HSLU

AI/ML

Evaluating Student Summaries

AICH Report

Team 4

Jonathan Carona, Tharrmeehan Krishnathasan, Josef Rittiner

Coach:
Pascal Baumann

Lucerne University of Applied Sciences and Arts
January 14, 2024

Eidesstattliche Erklärung

Wir erklären hiermit, dass wir die vorliegende Arbeit selbständig und ohne unerlaubte fremde Hilfe angefertigt haben, alle verwendeten Quellen, Literatur und andere Hilfsmittel angegeben haben, wörtlich oder inhaltlich entnommene Stellen als solche kenntlich gemacht haben, das Vertraulichkeitsinteresse des Auftraggebers wahren und die Urheberrechtsbestimmungen der Hochschule Luzern respektieren werden.

Jonathan Carona
Hochschule Luzern

Josef Rittiner
Hochschule Luzern

Tharrmeehan Krishnathasan
Hochschule Luzern

Abstract

This project revolved around the Kaggle Challenge: CommonLit - Evaluate Student Summaries. The aim of this challenge was to create a machine learning model, which could accurately assess the quality of summaries written by students in grades 3-12. Using the data provided by the challenge, which contained the summaries of over 7'000 students across four different topics, the team developed three distinct models.
The ROUGE-Based Model is a Neural Network, which uses ROUGE metrics as it's input features. The LightGBM Model uses an array of scores, which can determine readability, complexity and the grade level of a summary. The DeBerta Transformer, a state-of-the-art language model which balances out the two while also considering the specific task the students were asked to capture in their summary.
These three models collectively form the final Ensemble Model. Each model independently preprocesses the texts and generates predictions. The Ensemble Model weighs the contributions of each model to produce a final score. The results showcase respectable performance in the Kaggle competition, placing 975th out of a total 2'065 participants.

Contents

1 Introduction

1.1 Problem Statement

This project stems from the Kaggle Challenge: Evaluate Student Summaries,[1] which was hosted by CommonLit. The objective of this competition was to assess the quality of summaries written by students in grades 3-12.
This involved developing a machine learning model to predict the scores given by human annotators for two different metrics: "Content" and "Wording", as described in Figure 1.1.

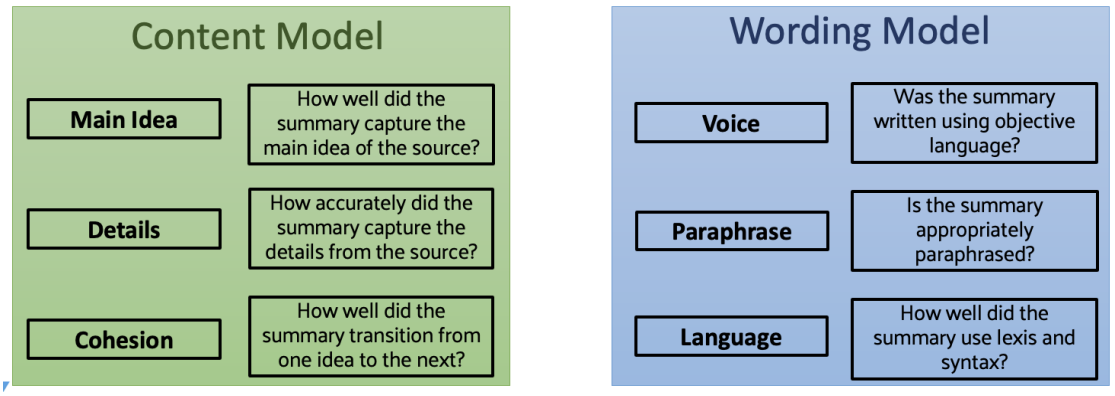

Figure 1.1: Explanations of the two target metrics to predict. Source: CommonLit - Evaluate Student Summaries

The human annotators were instructed to ignore grammar and spelling when evaluating the summaries. Additionally, the annotators were unaware of the students' grade levels. Meaning all summaries, regardless of what grades the students were in, were evaluated equally. This is also reflected in the data. There is no information on the students, only a unique student ID and the summary itself, together with its wording and content scores and an ID to identify to which reference text it was based on. Further details about the data are provided in Chapter 3.

