University of Pisa
Computer Science
Master Degree Artificial Intelligence

Human Language Technologies

Academic Year: 2023/2024
Date: 03/10/2023

Summary evaluation through Features Extraction and Large Language Models.

Authors
Giacomo Carfì, Christian Peluso, Alessandro Sardelli
g.carfi1@studenti.unipi.it, c.peluso5@studenti.unipi.it, a.sardelli3@studenti.unipi.it

Abstract

Thanks to state-of-the-art models and more insights we have accomplished great results challenging the **CommonLit Competition** on Kaggle [1]. In this report, we will explain our automatic routine useful when teachers and tutors want to evaluate students' summaries. Specifically, we will investigate how we have decomposed the task and what were the resulting conclusions so that we can make an assessment that is as complete as possible, taking into consideration semantics, grammar, paragraphs, coherence, context, and other important features of the elaborate. In order to achieve our intentions we will introduce the technologies used and the way they were chained together starting from a simple **Recurrent Neural Network**, then we move to use pre-trained Large Language Models like **BERT** and **DeBERTa**. We show that combining LLMs embeddings and features extracted from text data works to solve our problem and that we reach comparable results in line with the ones that outperformed the competition leaderboard.

Contents

1 Introduction

In recent years, pre-trained large language models (LLMs), like **BERT** [2] and **GPTs** [3], have shown remarkable performance on many natural language processing (NLP) tasks, such as text classification, question answering, and summarization, becoming the foundation of modern NLP systems. By performing self-supervised learning, such as masked language modeling, LLMs learn to encode various knowledge from text corpora and produce informative representations for downstream tasks.

In this report, our team will present the work done to participate in the CommonLit Challenge, a competition that requires automatic evaluation of students' summaries to aid teachers with their work.

In our project, first, we used **Recurrent Neural Network** to establish a baseline, then we tried to use **LLMs** to automatically evaluate human-created summaries, so that, given a chapter to be processed and a specific outline to be adhered to, a continuous score is assigned to them that affirms how good the summary is. So in this case, summaries evaluation is a regression task. It is challenging because, given a track to follow, it is arduous for a machine to understand the context, especially when there is a limit on the number of tokens digested at once, indeed very long sentences will be truncated. Automatic extraction, by using classical methods, of information from the text therefore becomes crucial, since this can be combined with the embedding from the LLM in order to increase the whole performance.

This report is structured as follows: in Section 2 we provide information on the competition, in Section 3 we talk about how we analyzed and engineered the data available, in Section 4 we explain the models used, in Section 5 we will clarify the experiments performed, including ablation tests that we performed to identify most useful features. Finally, in Section 6 we discuss the results obtained, and in Section 7 we will indicate possible future improvements and our conclusions. The code of our project can be found in [4].

2 CommonLit Challenge

Within this particular section, we aim to provide you with comprehensive insights sourced directly from Kaggle's web page, shedding light on the essence of the challenge at hand.

The primary objective of this competition is to evaluate summaries written by students spanning from grades 3 to 12. The main challenge lies in the model's ability to analyze how cleverly a student is able to extract the principal ideas and concepts found within a given source text. Furthermore, the model is entrusted with the responsibility of grading the clarity, precision, and completeness of the summaries produced.

The challenge is promoted by CommonLit, a nonprofit organization firmly rooted in the realm of educational technology.

In order to *submit* our results to the competition we had to meet some specific requirements:

- CPU or GPU Notebook $\leq$ 9 hours run-time;
- Internet access disabled;
- Freely & publicly available external data is allowed, including pre-trained models;
- All the libraries must be under MIT license so freely usable under commercial use;

The competition offers two prizes, the first prize goes to the model that obtains the best score in terms of loss. There is also a strand branching out from the main competition, that rewards efficient notebooks that are not using GPU. This section of the competition takes into consideration the running time together with predictive performance. This branch is hosted because highly accurate models are often computationally heavy and frequently prove difficult to utilize in real-world educational contexts.

Since the metrics used to evaluate the elaborates are multiple, the competition requires minimizing a loss computed on the averaged Root Mean Squared Error (RMSE) for both the Content and Wording scores, becoming the so-called Mean Column-wise RMSE (MCRMSE).

$$\text{MCRMSE} = \frac{1}{m}\sum_{j=1}^{m}\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_{ij}-\hat{y}_{ij})^2}$$

where y_i and $\hat{y}_i$ are the actual and predicted values, respectively, m is the number of columns and n is the number of samples. Instead for the efficiency price, the score is calculated:

$$\text{Efficiency} = \frac{\text{MCRMSE}}{\text{Base} - \min \text{MCRMSE}} + \frac{\text{RuntimeSeconds}}{32400}$$

Where MCRMSE is the submission's score on the main competition metric, Base is the score of the "baseline" submission of the mean of each target on the training set, $\min$ MCRMSE is the minimum MCRMSE of all submissions on the hidden test set Leaderboard, and RuntimeSeconds is the number of seconds it takes for the submission to be evaluated. The objective is to minimize the efficiency score.

3 Data Preparation

In order to meet the needs of the competition we had to explore the dataset and analyze how it is composed, specifically in Section 3.1 we will explain what the dataset consists of in detail, and in Section 3.2 we will explain what features we have extracted.

3.1 Dataset

The dataset consists of sections of books discussing a specific topic, a question related to this topic, and a summary representing the student's answer. Summaries are written by students attending schools from 3rd to 12th grade and topics can range different genres and subjects, and each of them has been evaluated with two scores, respectively for **content** and **wording**. The content score indicates how appropriate and coherent the text is in gathering the main ideas from the resource. The wording value, instead, indicates whether the language used is objective, appropriately paraphrased, and whether there has been a correct use of lexicon and syntax. The goal of the competition is to predict content and wording scores for summaries on **unseen topics**. In detail, the dataset provided consists of two `Comma Separated Value` files named respectively `summaries_train.csv` and `prompts_train.csv`. The first contains the summaries written by the students, and specifically, the columns are:

- *student_ id* - The ID of the student writer.
- *prompt_ id* - The ID of the prompt that links to the prompts file.
- *text* - The full text of the student's summary.
- *content* - The content score for the summary. The first target.
- *wording* - The wording score for the summary. The second target.

