# To Be Equal or Not To Be Equal

## That is the question

Dànae Canillas Sánchez - danae.canillas@est.fib.upc.edu
Xavier Rubiés Cullell - xavier.rubies@est.fib.upc.edu

June 2020

**Abstract**

In this project, a study of the dataset provided by Quora in its Kaggle competition has been carried out in order to detect duplicate questions. We will fine-tune pretrained transformer models from the Hugging Face library. We will present the results for different models (BERT, XLNet, DistilBERT, ...) and different hyperparameter combinations that have been used. Finally, we will explore sentence embedding meaning.

https://github.com/XavierRubiesCullell/ProjectQuora

# Contents

# 1 Introduction

## 1.1 Approach

Quora is one of the world's most popular Q&A forums, where questions from many different topics are solved. Being that said, it is perfectly normal that multiple users post (almost) the same question without knowing it already exists on the website. So preventing that will be our task. This tackles some problems, as having repeated questions:

- adds useless load to the website system
- can cause seekers to spend more time finding the best answer to their question
- may make writers feel they need to answer multiple versions of the same question.

As our model will be trained with Quora data, it can work with other examples of data that do not belong to the database with which we are training. For that reason, we would like to see whether we can obtain a generalized model that adapts to all kinds of phrases. We can refer to phrases in this case since we have found that the dataset also contains indirectly formulated questions that can be considered as any kind of sequence.

With the increasing number of online community engagement, there are a lot of sites that work in a similar way, such as other forums like Stack Overflow or enterprises customers sections. Therefore we could conclude that the solution obtained can be used in multiple tasks.

## 1.2 Dataset

The dataset will be taken from the Quora competition at Kaggle:
https://www.kaggle.com/c/quora-question-pairs

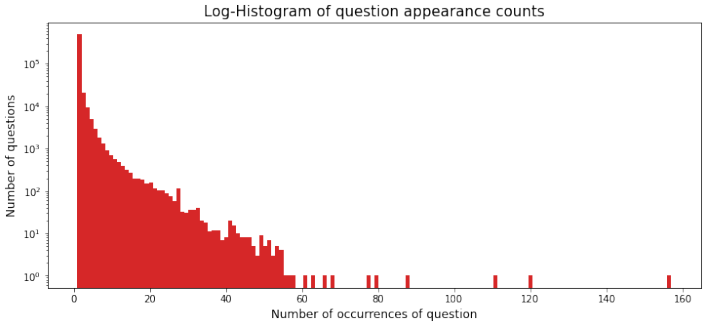
Contains 404290 pairs of questions organized in the form of 6 columns as explained:

- *id*: Row ID
- *qid*1, *qid*2: The unique ID of each question in the pair
- *Question*1, *Question*2: The actual textual contents of the questions.
- *is_duplicate*: Label is 0 for questions which are semantically different and 1 for questions which essentially would have only one answer (duplicate questions).

## 2 Data Analysis

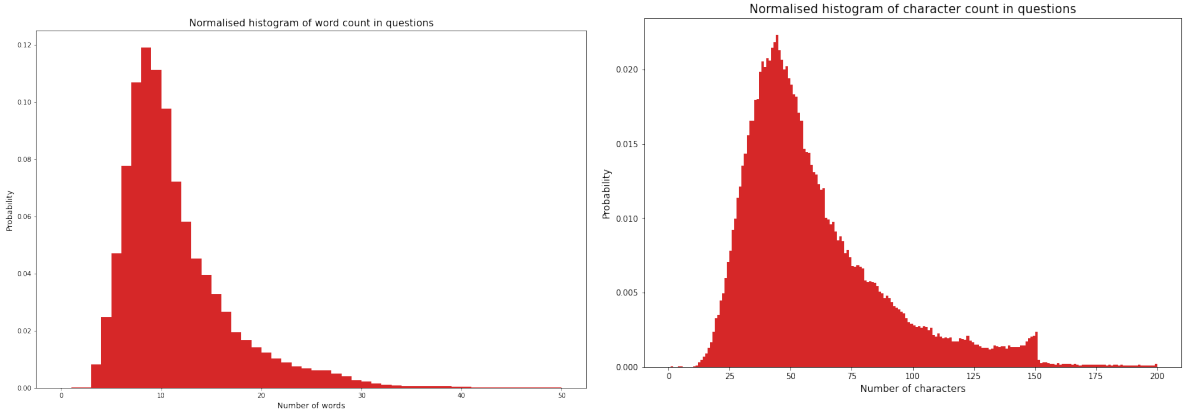
The dataset has 404290 rows, so this is the number of question pairs we have to deal with. We computed how many of them were duplicate (having the target as true), in order to see how the dataset was balanced and whether we needed to balance it. The percentage of duplicate pairs turned out to be 36.92%, which is a value we considered to be moderate, so no class balancing was needed.

