Quora Questions Pairs

Federica Baldi, Daniele Cioffo, Edoardo Fazzari
MSc in Artificial Intelligence and Data Engineering

TABLE OF CONTENTS

1. Introduction 1
1.1. Dataset 1
1.1.1. Splitting and New Sets Analysis 3
2. Data Preparation 5
3. From-scratch architecture 6
3.1. Basic Experiment 6
3.2. Class Weight for unbalanced classes 7
3.3. Data Augmentation to balance classes 8
3.4. Total Data Augmentation 8
4. Siamese Neural Network 10
4.1. GloVe embeddings 11
4.1.1. LSTM with 50 units 11
4.1.2. Bidirectional LSTM 12
4.1.3. Data Augmentation 14
4.2. Word2vec embeddings 14
4.2.1. LSTM with 50 units 15
4.2.2. Bidirectional LSTM 16
5. Transfer Learning and Feature Reuse 18
5.1. Way of Extracting the Features 18
5.2. Experiments Typologies 18
5.3. Doc2Vec 18
5.3.1. The Deep Neural Network Used 19
5.3.2. Experiments Results 19
5.4. Paraphrase-MiniLM-L12-v2 20

5.4.1. The Deep Neural Network Used....21

5.4.2. Experiments Results....21

5.5. All-mpnet-base-v2....22

5.5.1. The Deep Neural Network Used....23

5.5.2. Experiments Results....23

5.6. All-distilroberta-v1....24

5.6.1. The Deep Neural Networks Used....24

5.6.2. Experiments Results....26

6. Ensemble....29

6.1. Why we need an Ensemble....29

6.2. The Ensemble and its Results....29

7. Error Analysis....31

7.1. “Duplicates” class....31

7.2. “Not duplicates” class....32

8. Conclusions....34

9. References....35

10. Appendix A: Creating DistilRoBERTa ensemble....36

10.1. Genetic Algorithm Components....36

10.2. Genetic Algorithm Results....36

11. Appendix B: Creating the final ensemble....37

11.1. Genetic Algorithm Components....37

11.2. Genetic Algorithm Results....38

Quora Question Pairs

Federica Baldi, Daniele Cioffo, Edoardo Fazzari

MSc in Artificial Intelligence and Data Engineering

1. INTRODUCTION

Quora [1] is a question-and-answer platform where users can post questions and answers on any topic. Quora groups questions and answers by topics and allows users to vote or add comments. Also, users can collaborate by editing questions or suggesting changes to answers provided by other users.

Since new questions are posted every day, there is a high probability that some of them are literally the same or have similar intent, which creates unnecessary duplicates in both questions and answers. This can be data-heavy and can result in a degradation of the user experience. Indeed, multiple questions with the same intent can cause seekers to spend more time finding the best answer to their question and make writers feel they need to answer multiple versions of the same question. As a result, the answers get fragmented across different versions of the same question.

For example, the questions “What’s the fastest way to get from Los Angeles to New York?” and “How do I get from Los Angeles to New York in the least amount of time?” are the same question but asked differently. In order to identify duplicate questions and perhaps merge them into a single post, it is crucial to be able to understand their semantics and compute their similarity. Natural Language Processing (NLP) technologies can help solve this challenging task.

In this project, after a brief analysis of the available data, we will attempt to solve the task of assessing the similarity of question pairs using three different approaches. First, we will try to build an LSTM architecture on an embedding layer trained from scratch. Next, we will build a Siamese neural network inspired by the literature, leveraging pre-trained word embeddings. Later, we will take a transfer learning approach by using features extracted from pre-trained models to train a deep neural network. At the end, we will analyse the errors made by our best models to evaluate whether or not to resort to an ensemble model.

1.1. Dataset

The Quora Question Pairs dataset consists of a training set of 404,290 question pairs, and a test set of 2,345,795 question pairs, and is provided as part of a Kaggle competition [2].

Since the test provided does not contain labels for any question pair, the only measure of performance that can be obtained with this test set is accuracy (via online submission to Kaggle). We therefore felt it better to construct our own test set from the training set provided, since this would allow us to obtain performance metrics other than accuracy and perform further error analysis of our prediction models. Thus, our data exploration only considered this training set of 404,290 question pairs. More information about how we split this set into training, validation, and test sets is provided in subsection 1.1.1.

Each sample point has the following fields:

- *id*: unique ID of each pair
- *qid1*: ID of first question
- *qid2*: ID of second question

- *question1*: text of first question
- *question2*: text of second question
- *is_duplicate*: are the questions duplicates of each other (0 indicates not duplicate, 1 indicates duplicate)

Example of sample points are shown in Table 1.

id	qid1	qid2	question1	question2	is_duplicate
389764	148299	66348	How do I get started to create a new website?	How do create a website?	1
207750	21170	311536	How do I prepare for the Cisco 500-201 exam?	How can I get prepared for the Cisco 700-295 exam?	0
363780	493801	493802	What is the best brunch in Orange Country, CA?	Where can I get the best pizza in Orange Country, CA?	0

Table 1. Sample points from the training set provided by the Kaggle competition

Of the 404,290 question pairs, 255,027 (63.08%) have a negative (0) label, and 149,263 (36.92%) have a positive (1) label, making our dataset unbalanced.

While every question pair is unique, every question within the question pairs is not. Across all question pairs, there are 537,944 unique questions. Of this, 111,780 questions (20.78%) occur across multiple pairs, with one of them appearing 157 times. Figure 1 shows the number of times a question appears against the number of questions for that many occurrences.

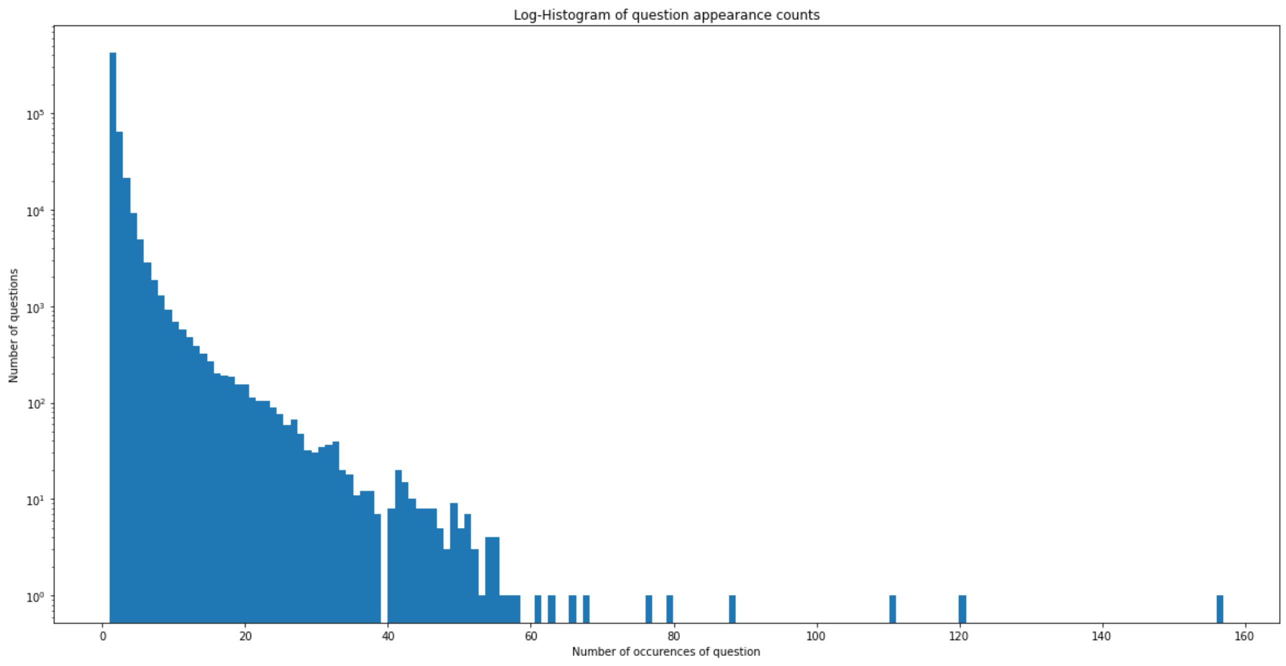


Figure 1. Log-Histogram of question appearance counts

Question pairs are not always meaningful, in some cases *question1* and/or *question2* is/are set to *NaN*. This happens three times in the training set provided by Kaggle. To solve this problem, the corresponding rows were removed from the csv.

1.1.1. Splitting and New Sets Analysis

As mentioned in the previous section (1.1), the original training set provided by the Kaggle competition was split into three parts: 80% (323,431) for training set, 10% (40,428) for validation set, and 10% (40,428) for test-set. The splitting was completely random: the original dataset was first shuffled and then split in the three parts. Thus, an analysis on the sets obtained was done.

The new *training set* contains 323,431 question pairs, 204,214 (63.14%) have a negative (0) label, and 119,217 (36.86%) have a positive (1) label, maintaining the same distribution of the original training set. Across all question pairs, there are 450,352 unique questions. Of this, 85,762 questions (19.04%) occur across multiple pairs, with one of them appearing 129 times. Figure 2 shows the number of times a question appears against the number of questions for that many occurrences.

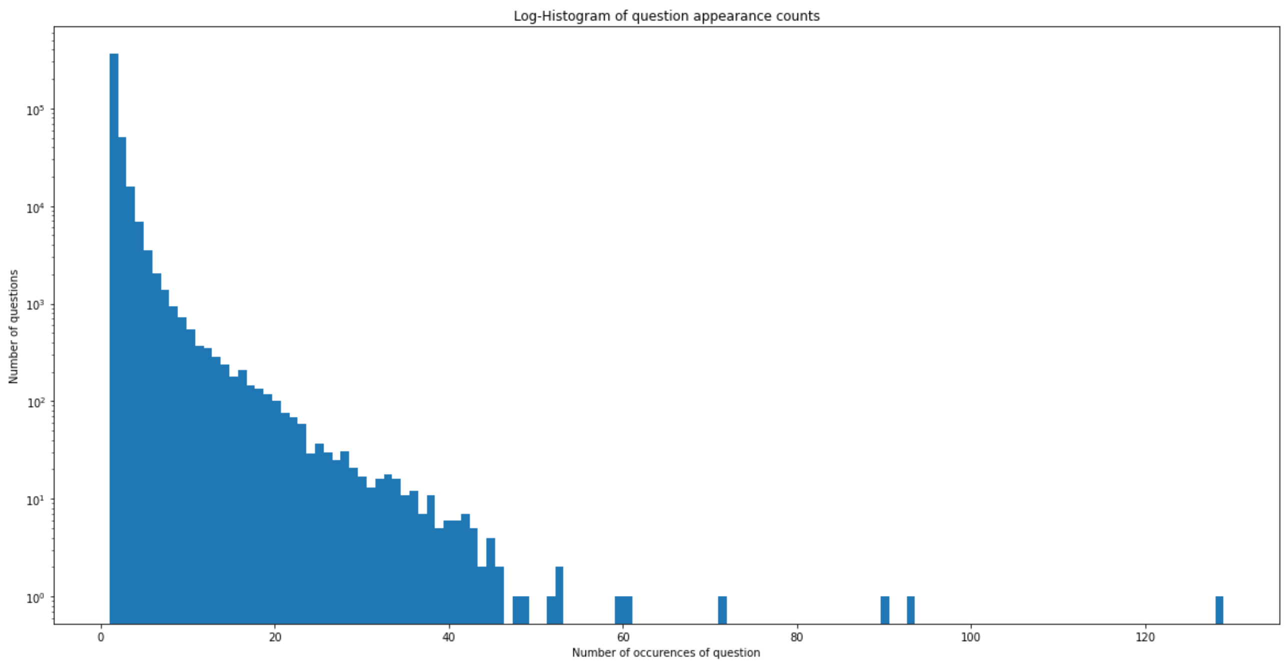


Figure 2. Log-Histogram of question appearance counts for new training set

The *validation set* contains 40,428 question pairs, 25,470 (63%) have a negative (0) label, and 14,958 (37%) have a positive (1) label, maintaining the same distribution of the original training sets. Across all question pairs, there are 73,085 unique questions. Of this, 5,448 questions (7.45%) occur across multiple pairs, with one of them appearing 16 times. Figure 3 shows the number of times a question appears against the number of questions for that many occurrences.

The *test set* contains 40,428 question pairs, 25,348 (62.7%) have a negative (0) label, and 15,080 (37.3%) have a positive (1) label, maintaining the same distribution of the original training sets. Across all question pairs, there are 73,132 unique questions. Of this, 5,367 questions (7.34%) occur across multiple pairs, with one of them appearing 16 times. Figure 4 shows the number of times a question appears against the number of questions for that many occurrences.