The second file contains the four training set prompts. Each prompt comprises the complete summary assignment given to students. In detail, there are:

- *prompt_ id* - The ID of the prompt that links to the summaries file.
- *prompt_ question* - The specific question the students are asked to respond to.
- *prompt_ title* - A short-hand title for the prompt.
- *prompt_ text* - The full prompt text.

The size of the training dataset provided to us is 7165 rows, while the full test set comprises about 17,000 summaries from a large number of prompts.

All the summaries, in the training dataset, refer to only 4 prompt IDs and they are even unbalanced, in fact, Table 1 shows the number of samples for each *prompt_ id*.

prompt_id	samples
39c16e	2057
3b9047	2009
ebad26	1996
814d6b	1103

Table 1: Number of samples for each prompt_id.

When dealing with massive datasets and text elaboration is important to see the length of the data we are working with, in such a way that we can prevent the limitations of the models we are going to use. Figure 1 shows a histogram of the number of segments of text generated after applying the tokenization on the *text* and *prompt_ text* columns. As we can see from the bars, we are facing very long tests since the number of tokens of the summaries goes from **26** to **856**, while the number of tokens of *prompt_ text* goes from **675** to **1202**. This is important because some models cannot handle such long inputs.

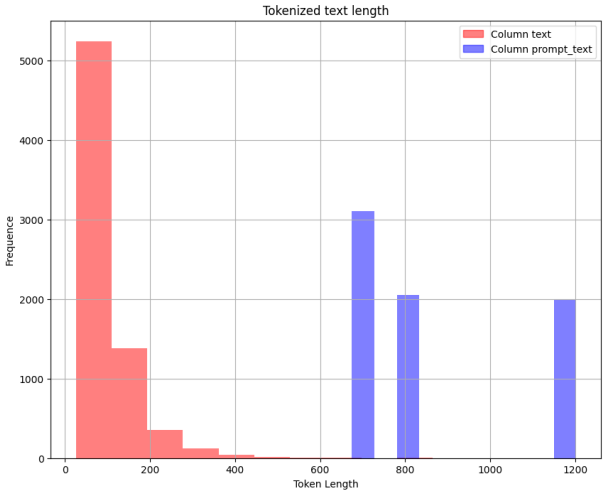


Figure 1: Histogram of the number of tokens for *text* and *prompt_ text*. We can see that there are very long prompt_text that can be over 1200 tokens.

3.2 Feature Engineering

Our initial approach was generating the predictions based on bare embeddings and a couple of layers fine-tuned specifically for the task. We observed that the single generation of context was not sufficient to achieve competitive results on the leaderboard. So, to ensure a well-designed architecture with all the necessary information, we decided to break down the task and analyze the structure of the prompts and summary texts, extracting features and combining them with the model's output.

In order to understand our process of thoughts here we will briefly explain what are the features and how they have been extracted.

- **Corrected-Text** Since the LLM are pre-trained on massive amounts of correct data, we assumed that in order to facilitate the assessment task and to better highlight the mistakes made by the students, it was necessary to correct the text. To do this expensive operation we used the Symmetric Delete Spelling Correction algorithm [5];
- **Edit-Distance** or Levenshtein distance, between two strings s1 and s2 is the minimum number of substitutions, deletions, or insertions that must be made to get s1 from s2, we got this metric for free from the application of the library above. This metric should aid to detect, analyzing strings that are not present in the vocabulary, how many errors the text contains in terms of wrong characters.
- **N-grams** are contiguous sequences of words that are collected from a sequence of text or speech corpus. The n in n-grams specifies the size of the number of words to consider. We extracted n-grams with n from 2 to 4 and we counted how many n-grams appear both in a summary and its relative prompt_text.
- **Named Entity Recognition** is an NLP method that extracts information from text. NER involves detecting and categorizing important information in text known as named entities. Named entities refer to the key subjects of a piece of text, such as names, locations, companies, events, and products, as well as themes, topics, times, monetary values, and percentages:
  - **Matches** all entities found in the summary were compared and counted against those in the *prompt_ text*;

- **TF-IDF Score** stands for Term Frequency, the frequency of the term in the document itself (more is better) on Inverse Document Frequency, a term which occurs in many documents is not a good discriminator, and should be given less weight (less is better). It is computed as following: $\text{idf}(t_i) = log\frac{N}{n_i}$ taken from [6]. It has been used in order to vectorize the terms appearing in each summary and then this array has been compressed into a score;
- **Part of Speech** (POS) tagging is a process where each word in a text is labeled with its corresponding part of speech. This can include nouns, verbs, adjectives, and other grammatical categories. It can be used to identify the grammatical structure of a sentence and to disambiguate words that have multiple meanings;
- **Text Length** of the summary written by the student (number of characters);
- **Text Length Ratio** of the summary prepared by the student with respect to *prompt_ text* length;
- **Different Words counter** is the sum of all the different words found in the summary;
- **Different Words ratio** of the the summary w.r.t. the number of different words in *prompt_ text*;
- **Misspelled word counter** is different from the edit distance cause takes into consideration all the errors in a word only once;
- **Number of words** is computed summing all the words, with repetition, found in the summary;
- **Words ratio** computed in accordance with the number of words in *prompt_ text*;
- **Stop-Words** we have listed all those common words that do not change the semantic of the text (like "of", "the", and so on), and from these, we have computed:
  - **Number** in the summary written by the student;
  - **Ratio** computed w.r.t. the number of stop-words in *prompt_ text*;
- **Punctuation** all characters that match a punctuation mark, have been listed and from those we have computed:
  - **Number** the sum of all of those inside the summary;
  - **Ratio** computed w.r.t. the number of punctuation characters in *prompt_ text*.