As we wanted to generate a table having all the sentences indexed, we needed the know how many unique questions there are. This number is 537933, meaning 270647 of the 808580 total number of questions are repetitions. We wanted to check whether sentences were repeated many times or not, so we plotted their multiplicity:



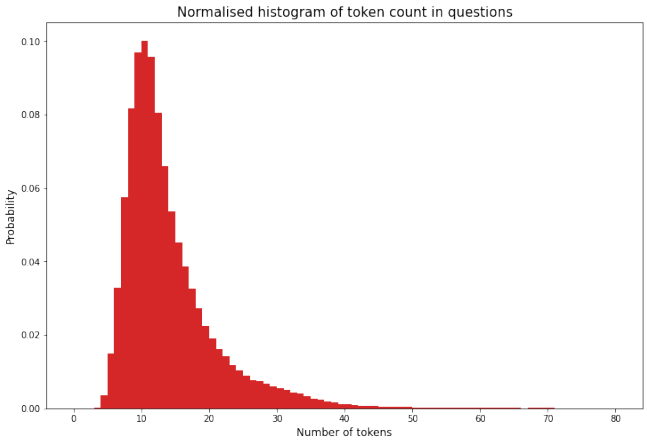

We can see questions follow in some way the Zipf Law, drawing an curve which almost always decreases. The most repeated questions appears almost 160 times.

As we needed to adjust the length of the sentences in order to set a fix value, we needed to see the length of the sentences. First, we plotted the character count length and the word count length. Here we can see them both:



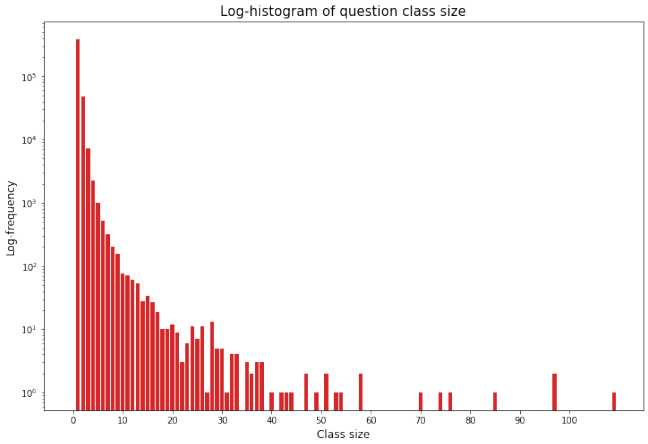

A noteworthy fact that we think is curious is the presence of several questions that only have one word. The maximum number of words found in the sentences is 237. The minimum number of characters present in a question is 0, which corresponds to the one-word question. The maximum number of characters present in a question is 1169. We printed the sentences that set the minimum values we explained and it results there are several questions that are empty.

Then we realised it was more accurate to take into consideration the token count length:




Previously we had commented on the maximum number of words that we have found in the sentences. The maximum number of tokens, logically higher, is 284 tokens.

Apart from the prediction with the transformer, we wanted to study the relationship between the sentences. We will define a sentence class as a set of sentences where every sentence is related to at least one of the other ones. We plotted the class size distribution in order to see whether existed classes with lots of sentences and which was the trend.




We can see the log-histogram follows a negative exponential curve, which means there are many more classes as we lower the size.

We finally plotted a wordcloud, to check which word pairs were the most frequent.


Figure 1: Wordcloud

| word | # of occurrences |
|---|---|
| 'the' | 372316 |
| 'What' | 292887 |
| 'is' | 216832 |
| 'I' | 212601 |
| 'a' | 208748 |
| 'to' | 204709 |
| 'How' | 202001 |
| 'in' | 191594 |
| 'of' | 159425 |
| 'do' | 145554 |

We can observe many typical question constructions in English, such as "What are", "What is", "How do" or "Is it". There are also comparison structures such as "the difference", "the best" or "difference between".

Apart from the analysis of the most frequent words, we have carried out a semantic study of the sentences, obtaining the following statistics:

| | |
|---|---|
| Questions Questions with question marks: | 99.87% |
| Questions with [math] tags: | 0.12% |
| Questions with full stops: | 6.31% |

As seen in the table, most of the questions have the symbol '?', Therefore there is a very small percentage of questions asked indirectly. In addition, as the Quora forum has already commented, it contemplates very diverse thematic questions, for example we find 0.12% of questions related to mathematics. Finally we find 6.31% of sentences containing '.'. The final point in this case does not indicate a percentage of indirect questions but a percentage of questions that are composed, that is, they consist of more than one sentence.