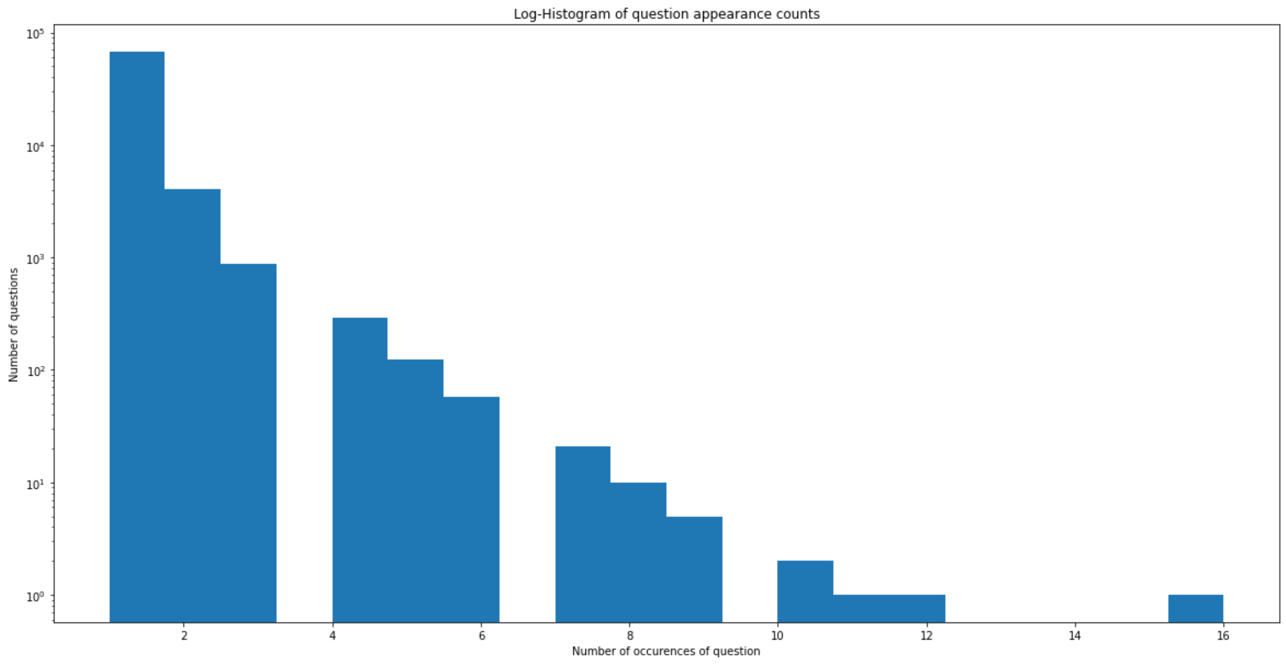


Figure 3. Log-Histogram of question appearance counts for validation set

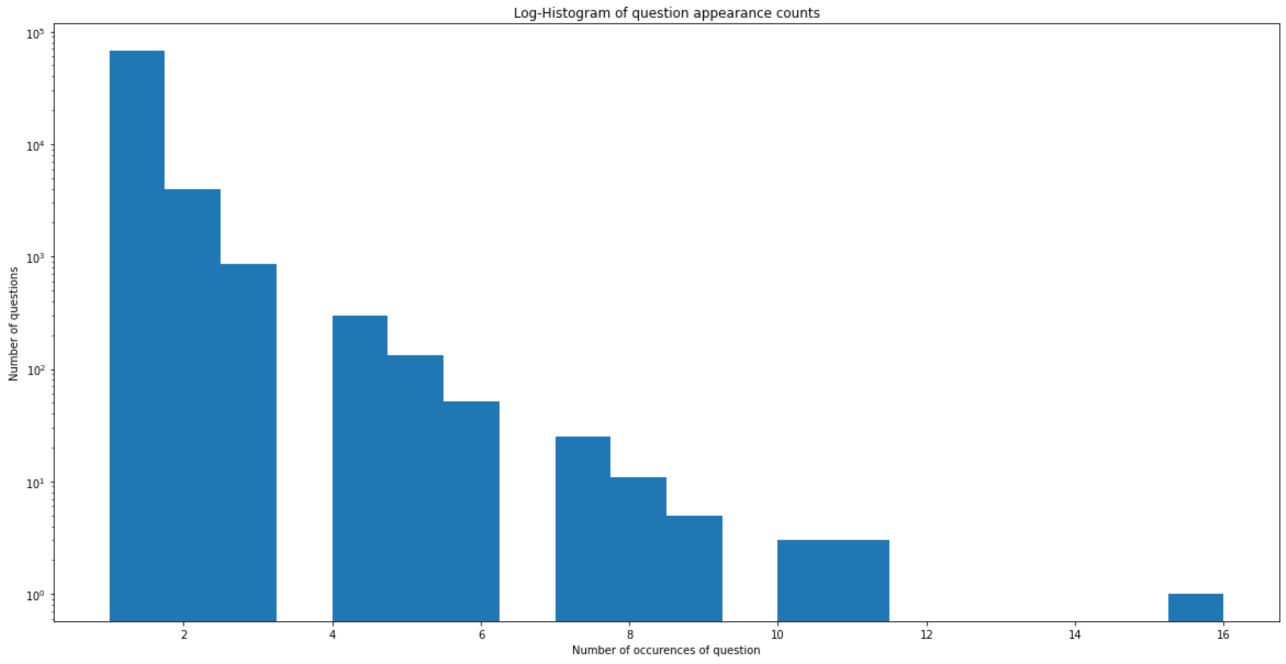
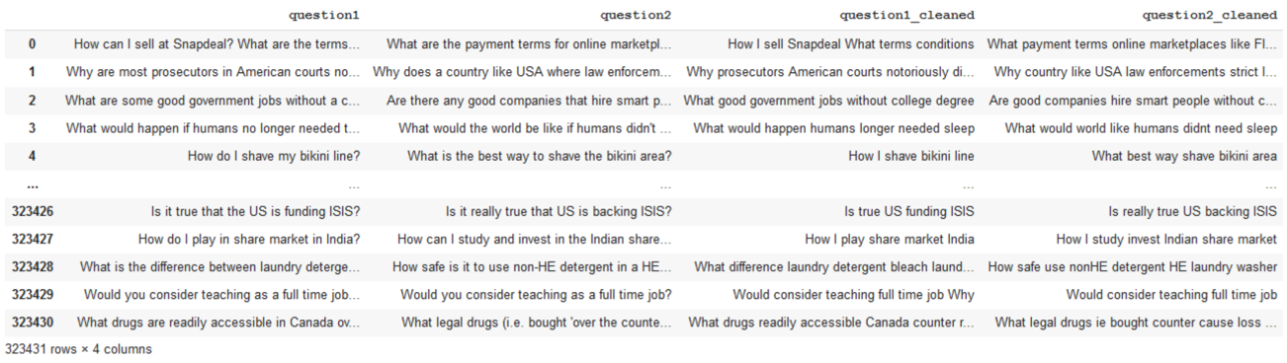


Figure 4. Log-Histogram of question appearance counts for test set

2. DATA PREPARATION

Before giving them as input to the networks built in the next Chapters 3 and 4, data were specially prepared. In particular, first a cleaning procedure was performed to remove punctuation, non-ASCII characters, stop words, etc. from the questions. The results thus obtained are shown in Figure 5.



	question1	question2	question1_cleaned	question2_cleaned
0	How can I sell at Snapdeal? What are the terms...	What are the payment terms for online marketpl...	How I sell Snapdeal What terms conditions	What payment terms online marketplaces like Fl...
1	Why are most prosecutors in American courts no...	Why does a country like USA where law enforcem...	Why prosecutors American courts notoriously di...	Why country like USA law enforcements strict l...
2	What are some good government jobs without a c...	Are there any good companies that hire smart p...	What good government jobs without college degree	Are good companies hire smart people without c...
3	What would happen if humans no longer needed t...	What would the world be like if humans didn't ...	What would happen humans longer needed sleep	What would world like humans didnt need sleep
4	How do I shave my bikini line?	What is the best way to shave the bikini area?	How I shave bikini line	What best way shave bikini area
...	...	...	...	...
323426	Is it true that the US is funding ISIS?	Is it really true that US is backing ISIS?	Is true US funding ISIS	Is really true US backing ISIS
323427	How do I play in share market in India?	How can I study and invest in the Indian share...	How I play share market India	How I study invest Indian share market
323428	What is the difference between laundry deterge...	How safe is it to use non-HE detergent in a HE...	What difference laundry detergent bleach laund...	How safe use nonHE detergent HE laundry washer
323429	Would you consider teaching as a full time job...	Would you consider teaching as a full time job?	Would consider teaching full time job Why	Would consider teaching full time job
323430	What drugs are readily accessible in Canada ov...	What legal drugs (i.e. bought 'over the counte...	What drugs readily accessible Canada counter r...	What legal drugs ie bought counter cause loss ...

323431 rows × 4 columns

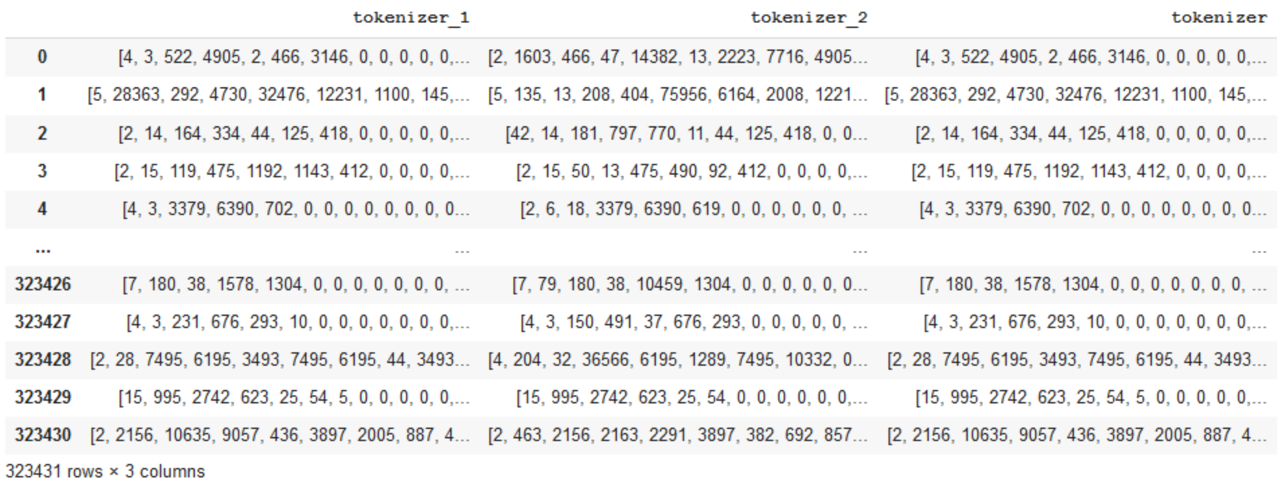
Figure 5. Cleaning procedure for questions

Next, we stacked all "cleaned" questions from the training set together, so that the vocabulary constructed by the Tokenizer contained all the words from both columns. At this point, it was possible to tokenize all questions in the training, validation, test testing sets by converting individual words into numbers.

Some questions are very long; however we think that typically the most semantically important information is contained between the first few words and not the last. For this reason, we decided to consider only the first 25 words for each question. For shorter sentences a zero-padding was added after the sequence of tokens, while longer sentences were truncated by removing the last tokens from the sequence.

As for the Siamese neural network that we will discuss in Chapter 4, the pre-processing ends here. On the other hand, regarding the architecture from scratch discussed in Chapter 3, an additional step is needed. In fact, while the Siamese neural network will take in input the two tokenized questions separately, the LSTM network will expect a single input.

To this end, as shown in Figure 6, the two tokenized questions of size 25 are finally concatenated to form the *tokenizer* field of size 50.



	tokenizer_1	tokenizer_2	tokenizer
0	[4, 3, 522, 4905, 2, 466, 3146, 0, 0, 0, 0, 0,...	[2, 1603, 466, 47, 14382, 13, 2223, 7716, 4905...	[4, 3, 522, 4905, 2, 466, 3146, 0, 0, 0, 0, 0,...
1	[5, 28363, 292, 4730, 32476, 12231, 1100, 145,...	[5, 135, 13, 208, 404, 75956, 6164, 2008, 1221...	[5, 28363, 292, 4730, 32476, 12231, 1100, 145,...
2	[2, 14, 164, 334, 44, 125, 418, 0, 0, 0, 0, 0,...	[42, 14, 181, 797, 770, 11, 44, 125, 418, 0, 0...	[2, 14, 164, 334, 44, 125, 418, 0, 0, 0, 0, 0,...
3	[2, 15, 119, 475, 1192, 1143, 412, 0, 0, 0, 0,...	[2, 15, 50, 13, 475, 490, 92, 412, 0, 0, 0, 0,...	[2, 15, 119, 475, 1192, 1143, 412, 0, 0, 0, 0,...
4	[4, 3, 3379, 6390, 702, 0, 0, 0, 0, 0, 0, 0, 0...	[2, 6, 18, 3379, 6390, 619, 0, 0, 0, 0, 0, 0, ...	[4, 3, 3379, 6390, 702, 0, 0, 0, 0, 0, 0, 0, 0...
...	...	...	...
323426	[7, 180, 38, 1578, 1304, 0, 0, 0, 0, 0, 0, 0, ...	[7, 79, 180, 38, 10459, 1304, 0, 0, 0, 0, 0, 0...	[7, 180, 38, 1578, 1304, 0, 0, 0, 0, 0, 0, 0, ...
323427	[4, 3, 231, 676, 293, 10, 0, 0, 0, 0, 0, 0, 0,...	[4, 3, 150, 491, 37, 676, 293, 0, 0, 0, 0, 0, ...	[4, 3, 231, 676, 293, 10, 0, 0, 0, 0, 0, 0, 0,...
323428	[2, 28, 7495, 6195, 3493, 7495, 6195, 44, 3493...	[4, 204, 32, 36566, 6195, 1289, 7495, 10332, 0...	[2, 28, 7495, 6195, 3493, 7495, 6195, 44, 3493...
323429	[15, 995, 2742, 623, 25, 54, 5, 0, 0, 0, 0, 0,...	[15, 995, 2742, 623, 25, 54, 0, 0, 0, 0, 0, 0,...	[15, 995, 2742, 623, 25, 54, 5, 0, 0, 0, 0, 0,...
323430	[2, 2156, 10635, 9057, 436, 3897, 2005, 887, 4...	[2, 463, 2156, 2163, 2291, 3897, 382, 692, 857...	[2, 2156, 10635, 9057, 436, 3897, 2005, 887, 4...

323431 rows × 3 columns

Figure 6. Tokenization into sequence of integers

3. FROM-SCRATCH ARCHITECTURE

Our first idea was to try using an LSTM network on embedding trained from scratch. As anticipated in Chapter 2, we use the *tokenizer* field as a representation to give in input to our neural networks, while the labels are obtained from the *is_duplicate* field. The output layer consists of a single neuron with sigmoid activation function. Multiple combinations were tried, we report only the best ones.

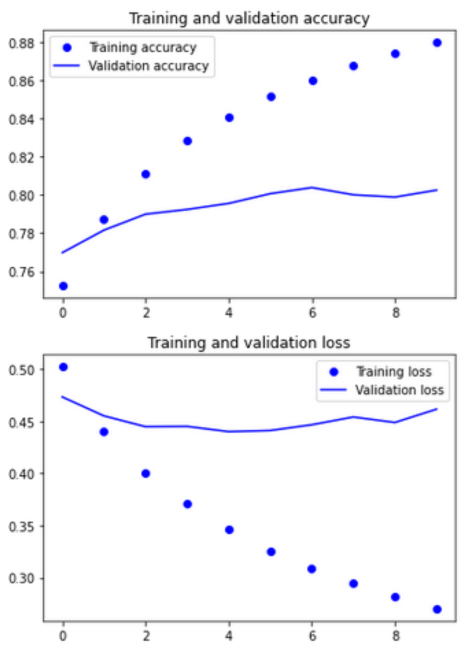
3.1. Basic Experiment

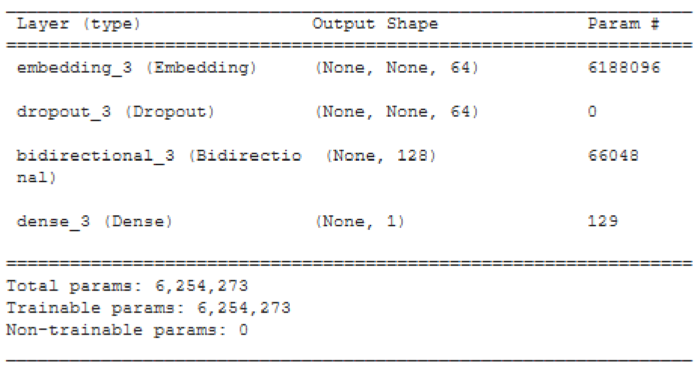
We initially tested the use of a simple bidirectional LSTM over the Embedding layer.

In this experiment we decided to use embedding of dimension 64, Dropout to fight the high overfitting, and a bidirectional LSTM layer with an output of dimensionality 64.

Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
0.3468	0.8408	0.4402	0.7956

Table 2. Training and validation results





```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, None, 64)          6188096

dropout_3 (Dropout)          (None, None, 64)          0

bidirectional_3 (Bidirectio  (None, 128)               66048
nal)

dense_3 (Dense)              (None, 1)                 129

=================================================================
Total params: 6,254,273
Trainable params: 6,254,273
Non-trainable params: 0
_________________________________________________________________
```

Figure 7. Learning curves and Summary of the network

The network can learn correctly, but the validation curve follows the training curve only up to a certain level, then there is a high overfitting. By changing the parameters of the network, we were not able to improve this situation of overfitting, even though we tried to reduce the number of parameters of the network.

Precision		Recall		Macro F1-Score
Not duplicates	*Duplicates*	*Not duplicates*	*Duplicates*	
0.8588	0.7146	0.8160	0.7745	0.7901

Table 3. Performance on test set

Already with this model the performances on the test set are satisfactory, we get a macro F1 score of 79%, with a model that is able to understand *Not duplicates* class better than *Duplicates* class.

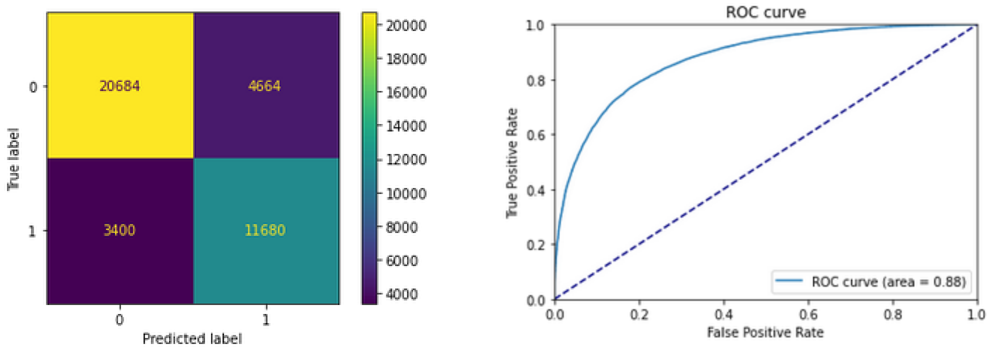


Figure 8. Confusion matrix and ROC curve

3.2. Class Weight for unbalanced classes

The two classes are slightly unbalanced, in fact we have 63.14% of pairs of non-duplicates questions. To improve the training, we tried to use the class weight mechanism. The whole purpose is to penalize the misclassification made by the minority class by setting a higher weight and at the same time reducing weight for the majority class. The weight for the minority class has been computed as 1.35, while for the majority class we have 0.79.

Precision		**Recall**		**Macro F1-Score**
Not duplicates	*Duplicates*	*Not duplicates*	*Duplicates*	
0.8754	0.6609	0.7495	0.8206	0.7698

Table 4. Performance on test set

With this procedure we have increased the precision of the class *Not duplicates* and the recall of the class *Duplicates* but losing something in the remaining values. In fact, the macro F1-score is decreased.