Further notes about the extracted features: the **TF-IDF score** was created taking into consideration the *prompt_ id*, in fact, it uniquely identifies the book chapter to be summarized, so it defines the difficulty of the text, in other words, the 'class' of the prompt_text. In the challenge information, it has not been mentioned, but we assumed that summaries from 3rd-grade students would be evaluated differently from summaries from 12th-grade students, professors would research more appropriate terms and more elaborate writing styles by a graduating respect a newbie. This distinction can highlight the words used in the summaries reflecting the abilities that the students acquire during their school years. So, first of all, the strings in *text, prompt_ text, prompt_ title, prompt_ question*, were made all lowercase, then they were preprocessed by removing unnecessary white-space, carriage return, punctuation, and stop-words. After that, we created four term-document matrices, one for each *prompt_ id*, where the terms are the words that appear on each student summary for that prompt, and the documents are all the student summaries plus the book chapter, the question, and the title. Each cell of a specific term-document matrix then contains the importance of a word w in the document d. Given a specific document, i.e. a specific student summary, the score is then computed following the intuition of Karp-Rabin Fingerprint $\text{score} = T[i]\gamma^{2^{j-i}} + T[i+1]\gamma^{2^{j-i-1}} + ... + T[j]\gamma^{2^0}$ assigning to each position a unique value according to the method propose in [7], where T[i] is the i-th term value and γ is the constant deciding the importance of the position, creating a scalar value.

The same dimensional reduction has applied for the POS tagging and NER, using the fingerprint in order to uniquely identify the part of speech or entity taken from the glossary dictionary from the Spacy Library [8].

Finally, all extracted features were normalized between 0 and 1, taking into consideration the *prompt_ id* (for the same reasons above) in order to be managed by neural network models. In Table 2 we show how all these features are correlated with target variables.

Feature	Content	Wording	Absolute Sum
tfidf_scores	0.8364	0.5685	1.4049
distance	0.5488	0.3324	0.8812
text_pos	0.5985	0.4701	1.0686
2grams-count	0.4841	0.1438	0.6279
3grams-count	0.3804	0.0425	0.4229
4grams-count	0.3351	0.0134	0.3485
misspelled_counter	0.6880	0.5564	1.2444
summary_word_counter	0.7916	0.5163	1.3079
summary_word_ratio	0.7751	0.5507	1.3258
text_length	0.7816	0.5566	1.3382
length_ratio	0.7816	0.5564	1.338
different_word_counter	0.7975	0.5524	1.3499
different_word_ratio	0.7975	0.5524	1.34989
stopword_counter	0.7541	0.5386	1.2927
stopword_ratio	0.7541	0.5386	1.2927
punctuation_counter	0.6747	0.4518	1.1265
punctuation_ratio	0.6747	0.4518	1.1265
word_counter	0.7751	0.5507	1.3258

Table 2: Correlation values with the targets (content and wording) and the absolute sum of both, please note that the correlation indicates only the trend of the sets that is the reason why the *count* and *ratio* feature correlations are the same.

4 Models

This section presents the models we implemented informing about the methods and variations that we adopted so that the requirements of the task can be met. We started testing Recurrent Neural Networks, then we used pre-trained LLMs. While LSTMs have been effective in various sequence tasks, Transformers have become the preferred choice for many state-of-the-art natural language processing applications due to their ability to capture complex contextual information efficiently. In Section 4.1 we will talk about Bidirectional LSTM and in Section 4.2 we will focus on different LLMs.

4.1 Bidirectional LSTM

The **Bidirectional Long Short-Term Memory** network (BiLSTM) has emerged as a pivotal tool for tackling a wide array of complex tasks, ranging from natural language processing and speech recognition to time series analysis and bioinformatics. Its distinctive ability to capture contextual information from both past and future contexts has made it a versatile and indispensable tool in data-driven research and applications. Those reasons drove us to use them as an initial approach to the problem, to establish a baseline as they represented the state of the art for a long time before they were outperformed by transformers. In particular, Bidirectional Long Short-Term Memory differs from the Unidirectional model in the fact that they process input in both directions, trying to reduce the problem of forgetting, caused by the length of input and long-distance dependencies present in it, essentially creating an embedding-per-token realized from the concatenation of precedent and successive tokens, embracing a wider context.

Our architecture will take in input the sequence of tokens representing the concatenation of *prompt_ text*, *prompt_ question* and *text*, tokenized with the tokenizer of bert-base-cased and incorporated by an embedding layer to be suitable for the LSTM. Afterward, the context generated by LSTM is passed through a ReLu layer and concatenated with the feature extracted, as shown in figure 2.

We would like to justify the choice of the tokenizer taken from the bert-base-cased due to the extensive availability thanks to the Hugging Face Community and because BERT is one of the first transformers so his tokenizer is one of the most ordinary.

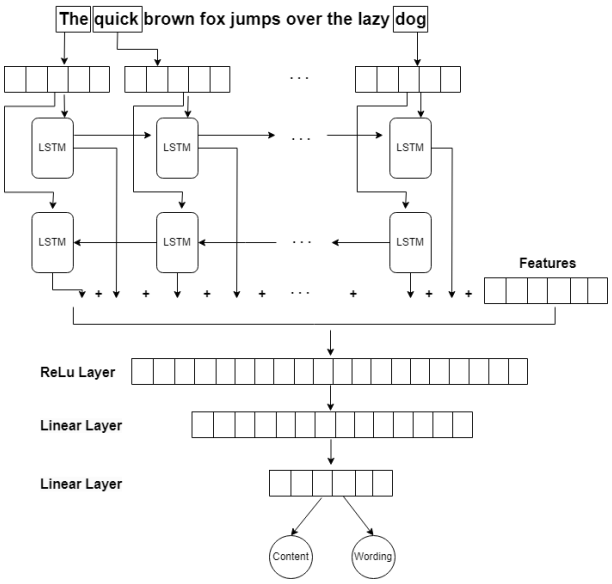


Figure 2: Architecture of Bi-LSTM

4.2 Large Language Models

After the first approach with the Bi-LSTM model, we decided to perform fine-tuning over pre-trained LLMs, given the recent developments such models have achieved in the field of NLP. We started with the best-known and most established transformer, namely **BERT**. We decided to test the **bert-base** and **bert-large**, which can receive up to 512 tokens in input, both in the **cased** version since those are case-sensitive and therefore is most suited for our case, especially for the *wording* score. We remark that the competition's aim is to evaluate the summaries written by the students also from a grammatical point of view, so making the distinction between capital letters and non-capital ones is therefore essential.