# 3 Input Formatting

In order to train our model we have to adapt the input to the one that is predetermined in Hugging Face pre-trained models. Each row of the dataset contains a pair of questions. The steps to process them are the following:

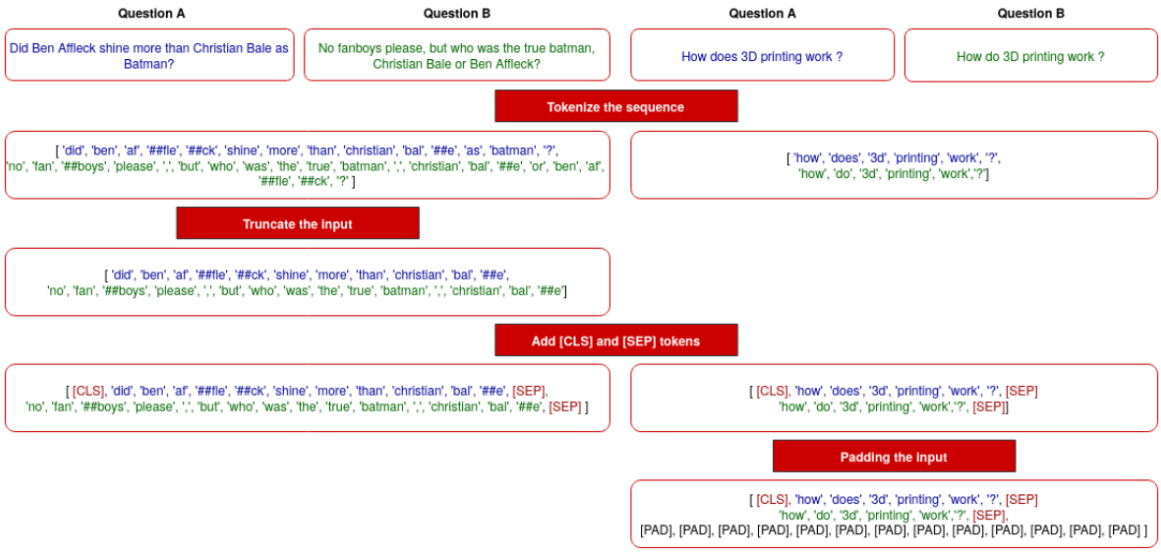



Figure 2: Input construction

1. Each phrase must be tokenized by the tokenizer that each architecture provides us. Tokenization separates the set of characters that make up the sentence into tokens. This division is done based on the appearance frequency of the subwords with which we divide our sentence. Therefore we do not obtain a token for each word as it can be seen in Figure 2.

2. The input of the model must have a fixed length.The length that corresponds to the concatenation of the two sentences has been fixed based on the distribution of the number of words and tokens that has been studied in Section 6.1.2. This can lead to two results:

   - If the concatenation of the two sentences adds has a smaller length than the fixed one, we will use padding to reach that quantity
   - If the union of the two sequences is greater than the defined length, we will have to truncate our input in order to adapt it. This truncation will be balanced according to the length of each phrase, that is, the result will not be more aggressive for one sequence than the other. The function is described in the Appendix 9.1

3. To identify the purpose of our task, it will be necessary to add special tokens to the tokens that encode the questions: [CLS] at the beginning of all the inputs to inform that we are working on a classification task and two [SEP] that indicate the end of each sequence.

The result will define one of the three inputs that will feed the model, the other two inputs are the following:

- Attention mask: This vector will tells us which tokens correspond to the information provided by the sequences and which tokens correspond to the padding.
- Segment tokens: Represents which tokens correspond to the first sequence and which to the second one.

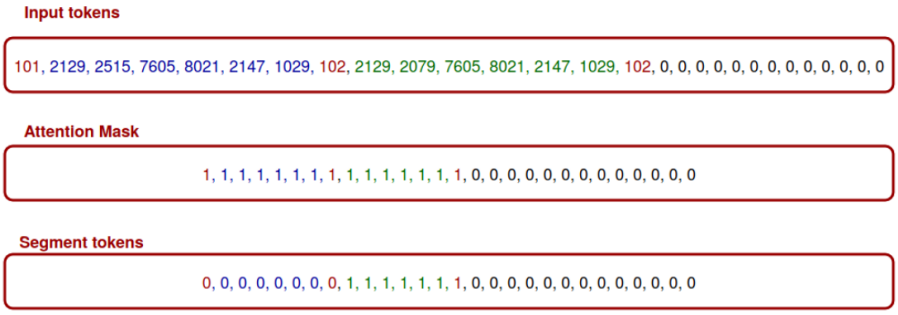



Figure 3: Input set

# 4 Models

## 4.1 Transfer Learning


### 4.1.1 Motivation

When we train a network from scratch, we find mainly two limitations:

- A large amount of data is needed in order to generalize the information learned.
- A large computational resources are required.