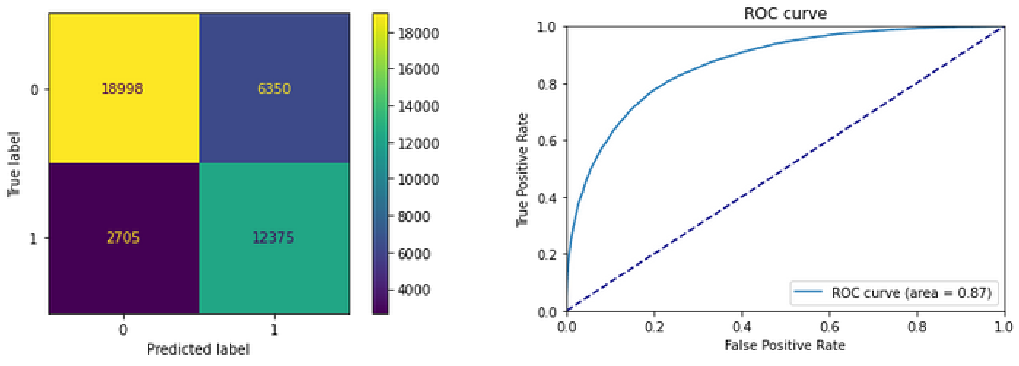


Figure 9. Confusion matrix and ROC curve

We can easily see that we have correctly predicted more sample of the minority class, but at the cost of a greater loss to the majority class.

3.3. Data Augmentation to balance classes

Another way to balance the classes is to augment the samples of the minority class. It is not easy to do data augmentation in the case of texts, in fact complex techniques are required such as exchanging some words with synonyms taken from a dictionary. Our idea is simple: we swapped the order of some of the questions to create new samples that are sure to be valid and at no cost. Obviously only the training set has been balanced, while the validation set, and the test set must keep the same distribution.

Precision		**Recall**		**Macro F1-Score**
Not duplicates	*Duplicates*	*Not duplicates*	*Duplicates*	
0.8792	0.6810	0.7709	0.8220	0.7832

Table 5. Performance on test set

The results are better than those obtained with the class weight, showing how the network has managed to learn something more thanks to the new samples, which turn out to be valid. But the macro F1-score is also this time lower than the initial one.

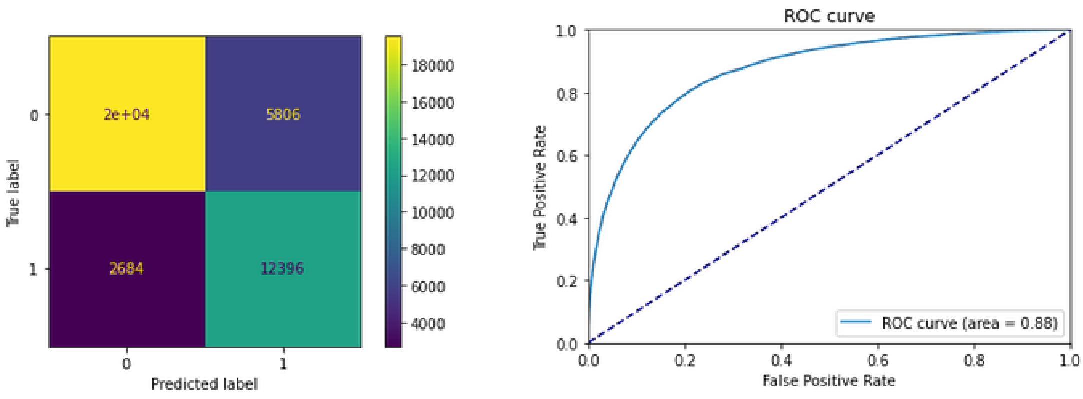


Figure 10. Confusion matrix and ROC curve

In comparison to the experiment with class weight we have improved the classification of the *Duplicates* class while losing even less on the *Not duplicates* class.

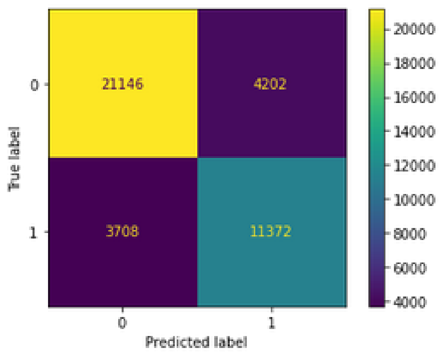
3.4. Total Data Augmentation

The last noteworthy experiment is with full data augmentation. In this case we extended the augmentation to all pairs, thus doubling our training set, at the cost of a much longer training time.

Precision		**Recall**		**Macro F1-Score**
Not duplicates	*Duplicates*	*Not duplicates*	*Duplicates*	
0.8508	0.7302	0.8342	0.7541	0.7922

Table 6. Performance on test set

Using this approach, we achieved the best results, obtaining the highest macro F1-score value, which is the main benchmark metric. In this experiment, we obtained the best balance in errors between the two classes.



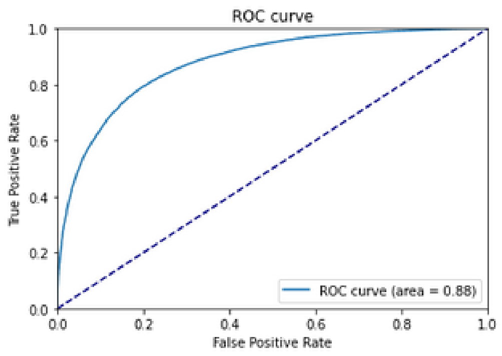


Figure 11. Confusion matrix and ROC curve.

4. SIAMESE NEURAL NETWORK

A **Siamese neural network** is an artificial neural network that uses the same weights while working on two different input vectors to compute comparable output vectors. The main feature of Siamese neural networks – that makes them particularly well suited to our task – is that they focus on learning internal representations that place similar samples close together (and dissimilar ones far apart).

In 2016, J. Mueller & A. Thyagarajan proposed a Siamese adaptation of the Long Short-Term Memory (LSTM) network for labelled data comprising pairs of variable-length sequences [3]. Inspired by their work, we decided to use a similar architecture to assess semantic similarity between question pairs in our dataset. The general architecture of the Siamese network we used is the one shown in Figure 12.

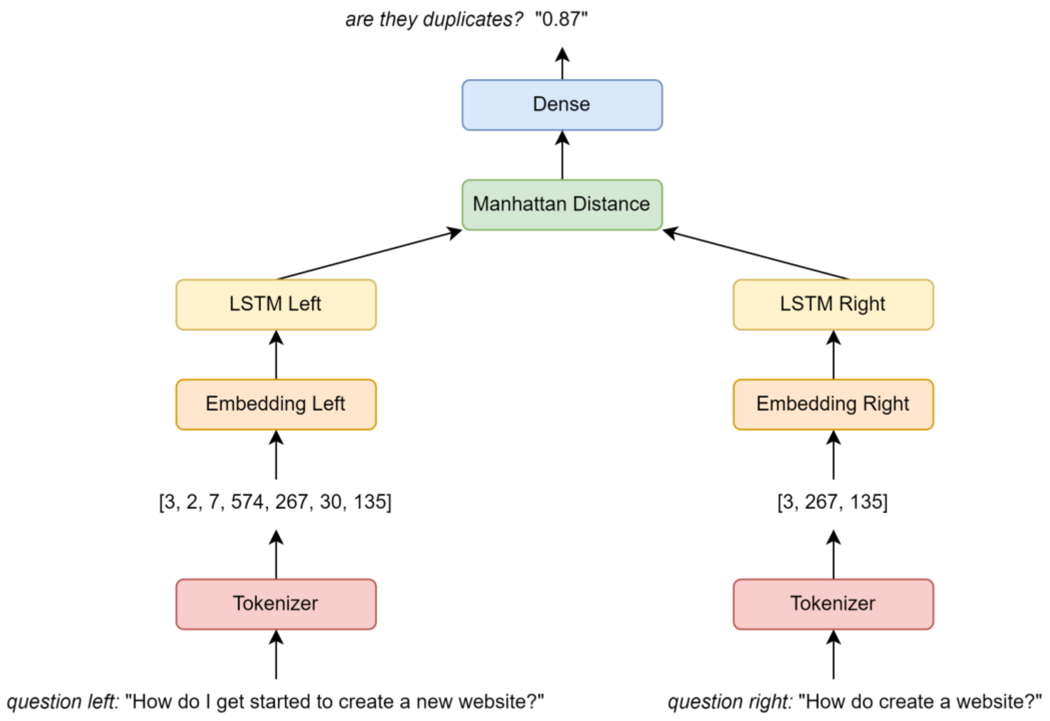

are they duplicates? "0.87"
Dense
Manhattan Distance
LSTM Left
LSTM Right
Embedding Left
Embedding Right
[3, 2, 7, 574, 267, 30, 135]
[3, 267, 135]
Tokenizer
Tokenizer
question left: "How do I get started to create a new website?"
question right: "How do create a website?"

Figure 12. General architecture of our Siamese neural network

As can be seen, first the questions are tokenized using the set of training questions as vocabulary. Please note that, prior to tokenization, a basic pre-processing is performed, as explained in Chapter 2. Next, the tokenized versions of the questions go through an Embedding layer whose weights were initialized by mapping the vocabulary built on the training set onto pre-trained word embeddings. The embedded sequences then go through an LSTM layer.

Finally, the Manhattan distance (also known as L1-norm) between the two outputs is calculated as follows:

$$L_1[(x_1, \dots, x_n), (y_1, \dots, y_n)] = \sum_{i=1}^{n} |x_i - y_i|$$

On top of it all, there is a single-neuron Dense layer with sigmoid activation function. Its purpose is to output the probability that the two questions are duplicates.

We performed several experiments, using various combinations of the parameters to find the best model. Below, we will show only the most significant ones.

4.1. GloVe embeddings

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space [4].

In our case, we decided to use the word vectors that were trained on Wikipedia 2014 and Gigaword 5 data. The downloadable archive contains text-encoded vectors of various sizes: 50-dimensional, 100-dimensional, 200-dimensional, and 300-dimensional. Our experiments were performed with the 50- and 100-dimensional ones, due to the fact that larger embeddings excessively increased the time required for training.

4.1.1. LSTM with 50 units

The first experiment was performed using an LSTM layer with 50 hidden units. Starting with the tokenized version of the questions, only the first 25 tokens were taken as input to the Embedding layer, which was constructed from vectors with both size 50 and 100. Please note that `trainable=False` has been set to keep the Embedding weights fixed.

The network was trained using patience as an early stopping technique and Mean Squared Error (MSE) as the loss function. Table 7 shows that the performance obtained by the two models are comparable on both the training set and the validation set, with the 50-dimensional embeddings being better by only about 0.005.

Embedding Dimensions	**Training Loss**	**Training Accuracy**	**Validation Loss**	**Validation Accuracy**
50	0.1298	0.8203	0.1443	0.7958
100	0.1259	0.8256	0.1392	0.8037

Table 7. Training and validation results

The trend of the learning curves is also very similar, as can be seen in Figure 13. Performance on the validation set seems to follow that on the training set, even if after a few epochs the plateau is reached, and the loss values become almost constant.

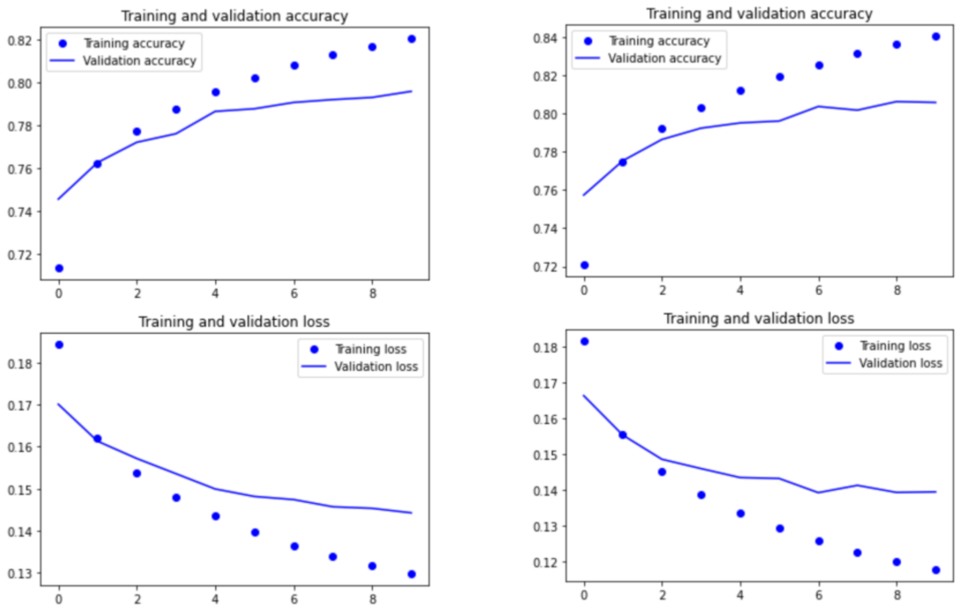


Figure 13. Learning curves, 50D embedding (left) and 100D embedding (right)

Embedding Dimensions	**Precision**		**Recall**		**Macro F1-Score**
	Not duplicates	*Duplicates*	*Not duplicates*	*Duplicates*	
50	0.8468	0.7135	0.8208	0.7504	0.7825
100	0.8336	0.7479	0.8572	0.7123	0.7875

Table 8. Performance on test set

The model also works quite well on the test set and especially on the "not duplicates" (0) class, as can be seen both in Table 8 and in Figure 14. The better performance on class 0 is probably due to the fact that it is the majority class, as we have already seen in previous experiments.

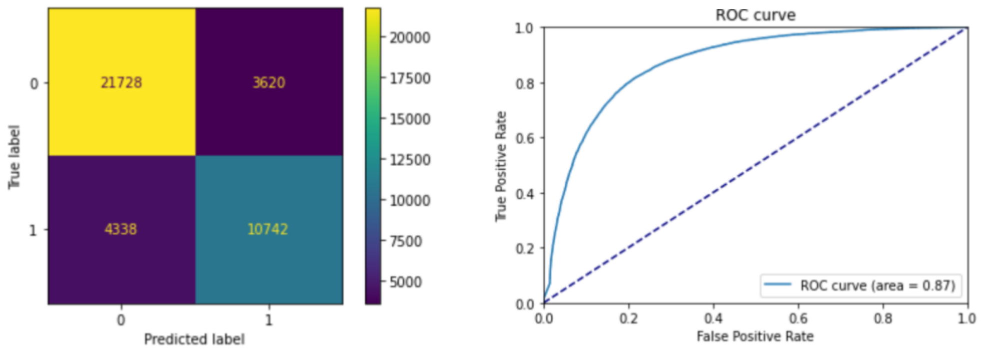


Figure 14. Confusion matrix and ROC curve (100D embedding)

4.1.2. Bidirectional LSTM

Next, we tried to use a Bidirectional LSTM layer with the aim of exploiting not only information about the previous context but also the future one. However, as shown in Table 9, this did not lead to the improvements in performance that we would have expected. This is likely due to the fact that, due to otherwise prohibitive training times, the number of units in the LSTM has been reduced to 10.