[1] CommonLit - Evaluate Student Summaries - Website

1.1.1 Background and Motivation of Problem

CommonLit is a nonprofit organization dedicated to promoting reading, writing, communication, and problem-solving skills in students. Writing summaries is a crucial skill that encompasses reading comprehension, writing proficiency, and critical thinking. Unfortunately, it is a time-consuming task for teachers to read, correct, and grade students' summaries, limiting the opportunities for students to practice these skills.
Being able to automatically evaluating students' writing would significantly reduce the workload for teachers. But there is a scarcity of datasets containing student writing. As a result, current techniques for summary evaluation primarily focus on automatically generated summaries rather than those created by humans, let alone students.

1.2 Goal of this Work

The primary objective of this project was to leverage the data provided by the challenge and develop a model capable of automatically predicting accurate scores. The Kaggle challenge presented two distinct prize categories: Leaderboard and Efficiency. Our focus was on the Leaderboard category, as the we believed that prioritizing accuracy over efficiency would provide a more valuable learning experience. Moreover, deliberately overfitting models was avoided, as we deemed it unrepresentative of real-world projects.

1.3 Evaluation Function

Submissions to the challenge were scored using the Mean Columnwise Root Mean Squared Error (MCRMSE), calculated as follows:

$$MCRMSE = \frac{1}{N_t}\sum_{j=1}^{N_t}\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_{ij}-\hat{y}_{ij})^2} \tag{1.1}$$

N_t represents the number of target columns, with $N_t = 2$ in this case (one column for the content scores and one for the wording scores). The variable n corresponds to the number of rows, i.e. the total number of summaries to be evaluated. And the variables y and $\hat{y}$ denote the actual and predicted values, respectively.
In short, the MCRMSE is obtained by computing the Root Mean Squared Error (RMSE) for each of the two metrics: content and wording. The final score is then determined by averaging all the calculated RMSEs across those two metrics.
The range of the MCRMSE is non-negative, with 0 being the theoretical best possible score. (It can be calculated easily in Python using the scikit-learn library.)

```
from sklearn.metrics import mean_squared_error
rmse_content = mean_squared_error(y_true[0], y_pred[0], squared=False)
rmse_wording = mean_squared_error(y_true[1], y_pred[1], squared=False)
mcrmse = 1/2 * (rmse_content + rmse_wording)
```

2 Machine Learning Theory

2.1 Neural Network

Neural Networks (NN) are machine learning models inspired by the structure of the human brain. Comprising three distinct layers, (input, hidden, and output) NNs consist of artificial neurons. In each layer, every neuron is connected to all the neurons in the previous and the next layer through learnable weights. Additionally, each neuron has an activation function, introducing non-linearity to the model's computations. By adjusting the weights between neurons, the model effectively learns patterns within the data, enabling it to make predictions based on given inputs.
Using NNs, both classification and regression problems can be solved. By either learning the features or by transforming predetermined features, NNs can predict the output based on a given input.

2.2 Recurrent Neural Network

2.2.1 Recurrent Neural Network

> "Recurrent Neural Networks (RNNs) are a type of neural network architecture which is mainly used to detect patterns in a sequence of data. [...] While Feedforward Networks pass information through the network without cycles, the RNN has cycles and transmits information back into itself. This enables them to extend the functionality of Feedforward Networks to also take into account previous inputs $X_{0:t-1}$ and not only the current input X_t." (Schmidt, 2019)

RNNs do not have a fixed input size, making them great for processing sequential data of varying lengths, such as summaries or other texts. This flexibility allows RNNs to evaluate and calculate values from data sequences of any length.

2.2.2 Long Short-Term Memory

> "Since [LSTMs] use a more constant error, they allow RNNs to learn over a lot more time steps (way over 1000). To achieve that, LSTMs store more information outside of the traditional neural network flow in structures called gated cells" (Schmidt, 2019)

LSTMs extends the capabilities of RNNs. They are better at maintaining and managing relevant information across longer sequences, making them better suited for processing and understanding lengthy sequential data.