We then decided to try the **DeBERTa** model for its disentangled attention matrix for content and position and enhanced masked decoder to learn from the precedent attention mechanism using a relative position embeddings in order to completely understand the spatial modality [9]. These improvements, permit to acquire more information and enlarge the max-length of tokens, compared to its predecessors. In fact, it can receive up to 1024 tokens as input sequences. The ability to receive more tokens is essential to process a larger input and to understand a wider context, indeed many summaries and *prompt_ text* can be longer than 512 tokens, as we have seen from 1. So we first tested **deberta-v3-base** and then we tried **deberta-v3-large** with 304M backbone parameters, 218M more than the base version, this decision has been made in order to achieve a better result in the plain competition, ignoring the efficiency constraints. Training is done end-to-end using the PyTorch [10] library.

5 Experiments

In this section, we will describe how we conducted the experiments and which tests we performed. First of all, we will provide general information about all the experiments to give an overall assessment of our models. We divided the available data into training set (60% of data), validation set (20% of data), and test set (20% of data). We have taken care to split the data considering the distribution of *prompt_ id*, which uniquely identifies the paragraph to be summarized, so each subset of data has the same distribution of *prompt_ id*.

As we noticed in Section 3.1, the dataset is unbalanced, therefore we decided to oversample only the training set during model training, to have an equal number of examples for each *prompt_ id*.

After training the model and evaluating its performance on the internal test set, we re-trained the model by dividing the available data into two subsets only: training set (90% of the data) and validation set (10% of the data). This was done to fit the model on as much data as possible.

The architectural choice for the models is to feed text input to LSTM or LLMs to generate embeddings, simultaneously, to improve the performance, the same text has been engineered to extract the features discussed above in 3.2 to extend the embedding vector. We then attached multiple dense layers to the

end of the models thus creating a feedforward neural network. The first hidden layer of this network is twice the size of the input vector, projecting the extended embedding into a larger space. Subsequent hidden layers halve the size of the previous layers in a funnel-like structure. The activation function used for these layers is ReLu. In order to regularise the model in training and to avoid overfitting, a dropout layer was added after each dense layer, and l2-regularisation was used on the norm of the weights. Also, to avoid overfitting, the training was done using early stopping.

5.1 Bi-LSTM Experiments

Experiments carried out on LSTM will be our starting point and baseline with which to compare more complex models like transformers. In the next paragraphs, we will explain how we have set these experiments.

Data Preprocessing: Bi-LSTM input sequence is the concatenation of *prompt_ text*, *prompt_ question* and *text*. As emerged from the dataset analysis, the length of concatenation of prompt text and summaries can even exceed 1500 tokens. This matter brought us to try out different formats for the input of the model. Initially, we tried fixing the maximum size of the input to **2000 tokens** and in case a certain input was shorter than this max length, we will proceed to pad the sequence with a special token to reach the required length. Then, as the results didn't quite satisfy our expectations, we decided to train LSTM with dynamic input length to avoid some of the sequences being composed mostly of padding tokens, as input length can vary a lot within the dataset. Unfortunately, in this last case, we lost the possibility to train the model with mini-batches, as we necessarily have to use online learning thereby extending considerably the training time.

Training and hyperparameter tuning: Initially, we tried to apply a sigmoid activation function for hidden layers, but we noticed a worsening pattern in the loss, so we stuck on using linear layers, as they seemed to perform better. For training, we used the **Adam** optimizer, for the case of padding the batch has been set to **10**, instead, as said, it is equal to 1 when we train with variable length sequences. To perform model selection and evaluate the Bidirectional LSTM model, it has been performed a grid search exploring different values for the size of embedding, the dropout rate (as regularization parameter), and the size of final dense layers. Is then used early stopping method with a patience parameter equal to 1 to interrupt the training as the validation score starts to get worse.

5.2 LLMs Experiments

Large Language Models are very big models that require large computational resources. Given the purpose of the competition, we have used a few expedients, which we will list below, both to improve the efficiency of the models and to train them using as few resources as possible.

Freeze Embedding We decided to freeze the weights of the LLMs as they are pre-trained models and would therefore perform well even without training. Freezing the weights leads to a significant decrease in training time and better memory usage. In addition, experiments were conducted by not freezing the weights in the last layer, so that fine-tuning could be performed for our task. Unfortunately, however, these experiments led to out-of-memory errors during submissions and were therefore discarded.

Gradient Accumulation When we train a neural network, our dataset is divided into mini-batches, and each batch is processed sequentially. The network makes predictions for each batch, which are used to calculate the loss concerning the actual target values. To improve the model, a backward pass is performed, computing gradients that guide the updates to the model's weights. With gradient accumulation [11] instead of updating the network weights on every batch, we can save gradient values, proceed to the next batch, and add up the new gradients. The weight update is then done only after several batches have been processed by the model. The primary benefit of gradient accumulation is its ability to emulate a larger effective batch size during training. This has several advantages like memory efficiency, in fact, training with larger batch sizes can be memory-intensive, particularly on GPUs, and gradient accumulation helps decrease memory usage and allows to use of larger effective batch sizes, often leading to more stable training processes and resulting in smoother convergence and potentially better generalization.

Dynamic Padding and Uniform Length Batching Improvements can be made during the tokenization phase, in fact, when we train LLMs on batches of sequences we need them all to be the same length. The dataset we have at our disposal, but also in the real world, contains sequences of different lengths, all different from each other. What is then done is to truncate the sequences that exceed a certain number of tokens in length, or for sequences that are too short, to perform padding, i.e. adding a special token [PAD] to the sequence in order to obtain the desired length. Because the pad token does not represent a real word, when most computations are done, before computing the loss, we erase the pad

token signal by multiplying it by 0 through the "attention mask" matrix for each sample, which identifies the [PAD] tokens and tells Transformer to ignore them. With **dynamic padding**, rather than padding all strings in the dataset considering the longest string or the maximum length, padding is done considering the longest string in the mini-batch. As the number of pad tokens added changes for each batch this leads to improvements in model efficiency since the number of pad tokens added is reduced and this leads to fewer computations being performed. An example can be seen in Figure 3. To further improve the efficiency of the model, we perform what is called **Uniform Length Batching**, which consists of generating mini-batches that contain sequences of more or less the same length, thus trying to minimize the number of pad tokens inserted and to avoid very short strings being put in the same batch as very long strings. Figure 4 shows an example of uniform length batching. So the process is to first sort the strings by length, then create mini-batches in a random way and not sequentially, where each mini-batch contains strings of more or less the same length, and finally perform padding within the batch. From experiments, this results in far fewer total tokens.