From a deep learning perspective our problem can be solved through transfer learning. For this reason we chose to use pre-trained models in order to build accurate solutions in a timesaving way. With transfer learning, instead of starting the learning process from scratch, we start from patterns that have been learned when solving a different problem.

A pre-trained model is a model that was trained on a large benchmark dataset to solve a problem similar to the one that we want to solve. These models have trained weights that encode a lot of vocabulary information and this reduces the training time as we start from a good learning base.

To customize our model and adapt it to the task we needed, we had to do fine-tuning. What we basically did at this stage is model the information learned in the pre-training so that it adapted to our input and thus to our classification task.

In order to achieve our main goal we simply added a new classifier, which was trained from scratch. We did not need to (re)train the entire model since the base network already contained features that are generically useful for the pair of sequences classification task.

#### 4.1.2 Model Selection

There is a huge variety of architectures with which we can achieve our purpose. After carrying out research work, we decided to use transformers, as they are the most widely used today and from which we can obtain the best results.

The BERT architecture has achieved state-of-the-art results on various NLP tasks. In our experiments we will use the last mentioned and other similar architectures for sequence classification, such as DistilBERT or RoBERTa, or quite different, such as XLM. These architectures are from Hugging Face library which provides a large number of pre-trained models that will help us achieve our objective.

- **BERT** is a bi-directional transformer pre-trained over a lot of data that learns language representations that can be used to fine-tune for specific machine learning tasks. While BERT outperformed the NLP state-of-the-art on several challenging tasks, its performance improvement could be attributed to the pre-training tasks of Masked Language Model and Next Structure Prediction along with a lot of data and Google's compute power.
- **DistilBERT** learns a distilled (approximate) version of BERT, retaining 95% performance but using only half the number of parameters. Specifically, it does not use token-type embeddings and retains only half of the layers from BERT. The idea is that once a large neural network has been trained, its full output distributions can be approximated using a smaller network.
- **RoBERTa** is a retraining of BERT with more data and compute power. To improve the training procedure, RoBERTa removes the Next Sentence Prediction (NSP) task from BERT's pre-training and introduces dynamic masking so that the masked token changes during the training epochs.
- **AlBERT** is a Lite BERT for Self-supervised Learning of Language Representations. This architecture uses factorized embedding parameterization since the researchers isolated the size of the hidden layers from the size of vocabulary embeddings computing some projections. It also shares all parameters across layers to prevent them from growing along with the depth of the network.
- **XLNet** is a large bidirectional transformer that tries to improve the training methodology from BERT since it uses larger data and more computational power in order to achieve better prediction metrics. To do so, introduces permutation language modeling, that means that all tokens are predicted but in random order, unless BERT that process the tokens sequentially. This helps the model to learn bidirectional relationships and therefore better handles dependencies and relations between words.
- **XLM** uses a known pre-processing technique (BPE) and a dual-language training mechanism with BERT in order to learn relations between words in different languages. Additionally, XLM is trained with a Translation Language Modeling (TLM) objective in an attempt to force the model to learn similar representations for different languages.
- **XLM-Roberta** offers over the original model (XLM) a significant increase in the amount of data that was used to train. The 'RoBERTa' part comes from the fact that its training routine is the same as the monolingual RoBERTa model, specifically, that the sole training objective is the Masked Language Model as well as XLM does.

We had also considered using Bart, a BERT model improvement, but the internal implementation of the model was different from the others since the input model was completely different compared to the others. If the vocabulary of the questions in Quora had been French, we would have chosen Camembert and Flaubert, two architectures that are from BERT's family as well and were trained using French data.