2.3 Transformer

> "The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely." (Vaswani et al., 2023)

Transformers are a type of machine learning model primarily used for processing sequential data like text or time series. They excel at capturing dependencies between input and output regardless of their distance in the sequence. This is achieved through a mechanism called "attention," which allows the model to weigh the importance of different parts of the input data when making predictions.

2.3.1 DeBERTa

> "Recent progress in pre-trained neural language models has significantly improved the performance of many natural language processing (NLP) tasks. In this paper we propose a new model architecture DeBERTa (Decoding-enhanced BERT with disentangled attention) that improves the BERT and RoBERTa models using two novel techniques." (He et al., 2021)

Disentangled Attention In standard transformer models like BERT, the attention mechanism is based on the interaction between the learnable values query (Q), key (K), and value (V) matrices derived from input embeddings.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.1}$$

where d_k is the dimension of the key vector k and query vector q.
DeBERTa, which is a BERT-based model, disentangles the attention mechanism into two parts: content-based attention and position-based attention.

Content-based Attention This part of the attention mechanism focuses on the semantic meaning of the tokens (content), similar to standard attention mechanisms. It computes the attention scores based on the content of the query and key tokens.

Position-based Attention This part is unique to DeBERTa. It computes attention scores based on the relative positions of tokens, independent of their content. This allows the model to understand the influence of the positional relationships of words on the overall sentence meaning.

Combining Content and Positional Attention Combining these two types of attention allows the model to capture both the content and their positional relationships, enhancing its ability to understand context and meaning.

2.4 Light Gradient-Boosting Machine

LightGBM is a Gradient-Boosted Decision Tree (GBDT) based Algorithm with two novel techniques, those being Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB).

GOSS This is a novel sampling technique, where data instances with smaller gradients are dropped randomly and data instances with larger gradients (or under-trained instances) are kept. This is due to the fact, that the information gain is higher for data instances with a larger gradient. The GOSS technique only focuses on those larger gradients. This effectively means that a large part of the performance can be retained, while using less data altogether and thus reducing the effective training time.

EFB This is a technique, where datasets with a high feature count get their dimensionality reduced. This is due to the fact that many of those datasets have sparse features (features with many zero entries). Those sparse features get combined into 'bundles' and since only zeros get overwritten, this does not affect the final accuracy while at the same time reducing the effective training time.

According to the authors: "LightGBM speeds up the training process of conventional GBDT by up to over 20 times while achieving almost the same accuracy." (Ke et al., 2017, p. 1)

For a concise explanation of those techniques and details regarding the LightGBM model, please refer to Ke et al., 2017.

2.5 ROUGE Measures

> "ROUGE stands for Recall-Oriented Understudy for Gisting Evaluation. It includes measures to automatically determine the quality of a summary by comparing it to other (ideal) summaries created by humans. The measures count the number of overlapping units such as n-gram, word sequences, and word pairs between the computer-generated summary to be evaluated and the ideal summaries created by humans." (Lin, 2004)

ROUGE Metrics

The paper describes four different ROUGE metrics, each with distinct variants:

ROUGE-N ROUGE-N measures the overlapping n-grams between the summary and the reference text, where N denotes the size of the n-grams. ROUGE-1 and ROUGE-2 are the most often used variations of ROUGE-N. However N can be any natural number.

ROUGE-L ROUGE-L calculates the Longest Common Subsequence (LCS) between the reference and the summary. Where a Subsequence is a sequence of tokens that occur in the same order, but do not have to be consecutive.

ROUGE-Lsum ROUGE-Lsum, a variation of ROUGE-L, also calculates the LCS between the reference and the summary. But ROUGE-Lsum splits the texts into 'sentences' by new-lines and calculates the LCS for each of these sentences. The average ROUGE-L score of these sentences is the final ROUGE-Lsum score.

ROUGE-W The ROUGE-W metric also evaluates the LCS between the reference and summary. However, ROUGE-W emphasizes consecutive tokens, assigning them greater importance based on the weight W.

ROUGE-S ROUGE-S measures the overlap of skip-bigrams (2-grams with a specified gap S) between the summary and the reference text. This variation allows certain tokens to be skipped.