Embedding Dimensions	**Precision**		**Recall**		**Macro F1-Score**
	Not duplicates	*Duplicates*	*Not duplicates*	*Duplicates*	
50	0.8676	0.6144	0.6931	0.8221	0.7369
100	0.8388	0.6833	0.7951	0.7432	0.7642

Table 9. Performance on test set

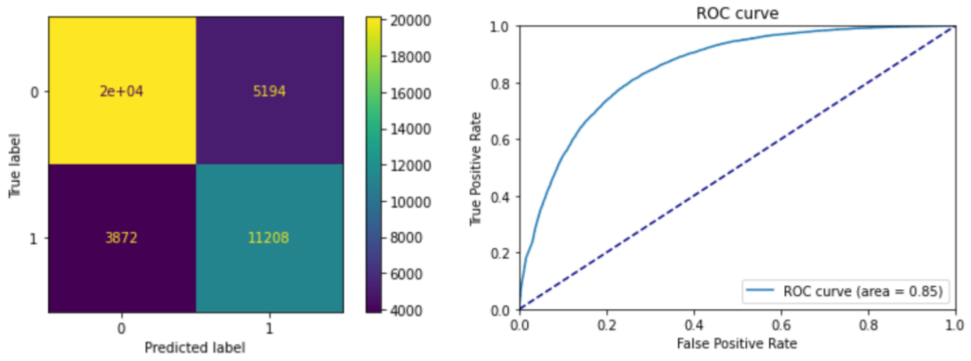
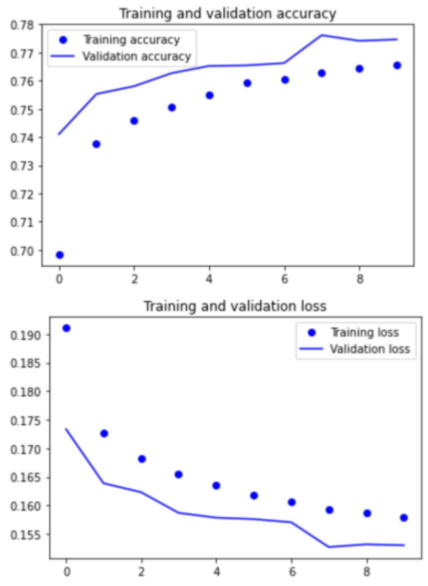
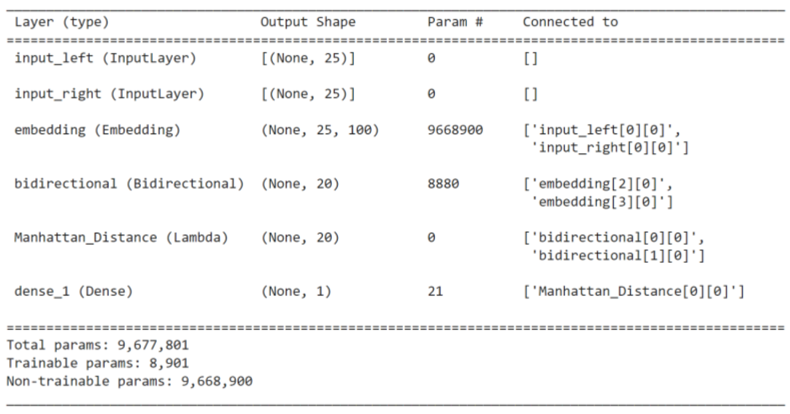


Figure 15. Confusion matrix and ROC curve (100D embedding)





```
_______________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
===============================================================================================
 input_left (InputLayer)        [(None, 25)]         0           []

 input_right (InputLayer)       [(None, 25)]         0           []

 embedding (Embedding)          (None, 25, 100)      9668900     ['input_left[0][0]',
                                                                  'input_right[0][0]']

 bidirectional (Bidirectional)  (None, 20)           8880        ['embedding[2][0]',
                                                                  'embedding[3][0]']

 Manhattan_Distance (Lambda)    (None, 20)           0           ['bidirectional[0][0]',
                                                                  'bidirectional[1][0]']

 dense_1 (Dense)                (None, 1)            21          ['Manhattan_Distance[0][0]']

===============================================================================================
Total params: 9,677,801
Trainable params: 8,901
Non-trainable params: 9,668,900
_______________________________________________________________________________________________
```

Figure 16. Learning curves and summary of the network (100D embedding)

The interesting thing we could notice when looking at the learning curves in Figure 17 is that the model seemed to suffer from a little underfitting in that the performance on the validation set is better than that on the training set. In an attempt to counteract this, we tried to increase the number of parameters in the model, making the Embedding layer trainable as well. This increased the number of trainable parameters from nearly 9k to more than 9 million.

Such a high number of parameters is risky because it could result in the opposite problem, that is, overfitting. However, as shown in Table 10, this allowed us to achieve better results on both classes with an improvement in macro F1-score of about 8%.

Embedding Dimensions	**Precision**		**Recall**		**Macro F1-Score**
	Not duplicates	*Duplicates*	*Not duplicates*	*Duplicates*	
100	0.8862	0.7495	0.8371	0.8193	0.8219

Table 10. Performance on test set

The model performs well on both classes, but the most impressive improvement was on the "duplicates" class. Indeed, out of 15080 test samples belonging to that class, only 2725 are misclassified (Figure 17).

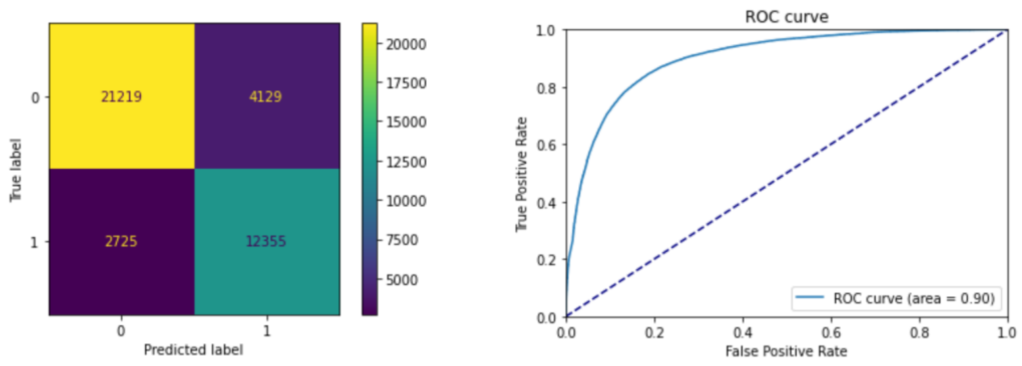


Figure 17. Confusion matrix and ROC curve

4.1.3. Data Augmentation

As we anticipated in the previous section, our data is slightly unbalanced and this may lead the network to perform better on the majority class, i.e., the 'not duplicates' class. To solve this problem, we considered using data augmentation in the training set for the minority class (see Section 3.3).

In this experiment, we first trained the Siamese neural network with bidirectional LSTM layer while keeping the embedding layer weights fixed. Next, we "unfroze" the weights of the embedding layer and performed finetuning, with the goal of improving the performance of the network and making it more suitable for our data.

As we can see from both Table 11 and Figure 18, indeed the network performance had improved on class 1, but at the expense of class 0. Consequently, the macro F1-score was comparable to the previous one and we couldn't claim to have any advantages in using data augmentation.

Embedding Dimensions	**Precision**		**Recall**		**Macro F1-Score**
	Not duplicates	*Duplicates*	*Not duplicates*	*Duplicates*	
100	0.9057	0.7290	0.8102	0.8583	0.8218

Table 11. Performance on test set

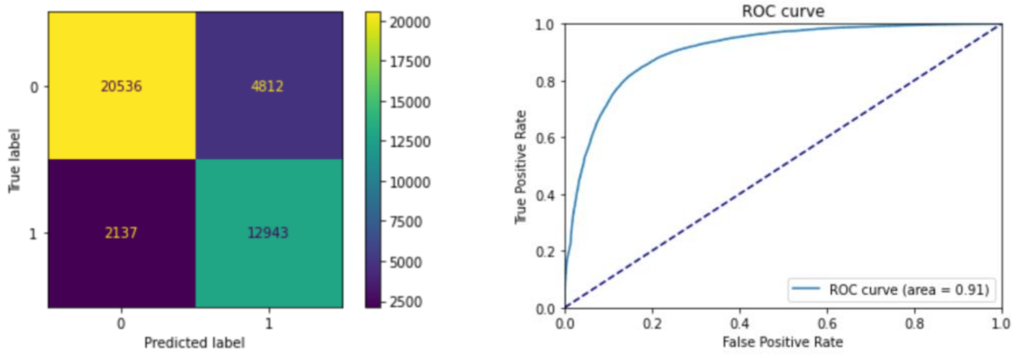


Figure 18. Confusion matrix and ROC curve

After experimenting with different combinations of hyperparameters without being able to obtain better results than those shown, we decided to change the embedding layer to assess whether another dimension and other weights could benefit the model. Hence, we will show in the next section the results obtained with word2vec embeddings.

4.2. Word2vec embeddings

Word2vec is one of the most popular techniques to learn word embeddings using shallow neural network [5]. The word2vec algorithm uses a neural network model to learn word associations from a large corpus of text. It can utilize either of two model architectures to produce a distributed representation of words: continuous bag-of-words (CBOW) or continuous skip-gram.

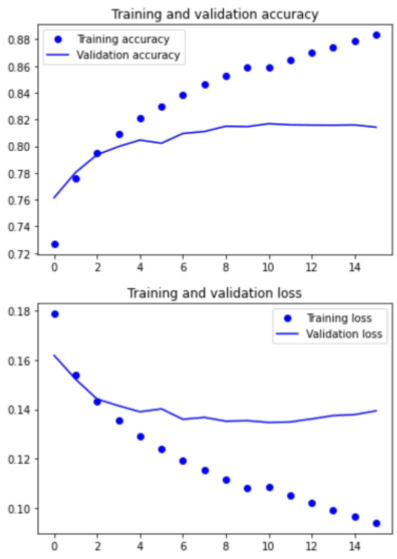
We decided to use the 300-dimensional vectors that were pre-trained on part of Google News dataset (about 100 billion words). Due to the large dimensionality of these embeddings, the experiments we have been able to perform are not many. However, the goal was to compare their results with those obtained using GloVe pre-trained vectors.

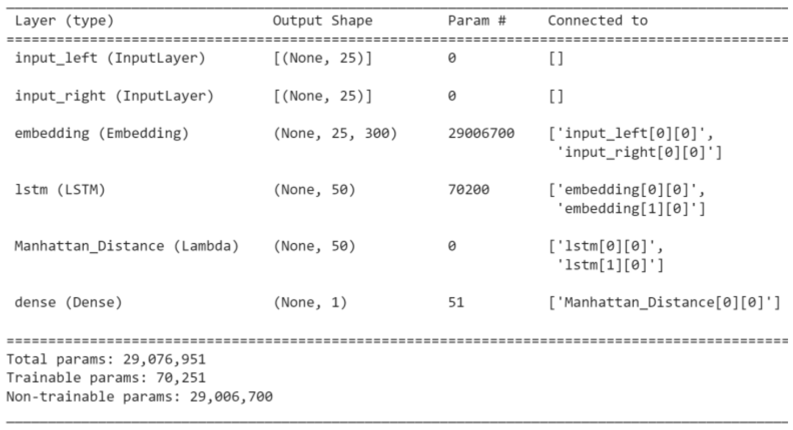
4.2.1. LSTM with 50 units

As with GloVe embeddings, our first experiment was with a 50-unit LSTM layer. Figure 19 and Table 12 show the trend during training and the best results on the training set and the validation set. As we can see, the network seems to learn too much from the training set, as already after a few epochs the plateau is reached on the validation set. However, even using regularization techniques, we could not overcome this result.

Trainable	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
False	0.1083	0.8587	0.1347	0.8169

Table 12. Training and validation results





```
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_left (InputLayer)         [(None, 25)]         0           []

input_right (InputLayer)        [(None, 25)]         0           []

embedding (Embedding)           (None, 25, 300)      29006700    ['input_left[0][0]',
                                                                  'input_right[0][0]']

lstm (LSTM)                     (None, 50)           70200       ['embedding[0][0]',
                                                                  'embedding[1][0]']

Manhattan_Distance (Lambda)     (None, 50)           0           ['lstm[0][0]',
                                                                  'lstm[1][0]']

dense (Dense)                   (None, 1)            51          ['Manhattan_Distance[0][0]']

==================================================================================================
Total params: 29,076,951
Trainable params: 70,251
Non-trainable params: 29,006,700
__________________________________________________________________________________________________
```

Figure 19. Learning curves and summary of the network

As for the performance on the test set, we can see from Table 13 how these are slightly better than those obtained using GloVe embeddings. This is most likely due to the larger size of the embedding, 300 versus 50 or 100.

Trainable	Precision		Recall		Macro F1-Score
	Not duplicates	*Duplicates*	*Not duplicates*	*Duplicates*	
False	0.8689	0.7300	0.8261	0.7905	0.8030
True	0.8743	0.7860	0.8722	0.7893	0.8304

Table 13. Performance on test set

Furthermore, after training the network while keeping the embedding layer weights fixed, we tried finetuning. In this case therefore we had more than 29 million parameters to adjust, but fortunately only for a few epochs.

Performing finetuning allowed us to better adapt the network to our problem and thus achieve better performance. As a matter of fact, the macro F1-score improved by about 3% and, as can be seen from Figure 20, we obtained an AUC of 0.91.

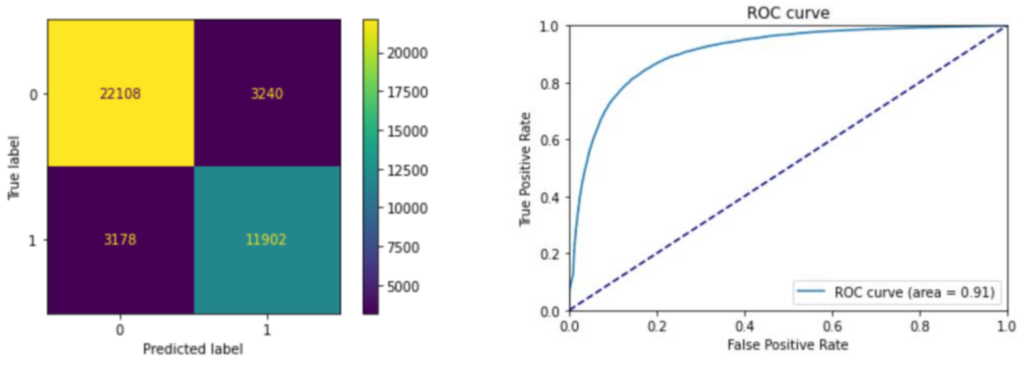


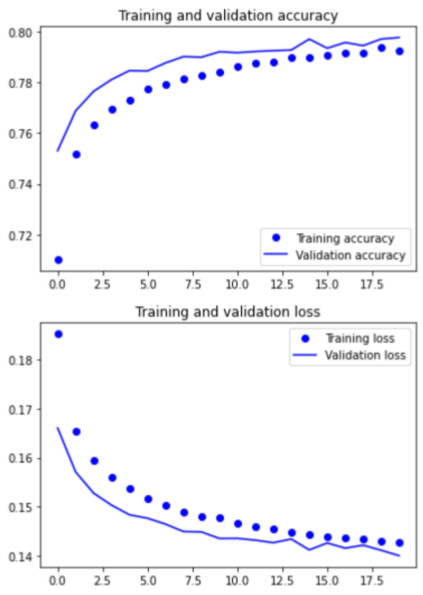
Figure 20. Confusion matrix and ROC curve (trainable = True)

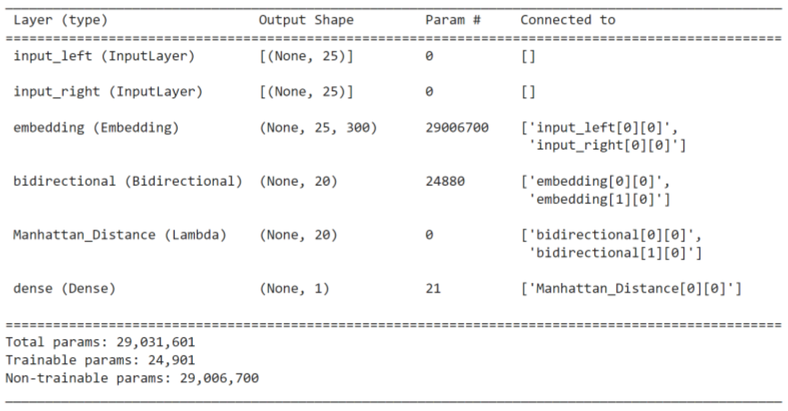
4.2.2. Bidirectional LSTM

The second experiment we ran was using a bidirectional LSTM layer. As you can see from the curves in Figure 21 and the results in Table 14, the model seemed to learn well, albeit slowly. In fact, it is possible to see a small underfitting, as the model seems to perform better on the validation set than on the training set.

Trainable	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
False	0.1492	0.7924	0.1401	0.7976

Table 14. Training and validation results





```
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_left (InputLayer)         [(None, 25)]         0           []

input_right (InputLayer)        [(None, 25)]         0           []

embedding (Embedding)           (None, 25, 300)      29006700    ['input_left[0][0]',
                                                                  'input_right[0][0]']

bidirectional (Bidirectional)   (None, 20)           24880       ['embedding[0][0]',
                                                                  'embedding[1][0]']

Manhattan_Distance (Lambda)     (None, 20)           0           ['bidirectional[0][0]',
                                                                  'bidirectional[1][0]']

dense (Dense)                   (None, 1)            21          ['Manhattan_Distance[0][0]']

==================================================================================================
Total params: 29,031,601
Trainable params: 24,901
Non-trainable params: 29,006,700
```

Figure 21. Learning curves and summary of the network

Trainable	Precision		Recall		Macro F1-Score
	Not duplicates	*Duplicates*	*Not duplicates*	*Duplicates*	
False	0.8561	0.8121	0.7093	0.7706	0.7861

Table 15. Performance on test set

As for the test set, the model does not have optimal performance. In fact, as can be seen in Table 15 and Figure 22 the results are worse than the previous ones. Finetuning the embedding layer would almost certainly have made improvements, but due to prohibitive training time, it could not be done.