## 4.2 Experiments

### 4.2.1 Evaluation method

To evaluate our models we have calculated the accuracy that indicates the proportion of correct guesses in the prediction, as well as the precision, the recall and the F1 score. For the comparison between models we have focused on the loss of the model and the accuracy of both training and validation. Each model has been trained in a set of epochs, in each of them we have calculated the aforementioned metrics, below we can show an example of the evolution:

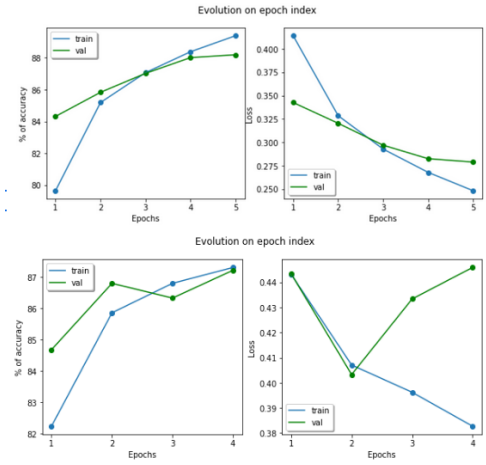



Figure 4: Examples of metrics evolution on epoch index

We ensured that the model did not overfit, in other words, we checked the training accuracy value was not far from the validation. We also studied the evolution of the loss in each model, considering always the epoch with the minimum validation loss value.

In the Figure 4 we find two examples of the evolution that we discussed before. The curves are quite different, in the first one we observe a positive evolution of the accuracy in both values since they grow following the same trend, the same happens with the loss that decreases in every epoch. In the second plot there is a good evolution of the accuracy but the error in validation increases considerably, in this case we would take the metric values from the second epoch.

### 4.2.2 Hyperparameters

Next, we will describe the set of hyperparameters used in our models along with a description of their purpose and the improvements or drawbacks their values can provide.

- Model type: Model architecture from the ones listed in Section 4.1.2
- Max_seq_length: Models need the tokenized sentence pair to have a fixed length to be inputted. The higher the sentence, the more information it has. However, as padding is needed for shorter sentences, it might only make the model last longer.
- Batch_size: Too low a value makes the model not perform well and have a long execution time. On the other side, too high a value also decreases the performance and also can exceed RAM capacity.
- Epochs: A value which is too low does not allow the model to improve enough. A value which is too high results in overfit, which means the model does not generalise well.
- Learning rate: It sets the speed at which the model learns.
- Adam epsilon: It is not a hyperparameter that we had modified, since it is an epsilon that avoids a division between 0.
- Warmup proportion: It sets the number of warmup steps. Learning rate is better decreased at the beginning.
- Max_norm: It sets the value at which gradients are clipped in order to avoid its explosion.
- Schedule method: We tried different learning rate scalings systems:

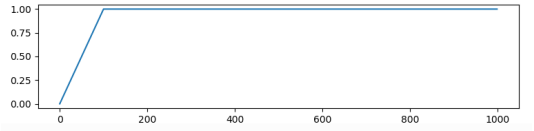



Figure 5: Constant Scheduler

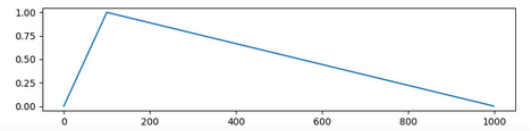



Figure 6: Linear Scheduler

- Accum_steps: Regulates the amount of memory used depending on the batch. It helped us in the beginning to make the models work.
- Criterion: Criterion used to calculate the loss: mean vs.sum

#### 4.2.3 Experiments & Results

The Hyperparameters Study shows the different experiments with the models, varying the hyperparameters in order to optimize their results. The optimisation as mentioned in the Section 4.2.1 has been carried out based on the results of the accuracy and the loss.

Our study has focused on the comparison of models taking BERT as a reference. Therefore we would like to study the behavior of the models that have been created based on its architecture and also with others that use different model structure.

As can be seen in the link table, looking at the results we could say that the majority of tested models have predicted us with a very high validation accuracy, around 89%. The models which obtained better results were XLNet and BERT, obtaining a validation accuracy of 90.62% and 90.44%, respectively. These are some conclusions we extracted from hyperparameter optimization:

- As explained in the Section 2, we had to define a fixed input size model. Knowing that the average measure of each sentence was around 35-40 tokens, we tried 60, 70, 80 and 90 as input measures to the model (union of the two sentences by adding the specific tokens of the classification task). The results were similar, no differences were perceived at the performance level. We set 70 as input size.
- The best batch sizes were 64 and 128. Increasing the size of this hyperparameter we decreased the accuracy and proposing small values increased the training time and the results were not satisfactory.
- The BERT authors do not recommend using more than 4 epochs since the models tend to overfit. We took it into account and we tested values between 2 and 4.
- BERT paper authors recommended to use learning rate values of 3e-4, 1e-4, 5e-5, 3e-5. This hyperparameter is very variable for each model but in general we obtained better results using small values, around 2e-5.
- The warmup ratios that achieved the best results were 0.01 and 0.001. The scheduler method that worked best for all models was constant.
- Finally, looking at the training and validation time the fastest model by far was DistilBERT. This is the reason why we did a lot of executions with it, apart from the fact it has a structure as BERT.