ROUGE-SU ROUGE-SU extends upon ROUGE-S by including unigram (1-gram) matches which function as a counter.

All ROUGE metrics consist of three key values: Precision, Recall, and F1-Score, each ranging from 0 to 1. These three values are calculated as follows:

$$Precision = \frac{\#Overlaps}{\#Tokens\ in\ Summary} \tag{2.2}$$

$$Recall = \frac{\#Overlaps}{\#Tokens\ in\ Reference} \tag{2.3}$$

$$F1 - Score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{2.4}$$

3 Data Curation

3.1 Data Sets

3.1.1 prompts_train.csv

The prompts_train.csv consists of 4 columns:

- **prompt_id:** The unique ID of the prompt which links to the summaries file.
- **prompt_question:** The specific question the students are asked to answer.
- **prompt_title:** A short-hand title for the prompt.
- **prompt_text:** The full prompt text.

This dataset only contains 4 entries, since there are only 4 different reference texts.

3.1.2 summaries_train.csv

The summaries_train.csv consists of 5 columns:

- **student_id:** The unique ID of the student writer.
- **prompt_id:** The unique ID of the prompt which links to the prompt 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.

This dataset contains over 7'000 entries with varying content- and wording scores.

3.2 Data Quality Assessment

The given data was of high quality because it was already cleaned up by the hosts of the competition, thus there were no empty or invalid entries. Data augmentation, using back-translation and paraphrasing techniques, did not work for this project since it was impossible to predict valid and logical content and wording score for the augmented student summaries. The most interesting features for predicting our target variables were the prompt texts, student summaries, and prompt questions. The IDs were used to merge the prompts and summaries datasets. This was done to just have one final training dataset.

3.2.1 Data Overview and Plots

The distribution of the content score, as seen in Figure 3.2, is evenly right-skewed while the distribution of the wording score, as seen in Figure 3.1, shows spikes in certain score ranges. The cause of those spikes occuring has not been determined, as no clear similarities between student summaries within that scoring range were found.

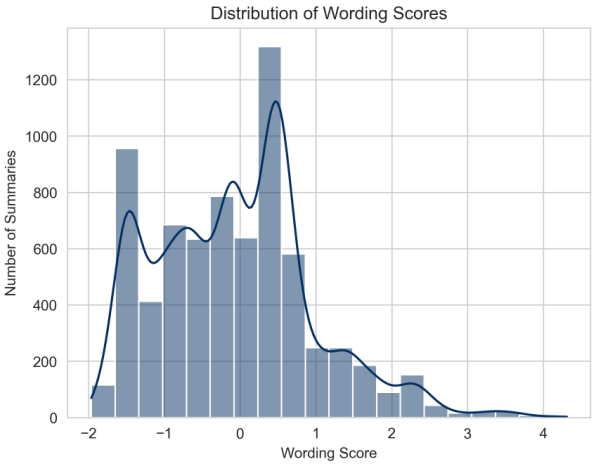

Figure 3.1: Distribution of the training set wording scores. It is unclear if this represents a multimodal distribution or if these are just outliers.

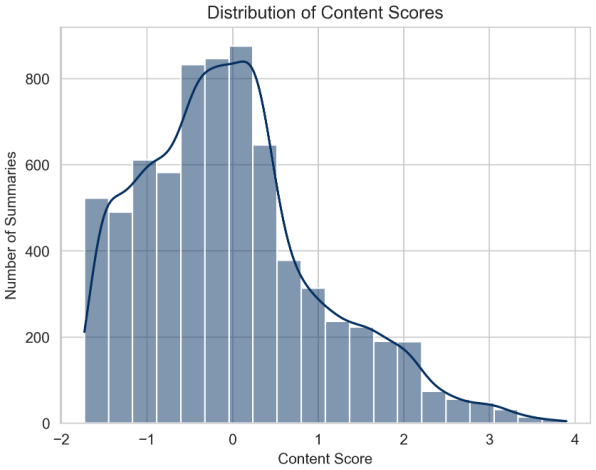

Figure 3.2: Distribution of the training set content scores. Nonetheless, a positive skew can be observed. It is unclear as to why so many summaries have been graded this way.