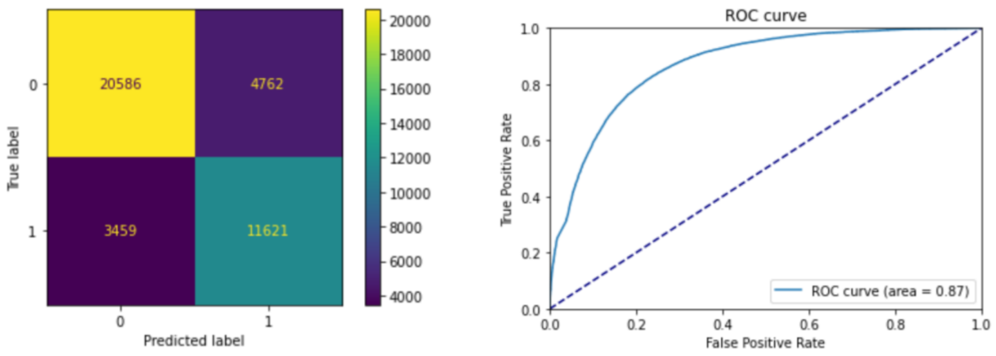


Figure 22. Confusion matrix and ROC curve

5. TRANSFER LEARNING AND FEATURE REUSE

To further improve our results, we decided to use transfer learning to extract features from our questions, to later use them in deep neural networks. To this aim, we exploited pre-trained models, made available through the python library *sentence-transformers* [6], and a Doc2Vec model we trained as Chapter 5.3 explains.

5.1. Way of Extracting the Features

For each set obtained as explained in Chapter 1.1.1, the features for each question were extracted and then combined in order to match the correct question pairs of the original sets. A quick example of that is presented in Figure 23.

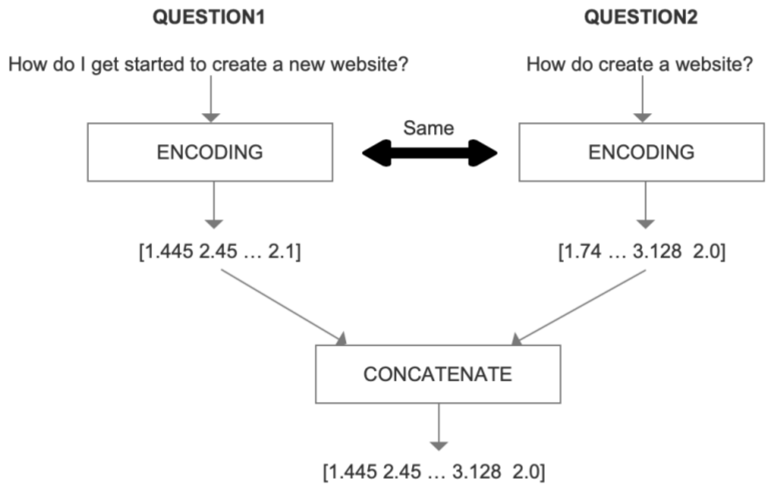


Figure 23. Feature creation

5.2. Experiments Typologies

Four experiments were done using the deep neural network presented above:

1. Basic experiment: training of the classifier using training and validation set.
2. Class weight: adding class weight in the fit function
3. Augmentation: doubling the training set switching question1 and question2 in each pair.
4. Augmentation and class weight: combination of method 3 and 4.

For all experiment the same loss and optimizer were used, *binary cross entropy* and *Adam* with learning rate equal to 10^{-3} respectively.

5.3. Doc2Vec

Doc2Vec is an unsupervised algorithm to generate vectors for sentence/paragraphs/documents [7]. This model was used since unlike sequence models like RNN, where word sequence is captured in generated sentence vectors, Doc2Vec sentence vectors are word order independent. In order to make use of it we imported the *gensim* library [8], which provided the model.

The first step was to train it. The training was done considering all the questions present in our training set, thus both *question1* and *question2*. Also, other hyperparameters were needed and their configuration are reported in Table 16.

Vector_size	100
Window	5
Min_count	3
Epochs	25

Table 16. Doc2Vec hyperparameters

The explanation to those parameters is the following:

- *Vector_size*: is the feature vector (100 is the default value)
- *Window*: is the maximum distance between the current and predicted word within a sentence (5 is the default value)
- *Min_counts*: ignores all words with total frequency lower than this value.

5.3.1.The Deep Neural Network Used

The deep neural network used to classify correctly the pairs is made of multiple dense layers with an activation *ReLU* activation function, terminating with a single unit with a sigmoid activation function as previously done in Chapter 3. The structure of the network is shown in detail in Figure 24.

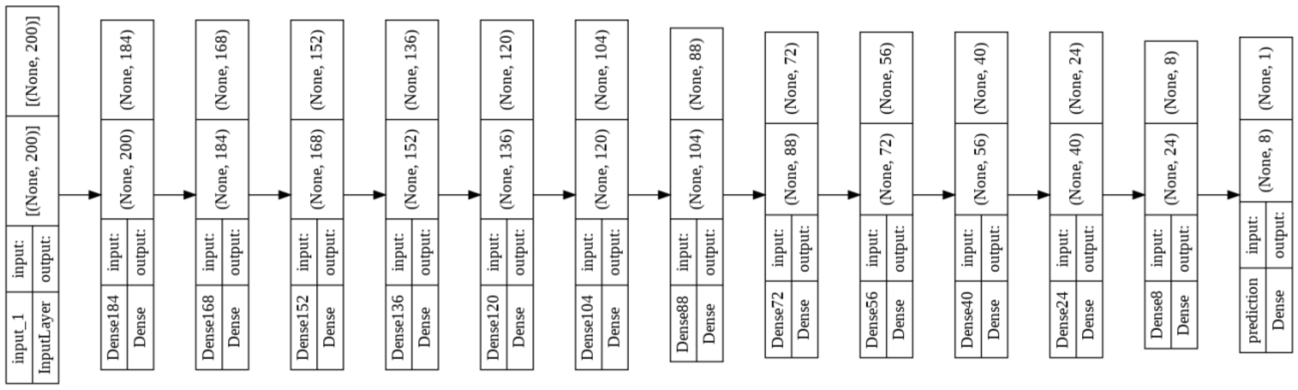

input_1 InputLayer input: output: [(None, 200)] [(None, 200)]
Dense184 Dense input: output: (None, 200) (None, 184)
Dense168 Dense input: output: (None, 184) (None, 168)
Dense152 Dense input: output: (None, 168) (None, 152)
Dense136 Dense input: output: (None, 152) (None, 136)
Dense120 Dense input: output: (None, 136) (None, 120)
Dense104 Dense input: output: (None, 120) (None, 104)
Dense88 Dense input: output: (None, 104) (None, 88)
Dense72 Dense input: output: (None, 88) (None, 72)
Dense56 Dense input: output: (None, 72) (None, 56)
Dense40 Dense input: output: (None, 56) (None, 40)
Dense24 Dense input: output: (None, 40) (None, 24)
Dense8 Dense input: output: (None, 24) (None, 8)
prediction Dense input: output: (None, 8) (None, 1)

Figure 24. Deep Neural Network used for Doc2Vec features

5.3.2.Experiments Results

The experiment results are:

Experiment	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy	Precision		Recall		Macro F1-Score
					Not duplicates	*Duplicates*	*Not duplicates*	*Duplicates*	
Basic	0.51581	0.7434	0.5158	0.7390	**0.9035**	**0.7871**	**0.8640**	**0.8449**	**0.8491**
Class weight	0.5057	0.7362	0.5525	0.7306	0.8166	0.6186	0.7350	0.6666	0.7201
Augmentation	**0.4927**	**0.7504**	**0.5029**	0.7447	0.7818	0.6770	0.8261	0.6125	0.7233
Class weight + augmentation	0.4983	0. 7401	0.5065	**0.7437**	0.8185	0.6632	0.7520	0.6737	0.7288

Table 17. Experiment results for Doc2Vec features

The model that performs better was chosen by us as the one trained in the “basic” experiment, since it outperforms the others on the test set measures (precision, recall) and the macro F1-score is significantly greater than the others, and for what concerns the training and validation, even if it is not the best, it is also comparable to the best values obtained. Another import thing to consider is that the “Basic” model is the one than is more balanced in guessing corrected labels for the two classes, without privileging one of them. Thus, we reported here the learning curves, the confusion matrix and ROC curve for the “basic experiment”.

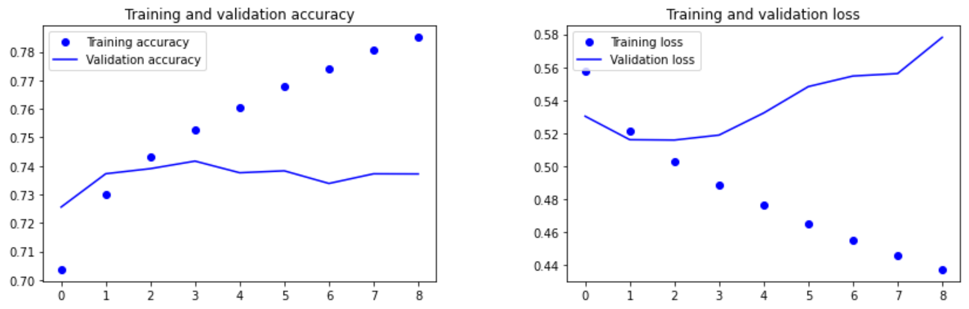


Figure 25. Learning curves for "Basic" experiment

The learning curves have a strange behaviour, in fact the curves tend to get worse as the epochs go by. This problem could be due to two main things; first, a learning rate maybe too high; second, since the *generalization gap* becomes larger and larger with each epoch, this could indicate that the features obtained on the validation set have a very different information content from those of the training set. Anyway, since we use *keras's call-backs* only the best model on the validation loss was saved (i.e., it is the one reported in the tables and figures). These considerations apply also to all the other analysis done in this Chapter.

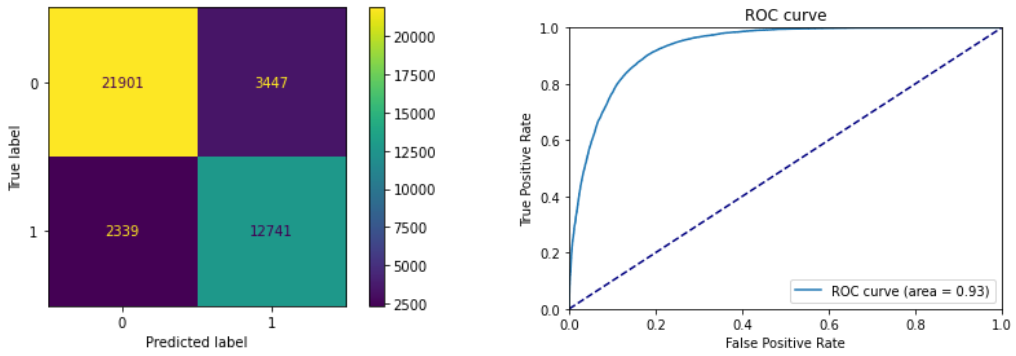


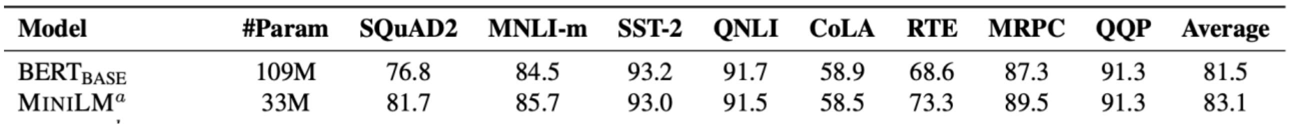
Figure 26. Confusion matrix and ROC curve for "Basic" experiment

For what concerns the ROC curve and the confusion matrix, the result obtained are far better than those obtained with previous approaches suggesting this “Transfer Learning” approach as valuable.

5.4. Paraphrase-MiniLM-L12-v2

MiniLM is a Transformer based Linguistic Model distillation proposed by Microsoft [9], in which the authors distilled the self-attention module of the last Transformer layer of the teacher model. The MiniLM-L12-v2 release is the uncased 12-layer model with 384-hidden size distilled from a Microsoft’s pre-trained UniLM-v2 model in BERT-Base size.

Specifically, the paraphrase-mini-L12-v2 model is sentence-transformer based using the Microsoft's MiniLM for paraphrasing, trained on multiple dataset[1].

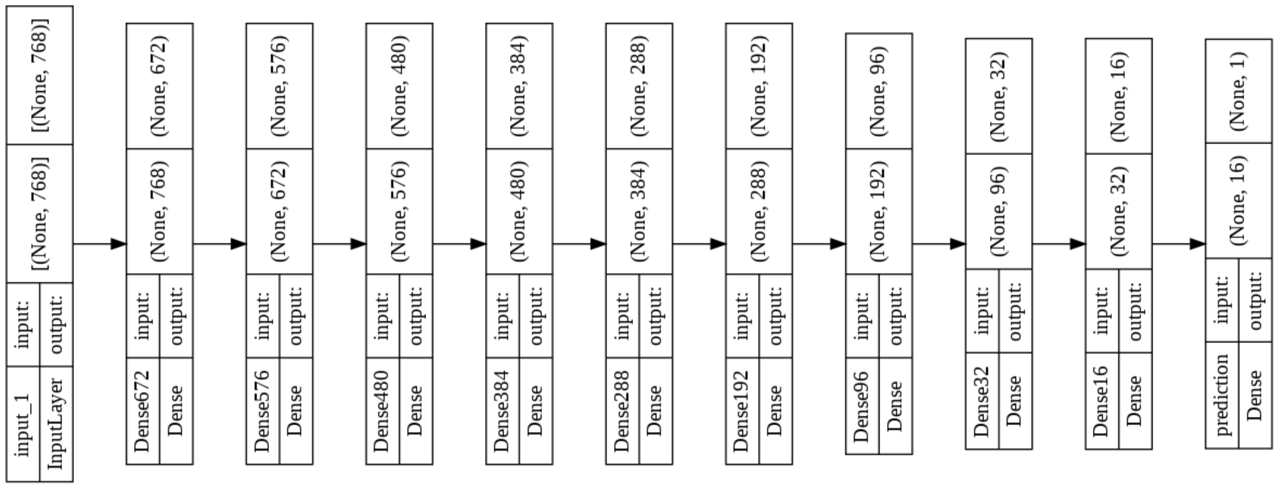


Model	#Param	SQuAD2	MNLI-m	SST-2	QNLI	CoLA	RTE	MRPC	QQP	Average
BERT$_{BASE}$	109M	76.8	84.5	93.2	91.7	58.9	68.6	87.3	91.3	81.5
MINILM^{a}	33M	81.7	85.7	93.0	91.5	58.5	73.3	89.5	91.3	83.1

Table 18. Results of MiniLM-L12-v2 and BERT$_{BASE}$ with 12-layer Transformer, 768-hidden size, and 12 self-attention heads.

5.4.1. The Deep Neural Network Used

The deep neural network used to classify correctly the pairs is made of multiple dense layers with an activation ReLu activation function, terminating with a single unit with a sigmoid activation function as previously done in Chapter 3. The structure of the network is shown in detail in Figure 27.



input_1 | InputLayer | input: [(None, 768)] | output: [(None, 768)]
Dense672 | Dense | input: (None, 768) | output: (None, 672)
Dense576 | Dense | input: (None, 672) | output: (None, 576)
Dense480 | Dense | input: (None, 576) | output: (None, 480)
Dense384 | Dense | input: (None, 480) | output: (None, 384)
Dense288 | Dense | input: (None, 384) | output: (None, 288)
Dense192 | Dense | input: (None, 288) | output: (None, 192)
Dense96 | Dense | input: (None, 192) | output: (None, 96)
Dense32 | Dense | input: (None, 96) | output: (None, 32)
Dense16 | Dense | input: (None, 32) | output: (None, 16)
prediction | Dense | input: (None, 16) | output: (None, 1)

Figure 27. Deep Neural Network used for paraphrase-mini-L12-v2 features

5.4.2. Experiments Results

The experiment results are:

Experiment	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy	Precision		Recall		Macro F1-Score
					Not duplicates	*Duplicates*	*Not duplicates*	*Duplicates*	
Basic	**0.2683**	**0.8802**	0.3089	**0.8614**	0.9035	0.7871	0.8640	0.8449	0.8491
Class weight	0.2980	0.8601	0.3155	0.8581	0.8929	0.8091	0.8846	0.8217	0.8520
Augmentation	0.2726	0.8779	**0.3080**	0.8644	0.8945	0.8077	0.8832	0.8249	0.8525
Class weight + augmentation	0.2755	0. 8726	0.3127	0.8577	**0.9070**	**0.7980**	**0.8721**	**0.8497**	**0.8561**

Table 19. Experiment results for paraphrase-mini-L12-v2 features

[1] AIINLI, sentence-compression, SimpleWiki, altlex, msmarco-triplets, quora_duplicates, coco_captions, flickr30k_captions, yahoo_answers_title_question, S2ORC_citation_pairs, stackexchange_duplicate_questions, wiki-atomic-edits

The model that performs better was chosen by us as the one trained in the “class weight + augmentation” experiment, since it outperforms the others on the test set measures (precision, recall, F1), and for what concerns the training and validation, even if it is not the best, it is comparable to the best values obtained. Thus, we reported here the learning curves, the confusion matrix and ROC curve for the “class weight + augmentation”.