# 5 Conclusions

In this project we explored the world of pre-trained models and their applications. We had never used this type of architecture before, but once tested we have been able to verify that they are very effective since we obtained very gratifying prediction results.

We had always heard that BERT has been one of the state-of-art models in the NLP field of study, so that is why we decided to carry out a comparative study with similar architectures that had also been pre-trained. Compared to the trainings that we have done this course, the number of epochs has been greatly reduced. As it has been mentioned above, with only 2 or 3 epochs we have obtained an accuracy greater than 90%.

Referring to the hyperparameters we can conclude that the two that influenced the most were batch size and the learning rate. The latter must be adjusted together with the number of epochs to train, the higher the one, the lower the other one. We ended up using less epochs and a lower learning rate than BERT authors recommended. We think it is due to we have a large dataset, so the model already has a long path to learn. The evolution of the learning rate was determined by the scheduler (the one that has worked best has been the constant) and by the number of warming up steps.

The development of this project was not easy at all. First of all, we needed to adapt the raw data to the input imposed by the Hugging Face library models. This led us to tokenize the phrases, balancing their sizes and adding special tokens for task recognition. Regarding the executions of the models, at the beginning we had to face various memory problems since we were not optimally loading the data.

Once the task proposed in the Kaggle competition was solved, we thought about exploiting the data in order to solve other applications: we moved away from the prediction/classification task in order to focus on the inference of the data. We developed three applications that will be explained next.

# 6 Extra work: Different approaches

Apart from the main task of classifying the questions, we wanted to go one step further in order to explode the information contained in the dataset. As our work was focused on the comparison of architectures taking BERT as the main reference, we have worked in the following applications using this model. Therefore, tokenization and predictions have been done with the mentioned architecture. Next, we present three different use cases:

## 6.1 Question classes

### 6.1.1 Prediction consistency

As we said before, we defined a class of sentences as a set of sentences where every one of them is related to at least one of the other ones. Sentence classes can refer to either the true values (target) or the predictions of the model. Our goal in this section was to check whether the model said every pair of sentences in a prediction class is similar.
Firstly, we loaded the model that performed the best in our experiments. We executed it once to predict the labels and to build the classes. We selected some classes of different sizes and specifically the biggest one. Then, we inputted all pairs in every class to the model and computed the accuracy of the prediction.
We thought we would get an accuracy far from 100%, but results were extremely poor. The accuracy was 2.70%. Our main hypothesis is the fact that errors are spread. Maybe the model predicts two samples are similar with a probability of 52%, but in fact they are not. Their classes will be joint, but their elements are far from being similar, which is what the model predicts.

### 6.1.2 Class visualisation

In the visualisations that can be found in plots we show the embeddings (tokenizations) of the questions of the dataset doing a reduction of dimensionality using two different methods: PCA and TSNE. These method helped us to reduce to two or three dimensions (depending on the plot) the encoding of the sentence in order to be represented in the space. In this plot we do not represent all the questions in the dataset: we have chosen a subset that contains questions grouped by classes (that have more than two questions).

PCA is a method that is based on reducing the dimensionality based on the dispersion of the data considering the directions of maximum variability. On the other hand TSNE is a method that reduces the size of the inputs guided by distances. Below we show two visualizations of the dispersion of the questions in two-dimensional space:

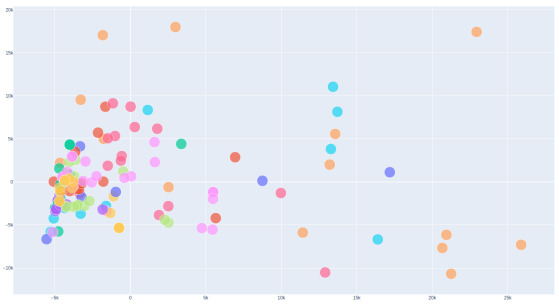


Figure 7: PCA scatter plot

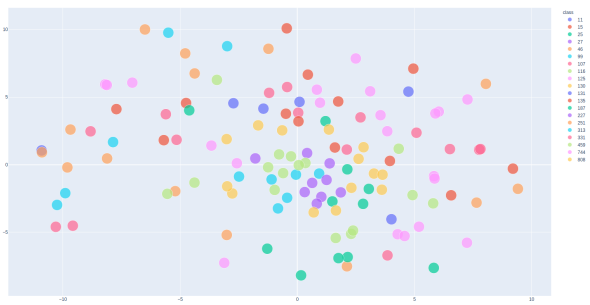


Figure 8: TSNE scatter plot

We can see that the dispersion of the points of the plot computed with TSNE is greater than PCA. We believe that this is because it is a distance based method. When we predict whether two sentences are duplicates, we compare their meaning and not on the order of the words in each sentence. When we calculate the distance between sentences we do it token to token, therefore we compare the positions of the words of each question.