Due to the fact, that most wording and content scores are under 1, it has been assumed that both scores were graded very strictly by the experts. It has to be mentioned, that those are normalized scores and when transformed back, the distribution stays the same. Consequently, the team also assumed that any machine learning models would give scores in similar ranges, due to those models being trained on those ranges.

It was also assumed, that the test set on Kaggle has a similar distribution, though this could not be verified as the team did not have access to the final test data.

Additionally, the number of student summaries for every reference topic has been analyzed as well. As seen in Figure 3.3, student summaries about the topic "The Third Wave" are underrepresented in this dataset, however this did not matter as the final test set contained hundreds of different topics.

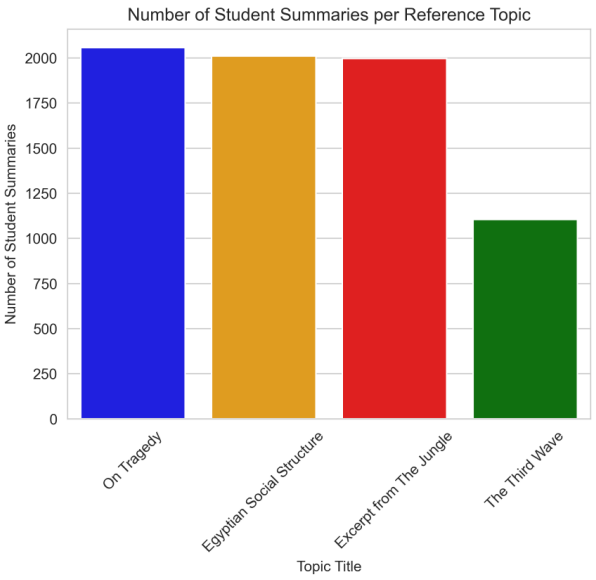

Figure 3.3: Occurences of the four different original texts the students had to summarize. As can be seen, the topic "Third Wave" is underrepresented in the training dataset. It is unclear from the challenge if this topic was not as popular, only available to higher grades or only used for a smaller cohort.

The length ratio between student summaries and reference texts in relation to both scores were plotted as well. For some reference text topics, like 'On Tragedy' or 'Egyptian Social Structure', there were certain instances, where the length of the summary turned out to be longer than the reference text itself, despite the objective of this task being summarization. See Figure 3.4 and Figure 3.5 for the mentioned plots.

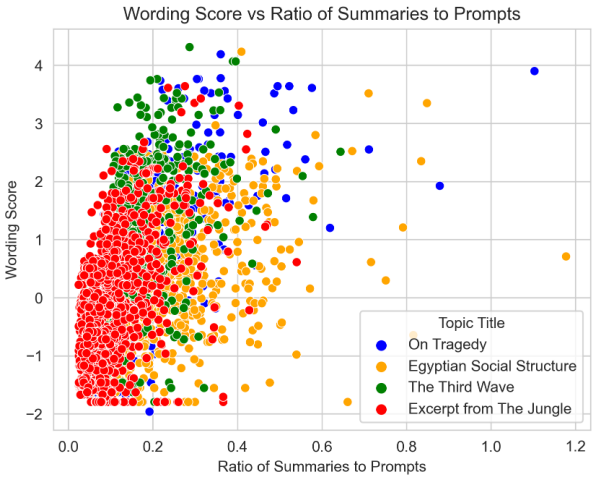

Figure 3.4: Wording scores in relation to the length ratio of student summaries and reference texts. It can be observed that some summaries about the topics 'On Tragedy' and 'Egyptian Social Structure' are longer than the reference texts. It is unclear as to why those are graded this highly, despite the goal being summarization.