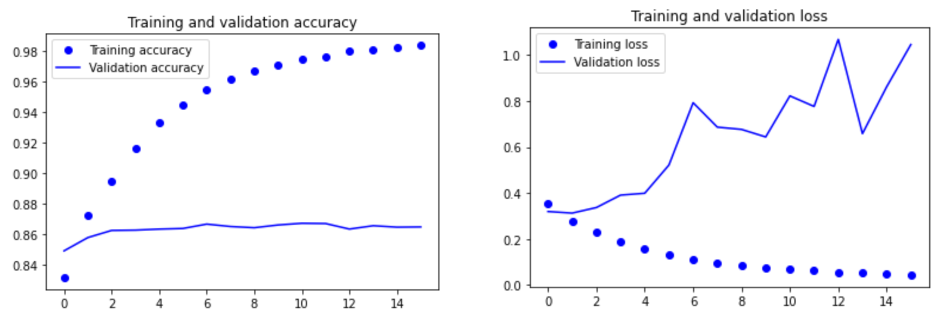


Figure 28. Learning curves for "class weight + augmentation" experiment (paraphrase-mini-L12-v2)

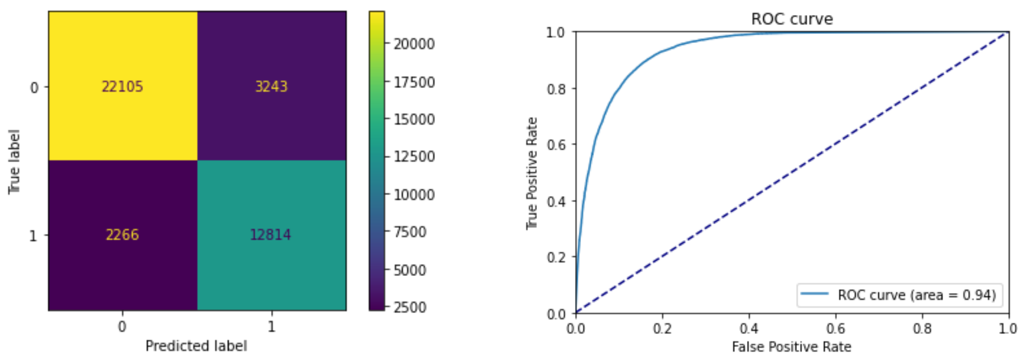


Figure 29. Confusion matrix and ROC curve for "class weight + augmentation" experiment (paraphrase-miniLM-L12-v2)

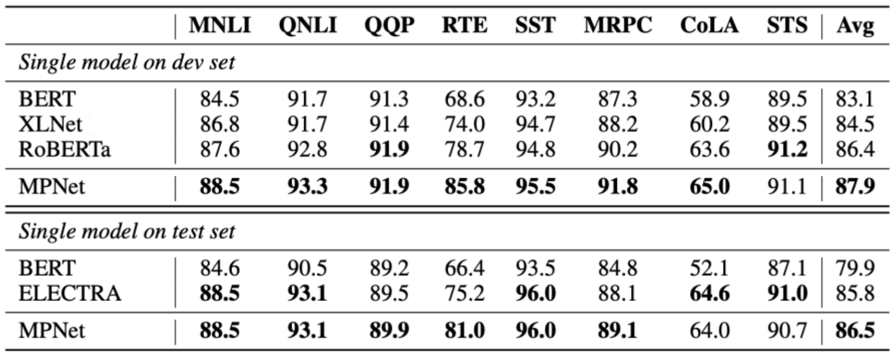
Confronting the confusion matrix here obtained with the best one for Doc2Vec, we achieved in this case better results for the true positive and true negative classes, with an increment of 205 negative and 73 positives correctly classified.

5.5. All-mpnet-base-v2

MPNet is a model proposed by Microsoft Research in 2020 [10]. It mixes BERT and XLNet strategies: BERT adopts masked language modeling (MLM) for pre-training; since BERT neglects dependency among predicted tokens, XLNet introduces permutated language modeling (PLM) for pre-training to address this problem.

However, XLNet does not leverage the full position information of a sentence and thus suffers from position discrepancy between pre-training and fine-tuning. This model, the proposed MPNet is a novel pre-training method that inherits the advantages of BERT and XLNet and avoids their limitations. MPNet leverages the dependency among predicted tokens through permuted language modeling (vs. MLM in BERT) and takes auxiliary position information as input to make the model see a full sentence and thus reducing the position discrepancy (vs. PLM in XLNet).

Specifically, the all-mpnet-base-v2 is based on MPNet-base and fine-tuned in on a 1B sentence pairs dataset, using a contrastive learning objective: given a sentence from the pair, the model should predict which out of a set of randomly sampled other sentences, was actually paired with it in our dataset.

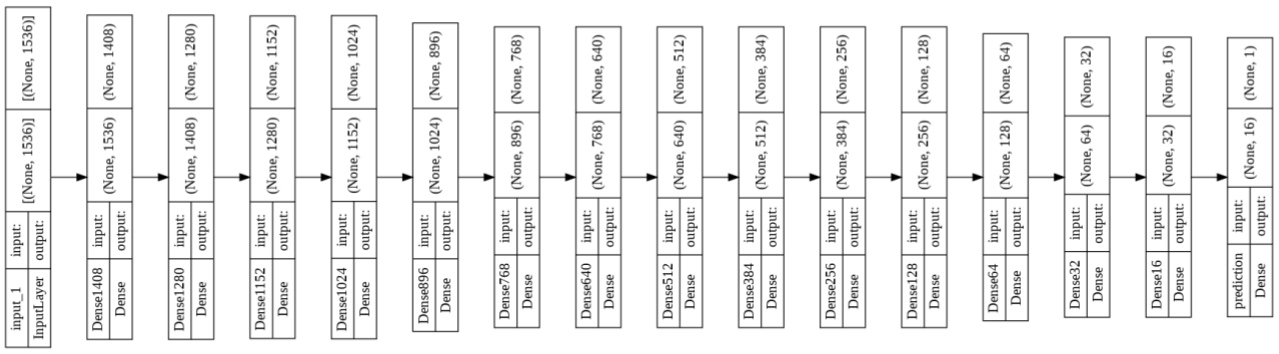


	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS	Avg
Single model on dev set									
BERT	84.5	91.7	91.3	68.6	93.2	87.3	58.9	89.5	83.1
XLNet	86.8	91.7	91.4	74.0	94.7	88.2	60.2	89.5	84.5
RoBERTa	87.6	92.8	**91.9**	78.7	94.8	90.2	63.6	**91.2**	86.4
MPNet	**88.5**	**93.3**	**91.9**	**85.8**	**95.5**	**91.8**	**65.0**	91.1	**87.9**
Single model on test set									
BERT	84.6	90.5	89.2	66.4	93.5	84.8	52.1	87.1	79.9
ELECTRA	**88.5**	**93.1**	89.5	75.2	**96.0**	88.1	**64.6**	**91.0**	85.8
MPNet	**88.5**	**93.1**	**89.9**	**81.0**	**96.0**	**89.1**	64.0	90.7	**86.5**

Table 20. Comparisons between MPNet and theMPNet and the previous strong pre-trained models under BERT$_{\text{BASE}}$ setting on the dev and test set of GLUE tasks. STS is reported by Pearman correlation, CoLa is reported by Matthew's correlation, and other tasks are reported by accuracy.

5.5.1.The Deep Neural Network Used

The deep neural network used to classify correctly the pairs is made of multiple dense layers with an activation ReLu activation function, terminating with a single unit with a sigmoid activation function as previously done in Chapter 3. The structure of the network is shown in detail in Figure 30.



input_1 | InputLayer | input: [(None, 1536)] | output: [(None, 1536)]
Dense1408 | Dense | input: (None, 1536) | output: (None, 1408)
Dense1280 | Dense | input: (None, 1408) | output: (None, 1280)
Dense1152 | Dense | input: (None, 1280) | output: (None, 1152)
Dense1024 | Dense | input: (None, 1152) | output: (None, 1024)
Dense896 | Dense | input: (None, 1024) | output: (None, 896)
Dense768 | Dense | input: (None, 896) | output: (None, 768)
Dense640 | Dense | input: (None, 768) | output: (None, 640)
Dense512 | Dense | input: (None, 640) | output: (None, 512)
Dense384 | Dense | input: (None, 512) | output: (None, 384)
Dense256 | Dense | input: (None, 384) | output: (None, 256)
Dense128 | Dense | input: (None, 256) | output: (None, 128)
Dense64 | Dense | input: (None, 128) | output: (None, 64)
Dense32 | Dense | input: (None, 64) | output: (None, 32)
Dense16 | Dense | input: (None, 32) | output: (None, 16)
prediction | Dense | input: (None, 16) | output: (None, 1)

Figure 30. Deep Neural Network used for all-mpnet-base-v2 features

5.5.2.Experiments Results

Due to the high size of the features extracted from this model, data augmentation was not possible for lack of Colab's resources. The experiment results are:

Experiment	**Training Loss**	**Training Accuracy**	**Validation Loss**	**Validation Accuracy**	**Precision**		**Recall**		**Macro F1-Score**
					Not duplicates	*Duplicates*	*Not duplicates*	*Duplicates*	
Basic	0.3037	0.8621	0.3272	**0.8596**	0.8860	**0.8053**	**0.8837**	0.8089	0.8460
Class weight	**0.2985**	**0.8623**	**0.3238**	0.8535	**0.9268**	0.7583	0.8313	**0.8897**	**0.8476**

Table 21. Experiment results for all-mpnet-base-v2 features

The results obtained from both experiments are very difficult to compare, with no experiment dominating over the other with performance. Precisely for this reason we have chosen to document here only the one with the class weights, since it has a greater number of better results than the other cases.

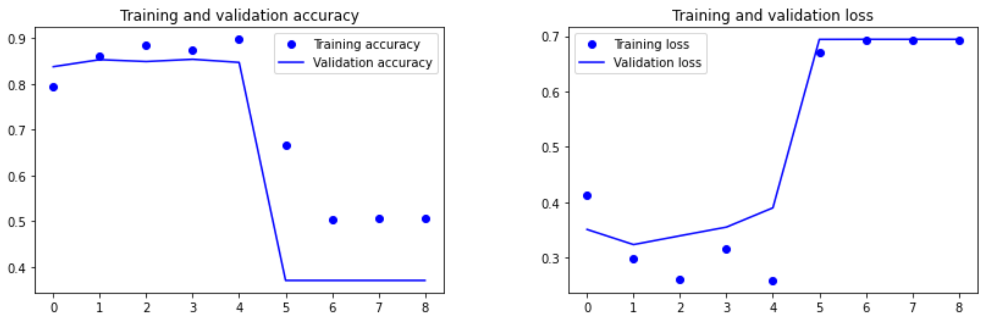


Figure 31. Learning curves for "class weight + augmentation" experiment (all-mpnet-base-v2)

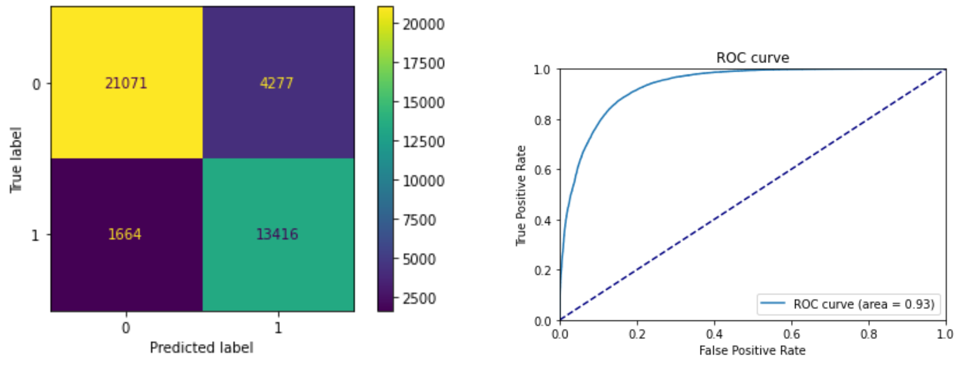


Figure 32. Confusion matrix and ROC curve for "class weight + augmentation" experiment (all-mpnet-base-v2)

As suggested by the macro-F1 score, the results obtained are slightly lower than those of the previous model, which was 0.8561. In fact, although this model tends to guess many more “duplicates” samples it also guesses many that should not be.

5.6. All-distilroberta-v1

The distilRoBERTta model is a distilled version of the RoBERTa-base model. Specifically, the all-distilRoBERTa-v1 is based on distilRoBERTa and fine-tuned in on a 1B sentence pairs dataset, using a contrastive learning objective.

Due to the high size of the features extracted from this model, data augmentation was not possible for lack of Colab’s resources.

5.6.1.The Deep Neural Networks Used

Due to the good results obtained from the features of all-distilRoBERTa-v1, as the next chapter underlines, multiple neural network models were used in order to find an even better result. From those models, an ensemble was constructed to strengthen the predictions and obtaining a more robust model. This ensemble was obtaining using a genetic algorithm, which is discussed in “Appendix A: Creating DistilRoBERTa ensemble”.