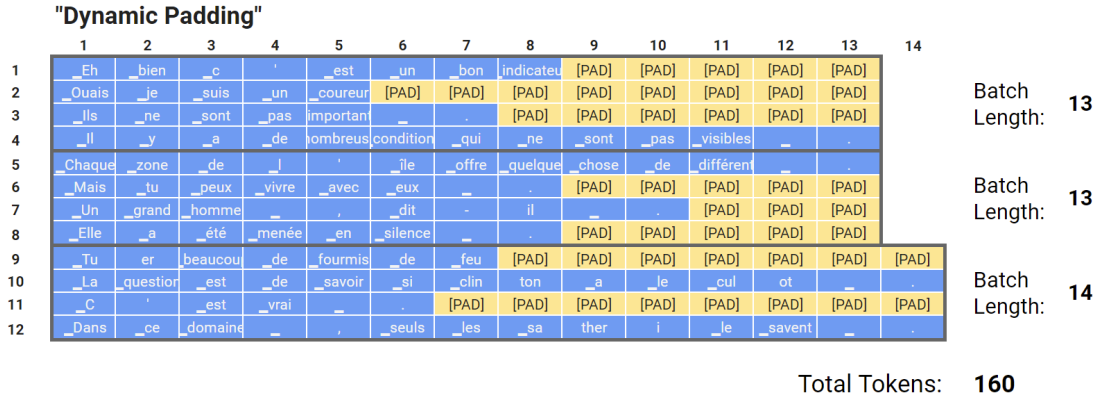


Figure 3: An example of dynamic padding. The picture is taken from the web and does not represent our dataset, but still, it makes comprehensible the technique.

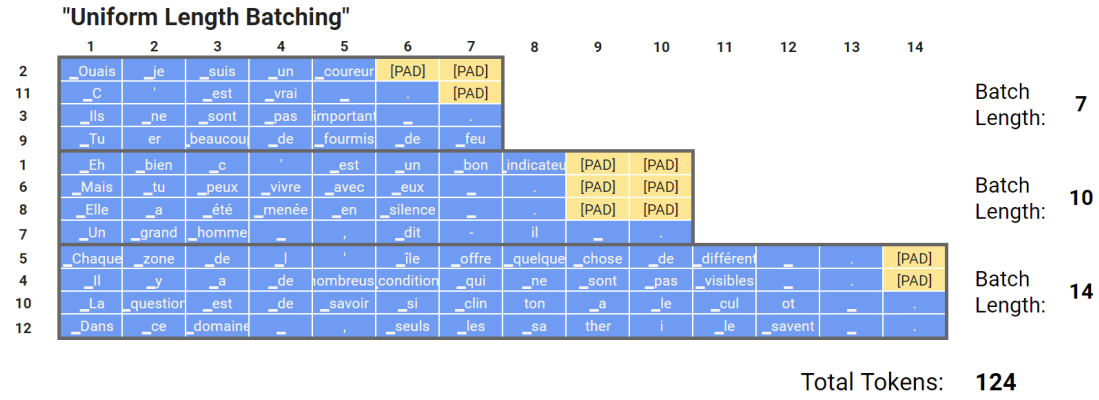


Figure 4: An example of Uniform Length Batching. The picture is taken from the web and does not represent our dataset, but still, it makes comprehensible the technique.

Pooling LLMs output are sequences of embeddings where the first token of the sequence, is the [CLS] token. To fine-tune our LLMs for a regression task different types of pooling strategies can be used in order to manage the pooler output. The first one that we tried is the *CLS Embedding* method, where the idea is that [CLS] token would have captured the entire context and would be sufficient for simple downstream tasks such as regression. So the idea was to extract CLS Embedding from the last hidden state of the model and use it as input for the feedforward part of the net. In Figure 5 an example of CLS Embedding.

Next, we tried what is called *Mean Pooling* where given that the last hidden state is of shape (*batch_size * max_len * hidden_state_dim*), instead of considering only the [CLS] token, we consider the average across *max_len* dimensions to get averaged embeddings. From our experiments, it was found that mean pooling achieves slightly better results in terms of loss.

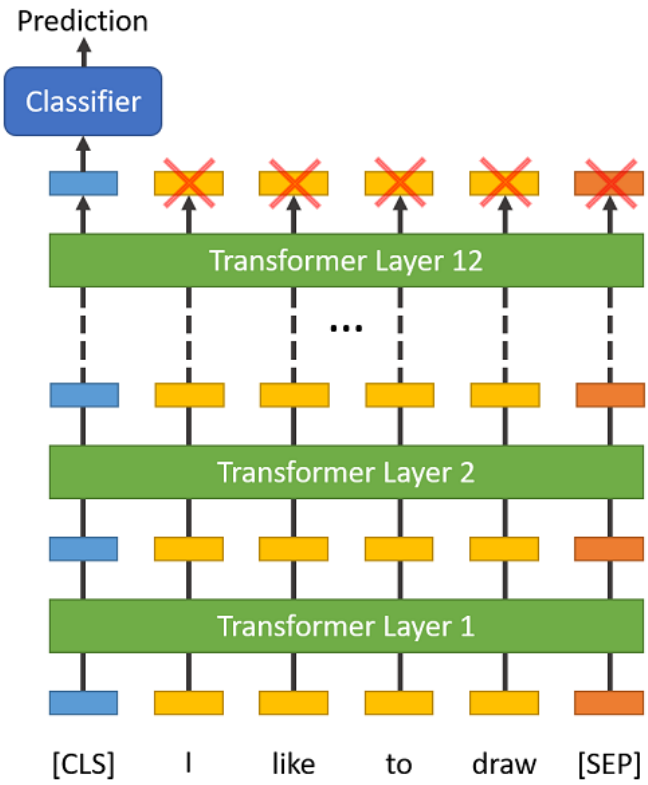


Figure 5: CLS Embedding example.

