
- **RNN Layers:** number of layers in our Recurrent Neural Network
- **Dropout:** Dropout is a regularization method where input and recurrent connections to LSTM units are probabilistically excluded from activation and weight updates while



training a network. This has the effect of reducing overfitting and improving model performance.

Flair provided a parameter optimizer. Since this method was really highly time-consuming we decided to stick with regular parameters. Of course, the next improvement would be to optimize the parameters of our model.

Experiments

Before ending with our final model we have tried a lot of models. First we tried to build the LSTM model on our own. Since we were facing a few issues of dimension, we have decided to use the model implemented by flair. Using Flair was very convenient even if we had to be precautious about the memory used by our model. In fact, in many cases, we were struggling with the problem conda out of memory issue. That is why we had to reduce the batch size which has lead to an increase in computing time.

Results and conclusion

In this project, we used the Fair framework to build a Textclassifier. The task of our model was to predict for each question of the dataset which question was the best answer (if a best answer was existing). Our model takes in input the title of the question and the text contained in the answer. After cleaning the text, we have decided to use the embeddings multilingual provided by Flair in combination of BERT embedding. We have also decided to use Poolembdding.

Training our model is really time consuming even if we decided to downsample our dataset to ten thousand rows. In a perfect world, we would have taken the whole dataset. Since Google Colab his restraining the RAM used in GPU and also the computing time, training our model was not easy. Nonetheless, after a few epochs we have reached an accuracy of **0.896** on the dev dataset and **0.93** on the test dataset. Training the model on the whole dataset during a longer time will likely lead to better results.



Next steps

In this model, we have only used the answers to the questions, leaving apart the comments. The further step for our model would be to include these comments. Another improvement would also be to compute the similarity between the question and the answers and to build a more traditional machine learning model (using Random Forest, XGBoost ...) using also the other numerical variables as for instance the number of votes. A good idea here would be to combine these two models using, for instance, stacking method.