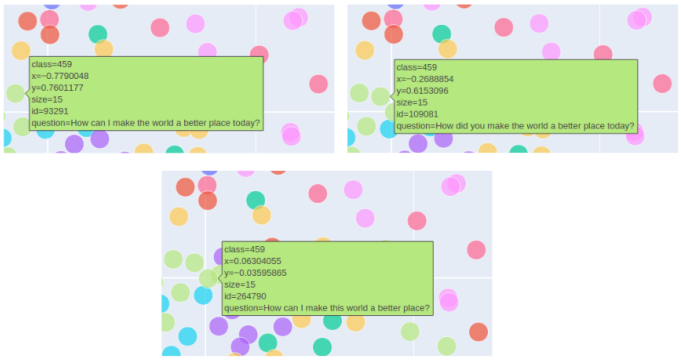



Figure 9: Nearby questions belonging to the same class

Theoretically, the points with the same color belong to the same class, but the colour palette is not very wide and some are repeated. To solve it, we have a hovering that shows the sentence and which class it belongs to, as can be seen in Figure 9.

## 6.2 Most similar sentence

As we said, one of the motivations of this project is the fact that we could use the model we developed every time a user intends to post a question on the website. If a question coincides significantly with another one, it would not be posted and maybe the user would be given its best answer or would be redirected to the question page.

We tried to do a home-made approach to that, by computing a distance between the embeddings of a pair of sentences. This could be used to find the most similar question(s) to that one.

We designed a first approach, consisting in computing the euclidean distance between the sentence and all the reference ones. Before the computation, we cut embeddings to 34 tokens, as it is the average length of a sentence in the model input (the set of the two sentences occupies 70 tokens, 3 of them being special tokens).

Here we can see some examples:

**Sentence:** Who was more voted in the presidential elections, Hillary Clinton or Donald Trump?
**Result:** Who will be the next president of USA: Hillary Clinton or Donald Trump?

**Sentence:** Is Harry Potter in love with Hermione?
**Result:** What does IMO mean in a text message?

**Sentence:** Which is the best rock band in history?
**Result:** What are the best video games to play?

The system works well in some specific cases, but it has two important drawbacks:

- Strict order is a key feature
- Distance in a specific dimension does not have any meaning. There are words with similar embedding which are not related in any sense. For example, "no" is tokenised as '2053' and "what" as '2054'. Another example is the case of "thoughts" and "Harry", 4301 and 4302 respectively.

This motivated us to design a method that:

- Does not consider the order: The sentences "Do all the people in USA have guns?" and "Do all people in USA have guns?" may have quite a high distance due to the shift because of the "the", while they are practically the same.
- Considers a distance which is binary, where we only evaluate whether two tokens are the same or not.

We developed a function where we only take into consideration the words in every sentence and compare their multiplicity. Results improved quite significantly. Here we can see the examples we saw before:

**Sentence:** Who was more voted in the presidential elections, Hillary Clinton or Donald Trump?
**Result:** Who will win the US presidential elections 2016: Hillary Clinton or Donald Trump?
**Similarity:** 66.67%

**Sentence:** Is Harry Potter in love with Hermione?
**Result:** Why don't Hermione fall in love with Harry?
**Similarity:** 61.54%

**Sentence:** Which is the best rock band in history?
**Result:** Which is the best alternative rock band?
**Similarity:** 77.78%

However:

**Sentence:** Are aliens green or grey?
**Result:** Are my eyes hazel or green?
**Similarity:** 57.14%

We denoted it does not work well when few words from the introduced sentence are significant (so most of the words in the sentence are articles, auxiliary verbs, ..., meaningless words in general). A good feature about this implementation compared to the previous one is that we can set a maximum similarity, which gives us a sense of similarity magnitude.