Input choices We thought that combining more strings as input for the LLMs would give more context, so in our experiments, the input for such models is given by combining the *prompt_ question* and the summary of the student, specifically [CLS] *prompt_ question* [SEP] text [SEP].

The choice to concatenate only these two strings is due to the fact that if we were to concatenate *prompt_ text* and *text* we would have sentences that were too long because the maximum number of tokens, as seen in 3.1, is longer than the maximum length the model can take as input, and therefore there would be significant truncation of the sentence, losing context.

Moreover, we conducted various experiments on the best combination of input to use, such as using the title and the summary, or we tried correcting the text of the summary and combining it with the title or question, but these experiments led to slightly inferior results compared to the above-mentioned combination. The evaluation done on all the possible input combinations are performed using a **Grouped K-Fold cross-validation**, since *prompt_ id* identify the class, we trained the model only on three prompt_ids and validated on the remaining one. We repeated it for each prompt and we computed the mean across the folds.

Finally, in our experiments, we noticed that gradient accumulation allowed us to use larger batch sizes, but a careful choice must be made on the batch size in order to avoid out-of-memory errors. We used then a batch size of **8** and a gradient accumulation step of **16**. All experiments were performed considering *Cosine Annealing Warm Restarts* [12] as a scheduler to facilitate convergence. A model selection with a random search was performed to find the best parameters regarding the number of layers for the feedforward part of the model, learning rate, weight decay, and dropout rate. In Table 3 we show information about the parameters used for the LLMs, instead in Figure 6 we can see the prototype of the final model.

Model	Max Tokens	FFlayers	Dropout	Learning Rate	Weight Decay
bert-base-cased	512	3	0.1	0.0013	0.01
bert-large-cased	512	4	0.05	0.0008	0.015
deberta-v3-base	1024	4	0.05	0.0013	0.01
deberta-v3-large	1024	4	0.1	0.0013	0.01

Table 3: LLMs tested and hyperparameters chosen

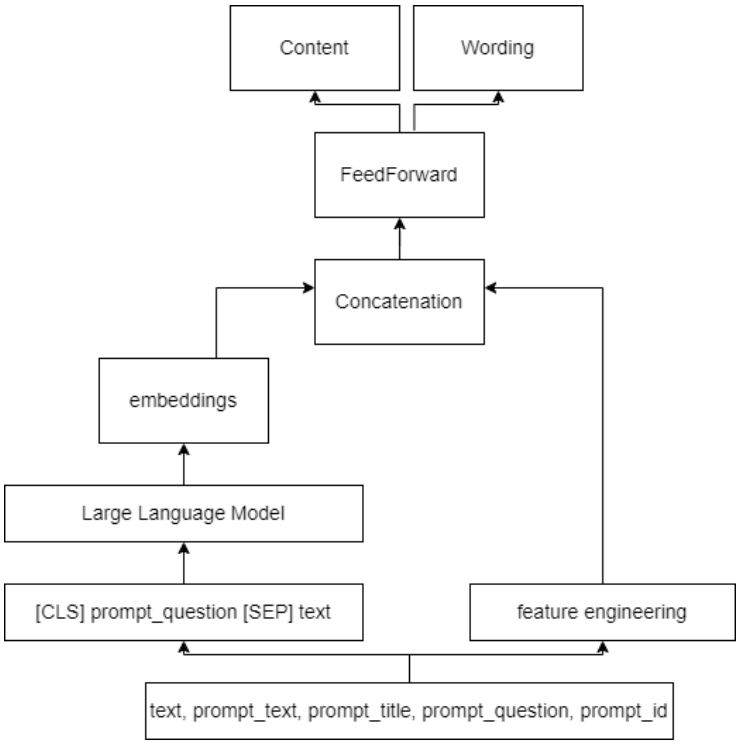


Figure 6: Example of the full large language model.

5.3 Competition Experiments

As pointed out in Section 3.2 the features extracted are many, unfortunately, as far as the competition is concerned, it was not possible to use all of them as the computation of some of them on the organizers' private test set took more time than allowed by the rules of the competition. So, in the end, we used the features below:

1. text_length;
2. summary_word_counter;
3. 2grams_counter;
4. 3grams_counter;
5. 4grams_counter;
6. misspelled_counter;
7. summary_word_ratio;
8. punctuation_counter;
9. different_word_counter;
10. stopword_counter;
11. length_ratio;
12. tfidf_scores;
13. punctuation_ratio;
14. different_word_ratio;
15. stopword_ratio.

Moreover, we tried to build also two different models, The first one was for predicting content scores using corrected student summaries, and features extracted from it. The second one was for wording scores using the original text. As we have seen the tokenizer inserts a special token for words missing from his vocabulary (`[UNK]`), this peculiarity leads to a different embedding that is specialized in highlighting misspelled words.

Unfortunately, these experiments did not lead to conclusive results since as mentioned above some features caused the experiment to run more than admitted.

5.4 Ablation Experiments

Feature ablation is a technique for calculating feature importance that works for all machine learning models. Given a dataset composed of m features, the procedure goes like this:

1. Train the model on the train set and calculate a score on the test set.
2. For each of the m features, remove it from the training data and train the model. Then, calculate the score on the test set.
3. Rank the features by the difference between the original score (from the model with all features) and the score for the model using all features but one.

So, in our tests, all the experiments have been computed in accordance with the following practice: the *bert-base-cased* model is used as comparison architecture, with the usual feed-forward layers to concatenate the features and the neurons adapted accordingly. Then, again, we used the Grouped K-Fold Cross-Validation technique, as was done for the input combination choice, and the scores were averaged for all the folds. This was done in order to the fact that the real test set has unseen *prompt_ id*, so the best way to assess the incidence of our feature is to train the model on some prompts and test it on a never-seen one. The experiments then proceed by removing one of the features at a time and computing the mean of the loss across the folds. The results will be discussed in Section 6.2.