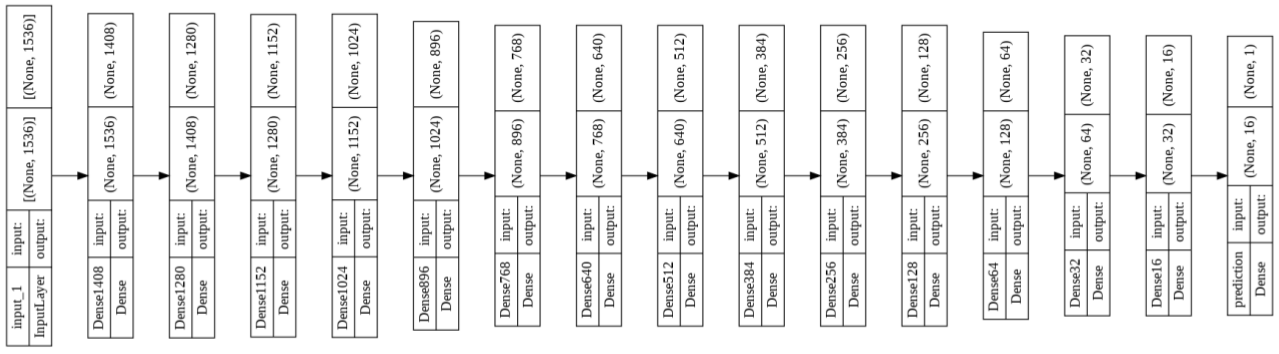


Figure 33. Neural Network with pace 128

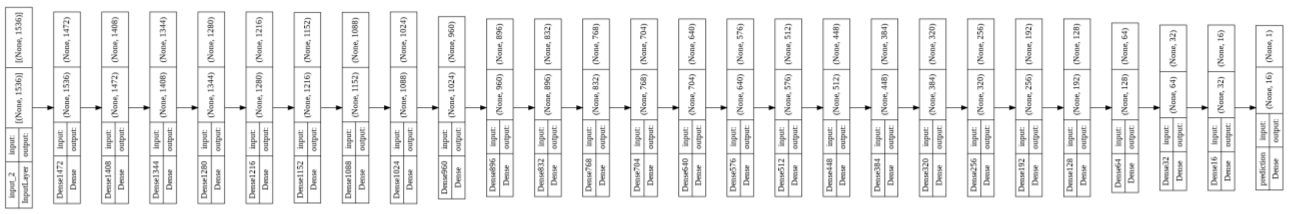


Figure 34. Neural Network with pace 64

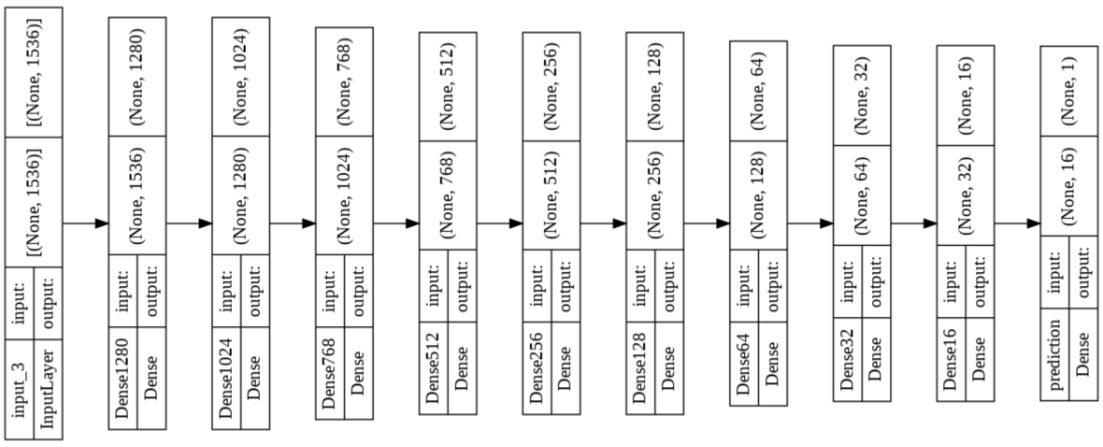


Figure 35. Model: Neural Network with pace 256

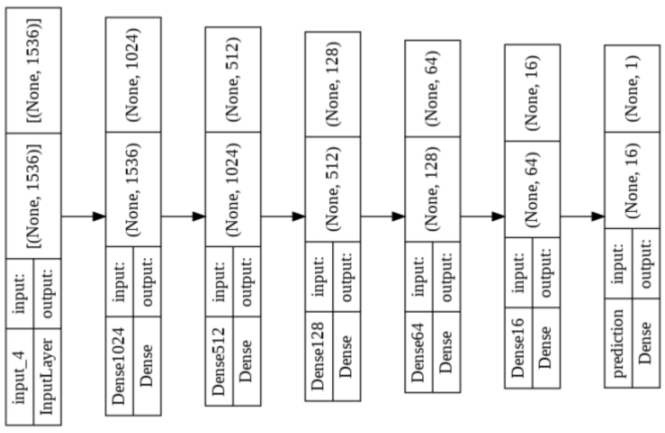


Figure 36. Model: Neural Network with pace 512

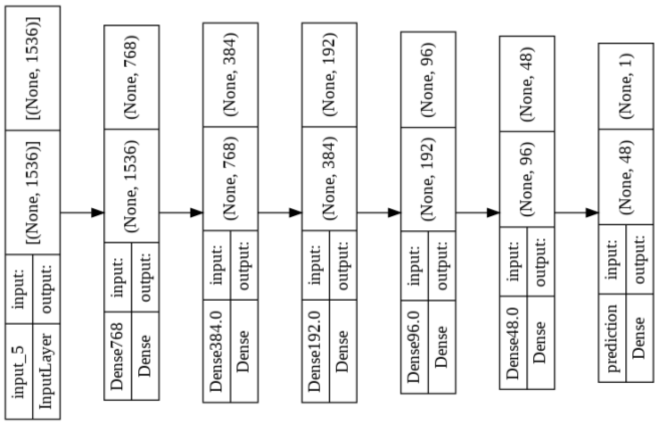

input_5 | InputLayer | input: | output: | [(None, 1536)] | [(None, 1536)]
Dense768 | Dense | input: | output: | (None, 1536) | (None, 768)
Dense384.0 | Dense | input: | output: | (None, 768) | (None, 384)
Dense192.0 | Dense | input: | output: | (None, 384) | (None, 192)
Dense96.0 | Dense | input: | output: | (None, 192) | (None, 96)
Dense48.0 | Dense | input: | output: | (None, 96) | (None, 48)
prediction | Dense | input: | output: | (None, 48) | (None, 1)

Figure 37. Model: Neural Network with halving by 2

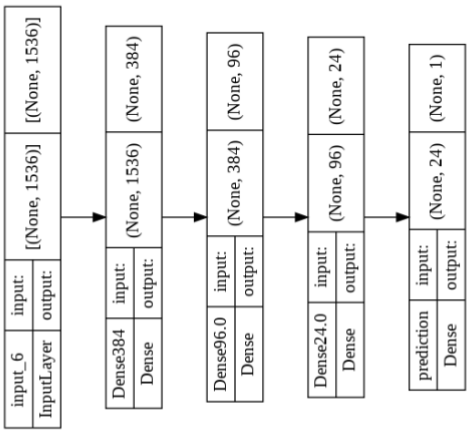

input_6 | InputLayer | input: | output: | [(None, 1536)] | [(None, 1536)]
Dense384 | Dense | input: | output: | (None, 1536) | (None, 384)
Dense96.0 | Dense | input: | output: | (None, 384) | (None, 96)
Dense24.0 | Dense | input: | output: | (None, 96) | (None, 24)
prediction | Dense | input: | output: | (None, 24) | (None, 1)

Figure 38. Model: Neural Network with halving by 4

5.6.2. Experiments Results

Due to the high size of the features extracted from this model, data augmentation was not possible for lack of Colab's resources. The experiment results are:

Model	Experiment	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy	Precision		Recall		Macro F1-Score
						Not duplicates	*Duplicates*	*Not duplicates*	*Duplicates*	
Pace128	Basic	0.2970	0.8658	0.3222	0.8602	0.8933	0.7963	0.8745	0.8244	0.8469
Pace128	Class weight	0.2553	0.8819	0.3296	0.8597	**0.9157**	0.7767	0.8515	**0.8682**	0.8511
Pace64	Basic	0.2867	0.8736	0.3425	0.8492	0.8842	0.7874	0.8701	0.8084	0.8374
Pace256	Basic	0.2827	0.8736	0.3267	0.8584	0.8776	0.8173	0.8949	0.7909	0.8449
Pace256	Class weight	0.2353	0.8925	0.3291	0.8592	0.9023	0.7926	0.8689	0.8418	0.8509
Pace512	Basic	0.2745	0.8774	0.3214	0.8568	0.8775	0.8231	**0.8991**	0.7889	0.8469
Pace512	Class weight	0.2814	0.8682	0.3231	0.8585	0.9136	0.7732	0.8490	0.8651	0.8484
Halving2	Basic	0.2769	0.8758	0.3195	0.8576	0.8767	**0.8228**	**0.8991**	0.7875	0.8568
Halving2	Class weight	0.2831	0.8677	0.3208	0.8584	0.9040	0.7948	0.8702	0.8447	0.8615
Halving4	Basic	0.2830	0.8738	0.3185	0.8588	0.8717	0.8209	0.8991	0.7777	0.8420
Halving4	Class weight	0.2881	0.8668	0.3251	0.8563	0.9142	0.7713	0.8472	0.8664	0.8478
Ensemble	Gene2	**0.2004**	**0.9229**	**0.2858**	**0.8734**	0.9035	0.8168	0.8878	0.8406	**0.8621**

Table 22: Experiment results for all-distilroberta-v1 features

Again, the results obtained from both experiments are very difficult to compare, with no experiment dominating over the other with performance. Therefore, to choose the best model some considerations were made:

1. *Training metrics*: for both accuracy and loss the best values were obtained by the ensemble, which also obtained the best values for the entire transfer learning analysis.
2. *Validation metrics*: the best values are obtained from the ensemble; this was guaranteed since the ensemble was built with the goal of maximizing the validation accuracy as described in "Appendix A: Creating DistilRoBERTa ensemble".
3. *Precision*: the best precision values were obtained by pace128-class-weight for the "not duplicates" class (0.9157) and halving2-basic for duplicates (0.8228). Although these two models have reached the best scores, they present some problems. Pace128-class-weight, in particular, obtains a very low score on "duplicates" (0.7767), resulting unbalanced; halving2-basic, instead, is much more balanced between the two classes performing well, so it turns to be better than the other model. If we consider this last model and we compare it with the ensemble, we can see that even if the ensemble has a slightly lower result for the class "duplicates", it has a discreetly higher value for the class "not duplicates". This makes the ensemble, from a general point of view, even better than halving2-basic.
4. *Recall*: for the recall it is not possible to define a model better than the others, but some considerations have been made and they are the following. The best recall for the "not duplicates" class was achieved by two models, halving2-basic and pace512-basic with a value of 0.8991. Among the two models, the one that is victorious is pace512-basic that obtains a recall value on "duplicates" a bit higher than halving2-basic. Even so, the value on duplicates is very low compared to the general average. Instead, the model with better recall on duplicates, pace128-class-weight, performs better than pace512-basic, having high values for both fields. However, considering both fields ("not duplicates" and "duplicate"), it can be seen that many models have high values that are fairly balanced with each other, including pace256-class-weight, halving2-class-weight, halving4-class-weight and the ensemble. Anyway, if we deem the average between the two fields, the greater score is achieved by pace128-class-weight.
5. *Macro F1-score*: the ensemble is the one with the highest score, also it is the highest score achieved so far.

Thanks to these considerations, the ensemble was chosen as the best model for distilRoBERTa since it performs better in general. The confusion matrix and the ROC curve for the ensemble are depicted in Figure 39.

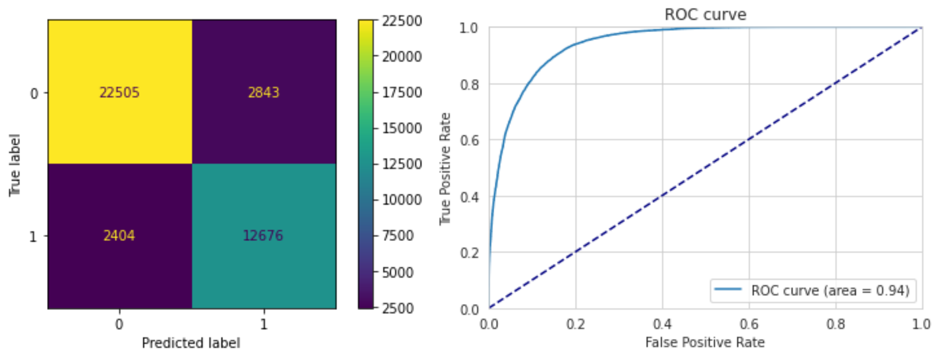


Figure 39. Confusion matrix and ROC curve for "ensemble" experiment (all-distilroberta-v1)

Comparing this confusion matrix to *all-miniLM-L12-v2*'s, which was our previous best, we can see that we get a more balanced situation. This is achieved at a small cost, the number of true positives ("duplicates") is slightly less (previously it was 12,814), but the number of true negatives ("not duplicates") has increased by 401 leading to this improvement.

6. ENSEMBLE

6.1. Why we need an Ensemble

In order to evaluate the convenience or not of resorting to ensemble solutions, an analysis of the errors committed by the models with the best performances has been carried out. Since the diversity of the selected set of classifiers is key to the proper functioning of ensembling, it was important to analyse the errors to understand whether they were common to all the classifiers or whether each classifier had different predictive power despite having similar performance.

First of all, we noticed that increasing the number of classifiers based on different architectures we get better and better results. For each architecture, the model with best performance was considered.

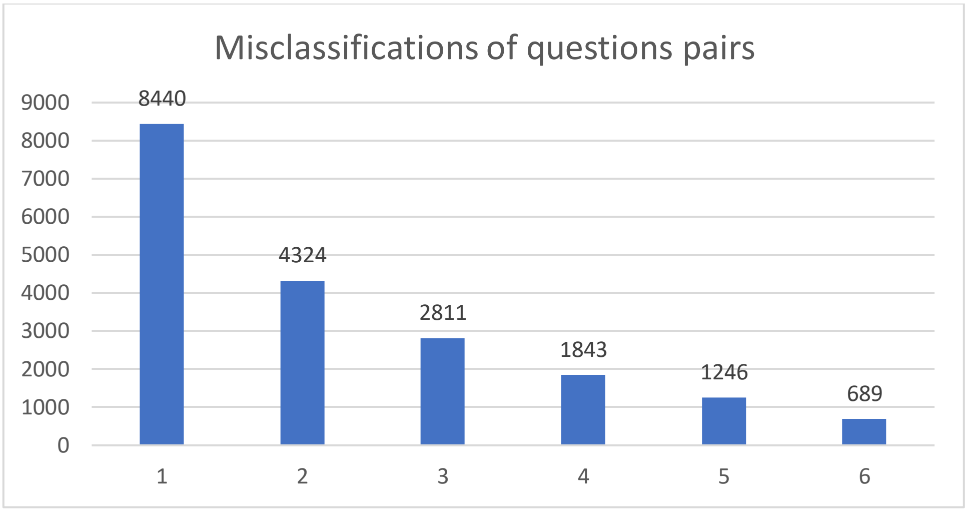


Figure 40. Misclassifications on the test set of the best models

Going from 1 classifier to 6 classifiers we get a 92% reduction in misclassifications. Obviously, we will not use all the models for the ensemble, but we will try various combinations in order to obtain the best result, as we will see later. For now, we can focus on understanding how many unique pairs are well classified by each architecture, and how many unique mistakes are made by each of them.

	Doc2Vec	**MiniLM-L12-v2**	**All-mpnet-base-v2**	**Distil-Roberta**	**Siamese**	**LSTM basic**
Unique misclassifications	3838	614	575	596	1123	1694
Unique correct classifications	380	155	160	182	171	198

Table 23. Unique classifications and misclassifications

6.2. The Ensemble and its Results

The ensemble was constructed as a *weighted average* ensemble: each model is assigned a fixed weight that is multiplied by the prediction made by the model and used in the average prediction calculation. Those weights were found using a genetic algorithm which is described in detail in "Appendix B: Creating the final ensemble", in this chapter only the final results are presented since the GA is not the objective of this project. Also, to improve our classification, the threshold for distinguishing between "duplicates" and "not duplicates" in the prediction was also computed through the genetic algorithm[2].

The weights associated to each model are:

Model	**Weight**	**Normalized weights[3]**
DistilRoBERTa-ensemble	0.9800993386369313	0.29123438015439385
Word2Vec Siamese	0.9514201675710127	0.28271242704264027
doc2vec-pace16 model	0.015659752451383548	0.004653261275426072
miniLM-pace96-aug-weighted	0.7612600770183728	0.2262067710147293
mpnet-pace128-weighted	0.6568890919528059	0.1951931605128106

Table 24. Weights associated to the models in the weighted average ensemble

And the threshold value found was 0.36785, thus predictions above this value were consider "duplicates".

The weights point out that the most important models are distilRoBERTa and the Siamese. On the other hand, the Doc2Vec-pace16 model's predictions were barely considered as the extremely low weight suggests.

The overall results are the following:

Validation Accuracy	**Macro F1-score (val)[4]**	**Precision**		**Recall**		**Macro F1-Score**
		Not duplicates	*Duplicates*	*Not duplicates*	*Duplicates*	
0.8924	0.8858	0.9220	0.8346	0.8971	0.8725	0.8813

Table 25. Results for ensemble

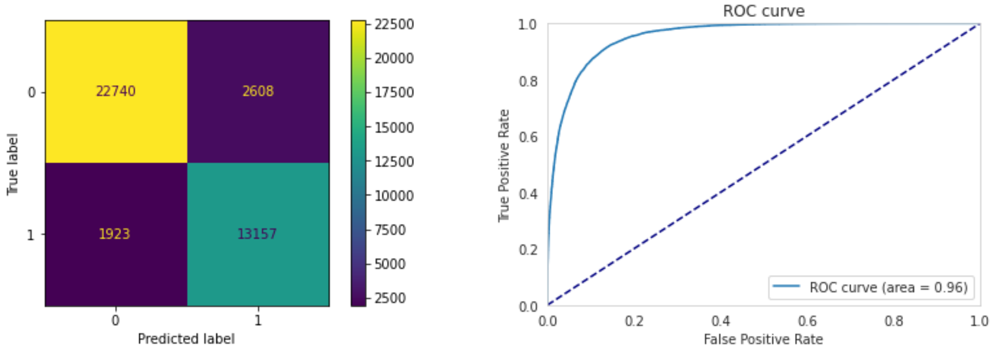


Figure 41. Confusion matrix and ROC curve for the ensemble (test set)

The obtained validation accuracy is the highest obtained in the whole project. Likewise, the precision and recalls values are better than the previous best, achieved by distilRoBERTa ensemble. This led to achieve a better macro F1-score which is 0.02 points greater than the one obtained with distilRoBERTa ensemble, making this final ensemble our best model.