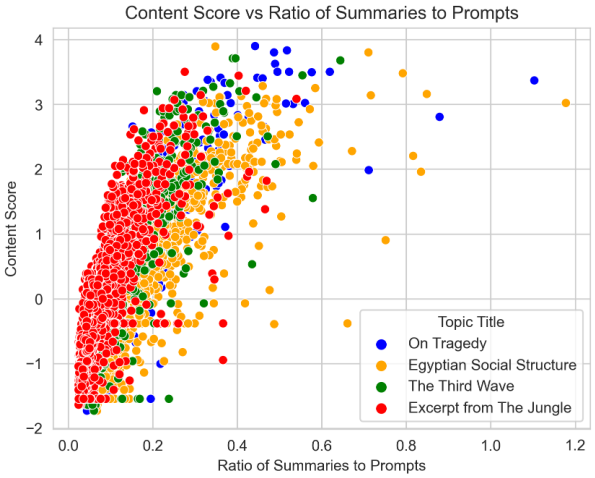

Figure 3.5: Content scores in relation to the length ratio of student summaries and reference texts. Just like the wording scores, it is unclear as to why so many student summaries about 'On Tragedy' and 'Egyptian Social Structure' have been graded this highly. Additionally, the data points are less distributed than for the wording scores.It is believed, that this is because longer student summaries are able to convey an idea better than shorter ones.

Finally, the content and wording scores have been plotted against each other to see if there is a correlation between them. As evident in Figure 3.6, there seems to be some correlation between the two. Calculating the Pearson correlation coefficient between wording and content scores gave a coefficient of 0.75 suggesting a moderate correlation. This indicates that the two scores influence each other in some way and thus the assumption was made, that this may be due to the grade level of the students, as students in a higher grades are more likely to get better scores and vice-versa since experts graded students irrespective of their grade level as mentioned in chapter 1.

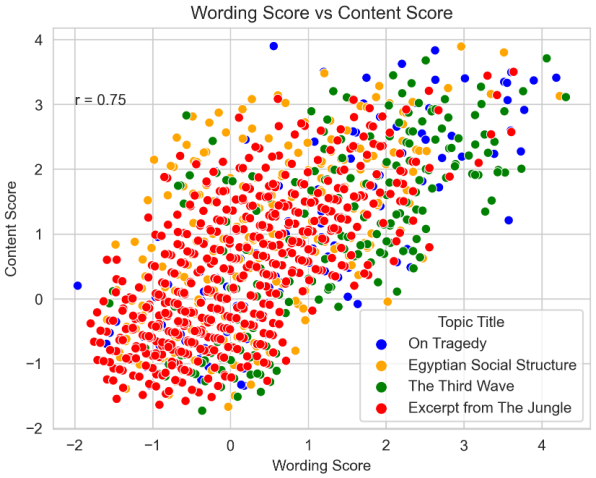

Figure 3.6: A moderate correlation between content and wording scores indicating a moderate amount of influence between both. Calculating the Pearson correlation coefficient gives a score of 0.75 suggesting a moderate correlation.

4 Modeling

4.1 Models

4.1.1 ROUGE-Based Model

The ROUGE-Based Model is a simple Neural Network utilizing ROUGE scores as its inputs. It was designed to predict the content score based on the summary and the reference text. While it does also predict the wording score, it does not perform as well on it.
The inspiration for this model arose during the exploration of existing methods for text evaluation. Among other metrics, such as BLEU, METEOR and Perplexity, ROUGE stood out, as it was originally designed for assessing summaries. However, as quoted in Chapter 2.5, the ROUGE Measures compare a computer-generated summary to a human-written one. So, for this project, it was adapted by substituting the machine-written summary with that of a student and replacing the (ideal) human-written summary with the full reference text.

Model Architecture

The ROUGE-Based Model consists of two distinct steps: the calculation of ROUGE-scores and the forward pass through the neural network, as illustrated in Figure 4.1. The ROUGE-scores are computed from the tokenized reference text and summary, after their stopwords were removed. The Precision, Recall, and F1-Score from the ROUGE-scores, along with the normalized length of the summary, serve as inputs to the network. The network then outputs the predicted content and wording scores for the summary.