# 7 Next Steps

- Before considering transformers, we already saw the use of architectures that included attention was positive, since using this method in oral and written treatment have been rewarding results in Deep Learning. One possible way was to use a Siamese LSTM network where we process each pair of questions independently and then apply the output to a classifier to predict whether these two sentences are indeed duplicates.
- Train from scratch some architectures we used. We would be able to see how important is the pretraining given when using these models.
- Train and use a transformer designed by us. We would be able to see whether those exact architectures perform better than other options. We could also try to improve their performances by modifying their architecture
- Improve most similar sentence
  - Delete tokens from common meaningless words, such as articles, some conjunctions, some prepositions, some adverbs, auxiliary verbs, ... We think our model should heavily rely on nouns and meaningful verbs.
  - Try DTW, which does not restrict order but takes it into consideration, which we still think might be interesting. We think it as "Was Michael Jordan better than Kobe Bryant?" is not the same as "Was Kobe Bryant better than Michael Jordan?"

# 8 References


Comparison of pre-trained models

- https://www.kdnuggets.com/2019/09/bert-roberta-distilbert-xlnet-one-use.html
- https://ai.googleblog.com/2019/12/albert-lite-bert-for-self-supervised.html

BERT FineTuning on Quora Question Pairs

- https://medium.com/@drcjudelhi/bert-fine-tuning-on-quora-question-pairs-b48277787285

Hugging Face Repository

- https://huggingface.co/transformers/pretrained_models.html
- https://github.com/huggingface/transformers

Identifying Semantically Duplicate Questions Using Data Science Approach: A Quora Case Study

- https://arxiv.org/pdf/2004.11694.pdf

Effective Approaches to Attention-based Neural Machine Translation

- https://arxiv.org/pdf/1508.04025.pdf

Quora Question Duplication

- https://pdfs.semanticscholar.org/d2e0/0f90aece22568ff25a39094b78bfa95087ba.pdf?_ga=2.71759827.1110382809.1592576740-152019488.1586561150

Natural Language Understanding within the context of Question Matching:

- https://cs.brown.edu/research/pubs/theses/ugrad/2019/li.michael.pdf

Natural Language Understanding with the Quora Question Pairs Dataset:

- https://arxiv.org/pdf/1907.01041.pdf

Detecting Duplicate Questions with Deep Learning:

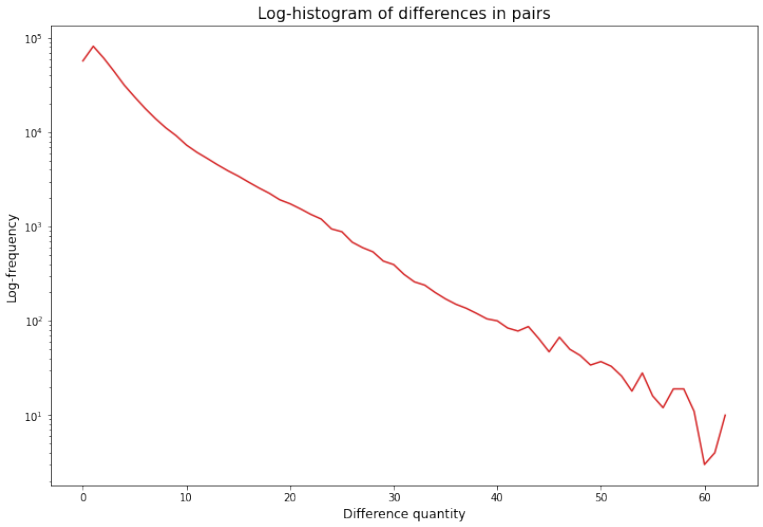
- https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1174/reports/2748045.pdf

# 9 Appendix

## 9.1 Truncate function motivation

As we stated before, we needed to truncate sentence pairs in order to always have the same length. Padding brings no problem, as it is a simple zero filling. However, when we truncate sentences, we are losing information, so it is an operation that needs to be studied in order to do it properly. Cutting to a determined size both sentences can be disproportionate, as it might happen that one sentence loses a lot of information, will the other remains (almost) the same. The same happens if we truncate the same number of tokens in every sentence.

In order to see whether we should develop a system considering both sentence lengths we plotted the log-histogram of the difference between sentences (until there was no more than 10 pairs per difference value).




We see that differences follow a log-distribution. The most frequent difference is 1, followed by 2 and 0. However, a lot of pairs have a significant difference between their length. This confirmed that we needed to develope a truncate function, depending on the length of both sentences.
So, if the length of the sentence A is $a$, the one from B is $b$ and we need to have $m$ tokens in total, we will take $m \cdot \frac{a}{a+b}$ from sentence A and $m \cdot \frac{b}{a+b}$ from sentence B.