[2] In the previous chapters this threshold was always set to 0.5, thus above it the prediction is turned to 1 ("duplicates"), otherwise 0 ("not duplicates")
[3] Between zero and one
[4] This parameter was the one optimized by the genetic algorithm

7. ERROR ANALYSIS

In this chapter, we will focus on analysing the errors made by the model that gave us the best results, namely the ensemble defined in the previous section. Error analysis is always important because it allows to highlight those that are the strong points and the weak points of the model. Moreover, it allows us to make hypotheses on the reason on why our model makes mistakes, leaving therefore space to possible improvements.

It is important to note that the task of assessing the similarity between two sentences is challenging even for humans and there is no absolute truth. Indeed, there will always be some degree of subjectivity in telling whether two questions are duplicates or not. This must also be taken into account with regard to the labels that were provided for the dataset, which reflect the annotator's thinking and thus may be partially incorrect in our eyes.

The fundamental goal of our error analysis is, for this reason, to assess whether the model made "justifiable" errors, i.e., attributable to the inherent subjectivity of the task, or "true" errors of misinterpretation of the semantics of the text.

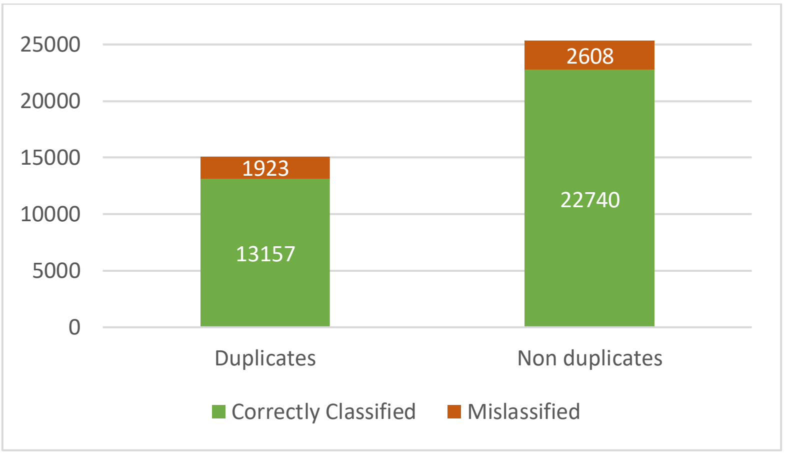


Figure 42. Classifications made by the best model, divided by class

Figure 42 shows the correct and incorrect classifications made by our model. One interesting fact we were able to observe is that with 4531 misclassified pairs we would potentially have 9062 different questions, but in reality we only have 4499 unique ones. Specifically, 32 questions are repeated four times and the rest (4467) are repeated two times.

This means that the number of questions that the model was not able to learn well is more limited, and thus the mistakes made are repeated. However, this does not allow us to understand what is misclassified and why.

In order to better understand the errors made, we randomly isolated 30 pairs of questions belonging to the "duplicates" class and 40 pairs of questions belonging to the "not duplicates" class. In the following sections, we will focus on some of these pairs of questions.

7.1. "Duplicates" class

As we have seen during almost all experiments, the class "duplicates" is the one with which our models have always had the most difficulty. This is probably due to the smaller number of examples of that class in the training set. Even using different strategies, we could not fully overcome this imbalance.

Table 26 shows some examples of questions that were labelled as duplicate, but were not classified as such.

1	What are some good ideas for website?	What is a good idea for a website?
2	What are the best products to Dropship?	What are the best products for dropshipping?
3	When should "per se" be said?	What does "per se" mean?
4	What is a positive ion called?	What is a positive ion?
5	How can I watch the Olympics live via computer?	How can I watch Rio Olympics Online Live?
6	Why did the United States buy Alaska in 1868?	Why did the US buy Alaska from Russia?

Table 26. Examples of question pairs labelled as duplicates that were not classified as such

Out of the question pairs shown in the table, we believe #1 and #2 are the ones for which we can most confidently assert similarity. As a matter of fact, as far as #1 is concerned, the difference between singular and plural (“idea” vs “ideas”) leaves the meaning completely unchanged. Similarly, in pair #2, even though infinitive of purpose is used once and ‘for + verb-ing’ is used in the other, the intended meaning is exactly the same.

The actual being duplicates of the next pairs of questions #3, #4 and #5, in our opinion is questionable. In fact, while there is some degree of similarity we do not feel confident in stating this as strongly as for previous pairs. In pair #3, in our opinion, the two questions differ slightly in intention. Specifically, while the first question is about the usage of the expression “per se”, the second one is about its meaning. However, we do acknowledge that often the meaning and usage examples of terms are given together, as in dictionaries for instance.

As for pair #4, the author of the first question may have been interested only in the name of the positive ion, i.e., cation, and not in an explanation of what it is. In a similar way, the author of the first question in pair #5 is talking about the Olympics in general, which is not necessarily the Rio Olympics in the absence of any other information.

Regarding pair #6, if it is true that the two questions refer to the same event, the model has little way of knowing this from the semantics of the questions alone. In fact, even though the ultimate intent of the questions is to know why the U.S. bought Alaska, one question refers to the year and the other to the country from which it was purchased.

7.2. “Not duplicates” class

The “not duplicates” class is the one that our models seemed to recognized best. However, also in this case errors were made and some examples of them are shown in Table 27.

As in the previous case, we can identify more or less severe misclassifications, in the sense of whether or not it is evident that the questions in the pair are not duplicates.

The first pairs of questions, from #1 to #5, are similar but not duplicates. In fact, although the context is almost the same, there are a few more words that make one of the two questions more specific than the other. For example, in pairs #1 and #2 they mention “India” in only one of the two questions; what applies in general may not be good for the specific case of that country. Also, in the first question of pair #3 they specify “in men” and in the second question in pair #4 they emphasize “for free”. Regarding pair #5, in the second question they mention “a few days”; the more general answer may not fit the author’s short trip.

In the next pairs of questions, #6, #7, and #8, the problem seems to be not the specificity of the question, but really the meaning. For example, in pair #6, “fastest” and “most effective” cannot be considered synonymous; rather, there is often some compromise to be made between the two. Also in pair #7 the intended meaning is different, on one side we have "in World War II" and on the other side "of all time". Finally, in couple #8, the “tips” for losing weight in a month don't necessarily cover just “exercise regimens”, they could also include nutrition and other activities.

Pairs #9 and #10, on the other hand, in our opinion are indeed duplicates even though they have not been labelled as such. In fact, the two question in pair #9 express exactly the same intent, albeit with a few tiny differences. With regard to pair #10, “agriculture” and “farming” can be considered synonymous.

1	How do I start YouTube channel to earn money?	How do I start YouTube channel and earn money in India?
2	Which are the best IIT JEE coaching classes in India?	Which are the best online IIT JEE coaching?
3	What causes hair loss for men?	What are the causes of hair loss?
4	How do you get a book published?	How do I get a book published for free?
5	What are the best things to do in Toronto?	What are the best things to see and do in Toronto for a few days?
6	What's the fastest way to learn Chinese characters?	What is the most effective way to learn Chinese characters?
7	Was General George S. Patton the best general of the World War II?	Was General George Patton the greatest general of all time?
8	What are some good exercise regimes for a losing 30 pounds in 30 days?	How can I lose 30 pounds in 1 month? What are some tips?
9	How do I convert a PDF to a Word document?	How can I convert a PDF to Word?
10	What is regenerative agriculture?	What is regenerative farming?

Table 27. Examples of question pairs labelled as non-duplicates, but which have been classified as such

In addition, one thing we were able to notice when analysing the misclassified question pairs is that there were also some mislabelled ones. For instance, Table 28 shows pairs containing almost the exact same questions that were, however, labelled as “not duplicates”. This leads us to think that there may be other labelling errors and thus the results obtained from our model may have been negatively affected.

1	How do I get in Harvard?	How do I get into Harvard?
2	What is the importance of money in our life?	What is the importance of money in your life?

Table 28. Mislabelled question pairs

8. CONCLUSIONS

In this work we tested three different approaches to solve the task of assessing the similarity of question pairs, taking advantage of both from-scratch, Siamese, and pre-trained architectures. Then we proposed an ensemble solution able to combine the classification power of the best models found. The results obtained are encouraging and show that it is possible to correctly classify almost all questions pairs, with a macro-F1 score of 88.13%.

As already mentioned, we haven't used the whole training set, but from the original training set we obtained also the validation set and the test set. This implies that the results are not directly comparable to those of the competition. In any case, the results are very good considering the difficulty of the task.

Unfortunately, due to computational limitations of Colab it was not possible to always test the performance with fine tuning. So, there is still room to improve the performance of our networks. It would be interesting to perform an hyperparameter optimization directly on the components of the models, to optimize the performance of the individual neural networks that make up the final ensemble.

9. REFERENCES

[1] Quora, [Online]. Available: https://www.quora.com.

[2] Quora, “Quora Question Pairs,” 2017. [Online]. Available: https://www.kaggle.com/c/quora-question-pairs. [Accessed February 2022].

[3] J. Mueller and A. Thyagarajan, “Siamese Recurrent Architectures for Learning Sentence Similarity,” in *AAAI*, 2016.

[4] J. Pennington, R. Socher and C. D. Manning, “GloVe: Global Vectors for Word Representation,” in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532-1543.

[5] T. Mikolov, K. Chen, G. Corrado and J. Dean, “Efficient Estimation of Word Representations in Vector Space,” 2013.

[6] N. Reimers and I. Gurevych, “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks,” in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, 2019.

[7] Q. V. Le and T. Mikolov, “Distributed Representations of Sentences and Documents,” 2014.

[8] R. Řehůřek and P. Sojka, “Software Framework for Topic Modelling with Large Corpora,” in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, ELRA, 2010, pp. 45-50.

[9] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang and M. Zhou, “MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 5776-5788, 2020.

[10] K. Song, X. Tan, T. Qin, J. Lu and T.-Y. Liu, “MPNet: Masked and Permuted Pre-training for Language Understanding,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 16857-16867, 2020.

[11] K. Deb and R. B. Agrawal, “Simulated Binary Crossover for Continuous Search Space,” *Complex Systems*, vol. 9, no. 2, pp. 115-148, 1995.

10. APPENDIX A: CREATING DISTILROBERTA ENSEMBLE

To improve and robust the results obtained using all-distilRoBERTa-v1, we created an ensemble based on *average voting*. The main problem was to choose which of the models, constructed on all-distilRoBERTa-v1's features, combined together obtained the best accuracy. Since the number of models was 11, grid search seemed too slow as a method of solving this problem, so we decided to use a genetic algorithm to solve the problem automatically.

10.1. Genetic Algorithm Components

The genetic algorithm workflow was implemented using the *Deap* framework in python. To this aim, we needed few preliminary components to decide in order to make our GA works.

A genetic algorithm needs few components:

- *Genotype*: in our specific case, our chromosomes are binary-encoded, and each gene represents a different model. Hence, we have individuals made of a number of genes equal to the number of different models that we have (i.e., the one presented in Table 22). If a gene is set to 1, it means that the model is consider for the ensemble.
- *Population*: we set the number of the population to 100 individuals.
- *Fitness Function*: the fitness value is obtained as the accuracy of the ensemble, given the individual and thus the models considered.
- *Selection Algorithm*: the selection algorithm used in this case is tournament selection. In each round of the tournament selection method, two individuals are randomly picked from the population, and the one with the highest fitness scores wins and gets selected.
- *Crossover Algorithm*: as crossover algorithm we decided to use the two-point crossover. In the two-point crossover method, two crossover points on the chromosomes of both parents are selected randomly. The genes residing between these points are swapped between the two parent chromosomes.

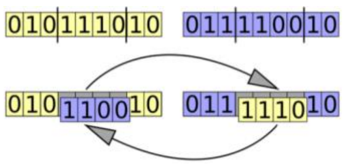


Figure 43. Two-point crossover example

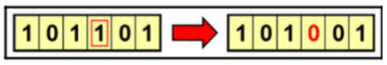


Figure 44. Flip bit mutation example

- *Mutation Algorithm*: as mutation algorithm we decided to use the multiple flip bit mutation. When applying the flip bit mutation to a binary chromosome, one gene is randomly selected and its value is flipped (complemented), as shown in Figure 44.
- *Elitism*: we set the number of individuals for the elitism mechanism to 5.

10.2. Genetic Algorithm Results

The results obtained from the genetic algorithm are reported in Table 29.

Gene	Precision		Recall		Macro F1-Score
	Not duplicates	*Duplicates*	*Not duplicates*	*Duplicates*	
0=[1,1,1,0,0,1,0,1,0,0,1]	0.9025	0.8162	0.8876	0.8387	0.8611
1=[1,1,1,0,1,1,0,1,0,1,1]	**0.9054**	0.8136	0.8849	**0.8446**	0.8619
2=[1,1,1,1,1,1,0,1,1,0,1]	0.9035	0.8168	0.8878	0.8406	**0.8621**
3=[1,0,1,1,1,1,0,1,0,0,1]	0.9031	**0.8174**	**0.8884**	0.8399	**0.8621**
4=[1,1,1,1,1,1,0,1,0,0,1]	0.9053	0.8134	0.8848	0.8444	0.8617

Table 29. Distilroberta GA results

Gene 2 was chosen as the best model, and it corresponds to the following models: *halving2*, *halving2_weighted*, *halving4*, *halving4_weighted*, *pace128*, *pace128_weighted*, *pace256_weighted*, *pace512*, *pace64*.

11. APPENDIX B: CREATING THE FINAL ENSEMBLE

In this appendix, the steps needed to obtain the weights and the threshold to be used in the weighted average ensemble are here discussed.

11.1. Genetic Algorithm Components

The genetic algorithm workflow was implemented using the *Deap* framework in python. To this aim, we needed few preliminary components to decide in order to make our GA works. A genetic algorithm needs few components:

- *Genotype*: in our specific case, our chromosomes are real-encoded and have values between 0 and 1. The number of genes is equal to the number of models[1] used plus a gene for the threshold, which decreets if a prediction is a 1 or a 0. The gene-information correspondence is as follows: distilroberta ensemble, Word2Vec Siamese model, doc2vec-pace16 model, miniLM-pace96-aug-weighted model, mpnet-pace128-weighted model, threshold.
- *Population*: we set the number of the population to 200 individuals.
- *Fitness Function*: the fitness value is obtained as the macro-f1 of the ensemble given the individual.
- *Selection Algorithm*: the selection algorithm used in this case is tournament selection. In each round of the tournament selection method, two individuals are randomly picked from the population, and the one with the highest fitness scores wins and gets selected.
- *Crossover Algorithm*: as crossover algorithm we decided to use the *bounded simulated binary crossover*, which is a bounded version of the *simulated binary crossover (SBX)* [11]. The idea behind the simulated SBX is to imitate the properties of the single-point crossover that is commonly used with binary-coded chromosomes.
- *Mutation Algorithm*: as mutation algorithm we decided to use the *bounded polynomial mutation*, which is a bounded mutation operator that uses a polynomial function for the probability distribution.
- *Elitism*: we set the number of individuals for the elitism mechanism to 5.

[1] The models are the best obtained for each model category in the Transfer Learning Chapter and the best model obtained in the Siamese analysis.

11.2. Genetic Algorithm Results

The results obtained from the genetic algorithm are reported in Table 30. Gene 2 was chosen as the best model.

Gene	Precision		Recall		Macro F1-Score
	Not duplicates	*Duplicates*	*Not duplicates*	*Duplicates*	
0=[0.9816009254735635, 0.9498029485579088, 0.020774320450703807, 0.7459131369333925, 0.6162475176147064, 0.3644184517582807]	0.9205	0.8362	0.8987	**0.8696**	0.8810
1=[0.9816009254735635, 0.9498069659758934, 0.020774320450703807, 0.7459131369333925, 0.6162475176147064, 0.3644184517582807]	0.9205	0.8362	0.8987	**0.8696**	0.8810
2=[0.9800993386369313, 0.9514201675710127, 0.015659752451383548, 0.7612600770183728, 0.6568890919528059, 0.3678491341881324]	**0.9220**	0.8346	0.8971	0.8525	**0.8813**
3=[0.9816009254735635, 0.9498069659758934, 0.01930153596534729, 0.7459131369333925, 0.6162475176147064, 0.3644184517582807]	0.9205	**0.8364**	**0.8988**	0.8695	0.8811
4=[0.9816009254735635, 0.9498069659758934, 0.019114565669288805, 0.7459131369333925, 0.6162475176147064, 0.3644184517582807]	0.9205	**0.8364**	**0.8988**	0.8694	0.8810

Table 30. GA results