In our case, ablation tests require a high number of model evaluations, and since there are only 5 submissions per day allowed by the competition with a maximum runtime of 9 hours, and their test set is really large, we didn't have the opportunity to conduct the ablation tests on the competition test set.

Finally, additional ablation studies were conducted on the features that could not be used for competition purposes. These results are shown in the Appendix 7.

6 Results & Discussion

In this section, we are going to provide information about test results and submission scores obtained. In Section in Section 6.1 we will show our score in the completion, while in Section 6.2 we will provide the results of the ablation tests.

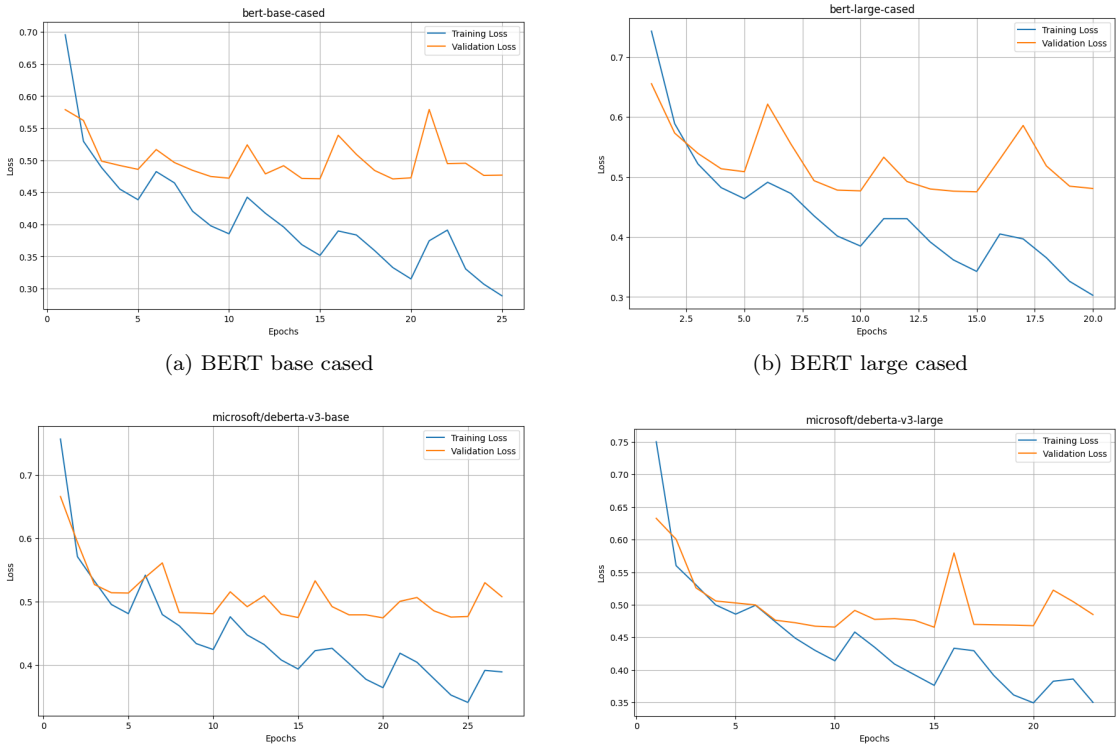
6.1 Competition Results

The results of the grid search are the ones shown below in Table 4. After the model selection, we retrain and evaluate the best model found on an unseen Test set, on which the model reaches the score of **0.562**. As we can see according to Table 5 Bi-LSTM are the ones performing worse, reaching **0.594** in public score, a symptom that those architectures don't possess the same expressiveness as transformers. The cause can be that despite the bidirectional property, book chapters can be very long in terms of word numbers. Indeed LSTM can struggle with capturing long-range dependencies effectively due to their sequential nature, while Transformers excel at this with their attention mechanisms. Moreover, LSTMs may require more hyperparameter tuning to perform well, whereas Transformers like BERT are already pre-trained and only require fine-tuning to have good performance. Finally, LSTMs are computationally expensive, especially for deep networks, making training and inference very slow. Transformers, on the other hand, are inherently parallelizable and can handle larger datasets more efficiently. For this reason, we were not able to get a public score from the submission for the competition.

Emb, dropout, h1, h2	Training score	Validation
40, 0.2, 150, 80	0.531	**0.532**
40, 0.1, 300, 50	0.532	0.536
40, 0.1, 200, 100	0.477	0.539
40, 0.2, 150, 50	0.496	0.539
40, 0.1, 200, 50	0.492	0.539
40, 0.1, 150, 80	0.698	0.543
40, 0.1, 150, 100	0.698	0.544
70, 0.1, 150, 100	0.463	0.544

Table 4: Grid_search of LSTM

Regarding LLMs, we can see in Figure 7 the progress of the learning curves. We see that the models actually learn and the idea of combining features with embeddings works. We also notice that the curves are not so smooth, this is given by the use of the Cosine Annealing Warm Restart as a scheduler since the learning rate is increased or decreased according to a schedule.



(a) BERT base cased

(b) BERT large cased

(c) DeBERTa base

(d) DeBERTa large

Figure 7: LLMs learning curves side by side.

In table 5 we show information about the score that we get with all the models and relative training time.

We notice how *bert-base* is the fastest model since it is the smallest one and it reached a good score overall. Using a bigger model like *bert-large* did not improve the results, this may be because a deeper model selection must be done. Switching to DeBERTa models, we saw that using a model with a bigger maximum token as input actually helps in improving the score. In fact, *deberta-base* improved the score with respect to *bert-base* and *deberta-large* was the model that scored best among the ones we tried. This is obviously at the disadvantage of efficiency as *deberta-large* is the largest model we tested and the one that took the longest in training.

Model	Valid.Set Loss	Internal TestSet Loss	Public Score	Training Time (in seconds)
Bi-LSTM	0.532	0.562	0.594	1890.54
bert-base-cased	0.4708	0.4585	0.52	**860.17**
bert-large-cased	0.4753	0.4615	0.559	2435.94
deberta-v3-base	0.4740	0.4512	0.511	1256.11
deberta-v3-large	**0.4655**	**0.4430**	**0.468**	3765.99

Table 5: Internal test set and public score results.

We must specify that public scores are temporary results, in fact, the competition is still running and public scores are provided using only the 13% of the test set. We note that the first in the ranking currently has a score of **0.421**. The final results will be provided at the end of the competition.

6.2 Ablation Test Results

As already presented above, ablation studies are conducted by removing one feature at a time. In Table 6 we can see the results of the model evaluation, as said, averaged on the 4-fold validation scores. The minimum of these values signals the most useless feature.

Removed Feature	Validation Loss	Standard Deviation
All Features	0.545737	0.0465
text_length	**0.543067**	**0.0465**
different_word_cnt	0.546947	0.0460
summary_word_ratio	0.54804	0.0440
different_word_ratio	0.550173	0.0431
stopword_ratio	0.550542	0.0431
stopword_cnt	0.552363	0.0466
misspelled_cnt	0.554444	0.0448
2grams_cnt	0.557822	0.0428
4grams_cnt	0.558265	0.0430
length_ratio	0.558557	0.0440
tfidf_scores	0.560073	0.0433
3grams_cnt	0.560289	0.0433
punct_cnt	0.560657	0.0467
summary_word_counter	0.562036	0.0477
punct_ratio	0.56206	0.0432

Table 6: The results of the ablation study ordered by the validation results. As we can see the text_length feature has been identified as the less critical.

As we can see, *text_length* is the least useful feature for the purposes of our model since its removal does not cause major deterioration in the model score. Differently, *punctuation_ratio* can be our most important feature since without this one the model has the greatest deterioration in terms of loss. Also, it appears that removing *text_length* feature would lead to slightly better results in terms of loss, but it is negligible.

We remark on the fact that the scores reported above are computed on a never-seen prompt task leading to higher validation results, thus these numbers may not be compared with precedent metrics.

7 Conclusions & Possible improvements

In conclusion, we recap briefly what we did. First of all, we have reassessed the importance of old-school metrics, based on statistical data, in order to highlight as much as possible the different aspects that may concern an elaborated text. After that we were able to evaluate the power but also the various limitations of pre-trained models. In fact, the use of these huge architectures requires many precautions. Beyond the scope of the competition, some improvements could be made, and some experiments could be done using other LLMs, such as RoBERTa [13] or AlBERT [14] to compare their improvements with BERT and challenge them with DeBERTa. Furthermore, some studies show that concatenating the outputs of the last four hidden layers leads to better results [15]. Also, many other features can be extracted from the data, like semantic similarity scores between *text* and *prompt_text* and many more. Finally, in our experiments the features are concatenated in a direct way to the embeddings, an improvement could be to use a feedforward neural network that takes in input only the features and returns in output a vector of the same size as the input, so that it learns a representation of the features and use this in concatenation with the embeddings. In conclusion, we can say that participating in this competition was really exciting and challenging. It gave us the opportunity to confront real problems and to learn a lot, even from the other teams that participated.

References

[1] Alex Franklin, asiegel, HCL-Jevster, Maggie julianmante, Natalie Rambis, Perpetual Baffour, Ryan Holbrook, and Scott Crossley. Commonlit - evaluate student summaries, 2023.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[3] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[4] A.Sardelli G.Carfì, C.Peluso. Commonlitchallange. `https://github.com/Sopralapanca/CommonLitChallenge`, 2023.

[5] Ei Phyu Phyu Mon, Ye Kyaw Thu, Than Than Yu, and Aye Wai Oo. Symspell4burmese: Symmetric delete spelling correction algorithm (symspell) for burmese spelling checking. In *2021 16th International Joint Symposium on Artificial Intelligence and Natural Language Processing (iSAI-NLP)*, pages 1–6. IEEE, 2021.

[6] Stephen Robertson. Understanding inverse document frequency: On theoretical arguments for idf. *Journal of Documentation - J DOC*, 60:503–520, 10 2004.

[7] Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987.

[8] Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. spaCy: Industrial-strength Natural Language Processing in Python. 2020.

[9] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: Decoding-enhanced bert with disentangled attention, 2021.

[10] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[11] Joel Lamy-Poirier. Layered gradient accumulation and modular pipeline parallelism: fast and efficient training of large language models, 2021.

[12] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2017.

[13] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[14] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

[15] Junjie Yang and Hai Zhao. Deepening hidden representations from pre-trained language models. *arXiv preprint arXiv:1911.01940*, 2019.

I Appendix

This part of the report has been used in order to clarify some aspects that cannot be covered in the other sections.

∗ More Ablation Studies

We have extracted a variety of features, such as the comparison of the summary corrected with the one misspelled, extracting the Edit-Distance between the twos, or the part of speech extraction analyzing the structure of the sentences. Being them the pillar of the old school's Natural Language Processing, we decided to give them the deserved importance, even though they have found no place to be useful in the competition. So this part's objective is to highlight the relevance of these features in correspondence with the architecture used.

As other features have already demonstrated to be relevant for the purposes of the problem, in Table 7 we have reported an ablation study with only the features not used for the submission.

We can see that the model felt more relevant to simple features that are a direct hint of the lexicon used by the student, such as the number of different words used, the number of bi-grams in common with the corrected text, or the Edit-Distance. Features like NER, POS, and TF-IDF instead have a mid result, maybe the network didn't succeed in finding a useful abstraction to use them properly, indeed there have been made great efforts in reducing this metric in scalar in order to use them, in a simplistic way, with the neural network.

Removed Feature	Validation Loss	Standard Deviation
3grams_correct_cnt	0.647502	0.0712
misspelled_text_cnt	0.654374	0.0684
stop_cnt	0.657405	0.0746
entities	0.671018	0.0708
text_pos	0.671214	0.0709
tfidf_scores	0.671826	0.0705
4grams_correct_cnt	0.672735	0.0702
length_ratio	0.678742	0.0694
distance	0.680196	0.0689
2grams_correct_cnt	0.682946	0.0733
different_word_cnt_ratio	0.704690	0.0742

Table 7: The results of the ablation study ordered by importance achieved from the model. We can see that the 3-grams count with the corrected and original text is the most important feature, followed from the misspelled words count and stop words count, so the number of useless words used for the summary.

Figure 8: Correlation between these features and the targets.