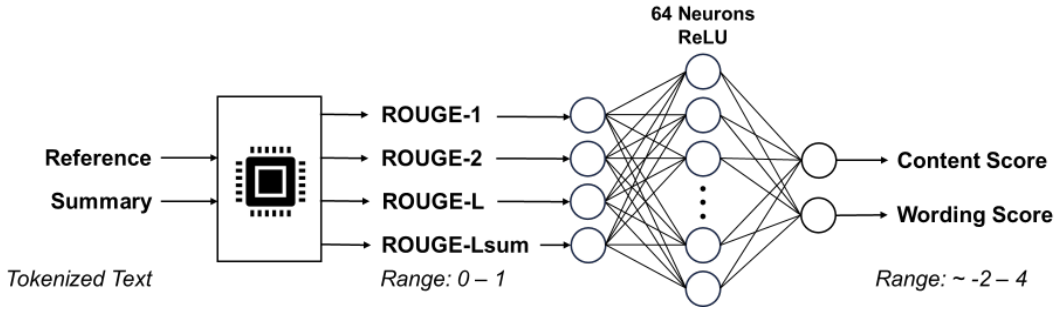

Figure 4.1: ROUGE-Based Model Architecture: The model takes the tokenized texts and calculates ROUGE scores from them. These scores get passed through a Neural Network wit a single hidden layer to predict the final scores.

Hyperparameter Tuning

During the development and training of the ROUGE-Based Model, various hyperparameters were explored and tuned using the Optuna[1] library. It was found, that removing stopwords and applying stemming or lemmatization had little to no impact on the models performance. Therefore stopwords were removed to reduce the number of tokens when calculating the ROUGE scores and neither stemming nor lemmatization were applied, as they were unnecessary. For the models inputs, specific ROUGE scores were used, including ROUGE-1, ROUGE-2, ROUGE-L, and ROUGE-Lsum. These scores were calculated using the Python library 'rouge-score'.[2] It's worth noting that later in the project, another library called 'rouge-metric'[3] was found, which could calculate the additional glsrouge scores. However, considering the likely marginal gains and to maintain consistency, it was decided not to re-implement the model with the new library.
Given the nature of this model and the task, where the network primarily maps the ROUGE scores (ranging from 0 to 1) into the content and wording scores (ranging from -2 to 4), the NN was quite robust against overfitting. Changes in the number of hidden neurons and hidden layers (as log as there was at least one hidden layer) had minimal impact on performance. Notably, the selection of the activation function played a bigger role; logistic functions like Sigmoid and tanh demonstrated poorer performance. However, ReLU and Leaky ReLU showed better results, ultimately leading to the adoption of ReLU in the final model.

- **Stopword Removal:** Stopwords were removed.
- **Stemming & Lemmatization:** Neither stemming nor lemmatization was used.
- **ROUGE Metrics:** ROUGE-1, ROUGE-2, ROUGE-L, ROUGE-Lsum
- **Hidden-Layers:** 1 Hidden Layer
- **Hidden-Dim:** 64 Neurons
- **Activation Function:** ReLU
- **Learning Rate:** 0.01
- **Epochs:** 10

Training

The models NN is initialized with random weights. Before training, it is also given the mean length (number of characters) and the standard deviation in length of the summaries in the training set. These two additional values are used to normalize the lengths of summaries,

[1] Optuna - Website
[2] rouge-score library - PyPI Website
[3] rouge-metric library - PyPI Website

when passing them to the network. Since there is no upper bound for how long a summary can be, Z-Score Normalization was applied:

$$z = \frac{x - \mu}{\sigma} \tag{4.1}$$

Where z is the normalized length x of a summary. μ is the mean length of the summaries in the trainings set and σ the standard deviation.
Since the ROUGE metrics have no learnable parameters, the python library 'rouge-score' can be used to precalculate these scores prior to training. These precalculated scores are used as inputs to train the network, using the Mean Squared Error (MSE) loss function over 10 training Epochs.

Limitations

It's important to note that this model only receives the summary and the reference text the summary was based on. It does not consider the specific task asked of the students. In cases where the prompt asks for specifics on one part of the text, a summary focusing on that prompt may be evaluated as worse than a summary that stays more general.
The model also doesn't know what words were used, only whether they overlap with the reference. Therefore, it cannot determine whether objective and neutral language was used or how well lexis and syntax was used. Due to these limitations it was expected, that this model performs better on the content score, and poorly on the wording score. (Also as evident by the correlation matrix; Figure 4.2.)

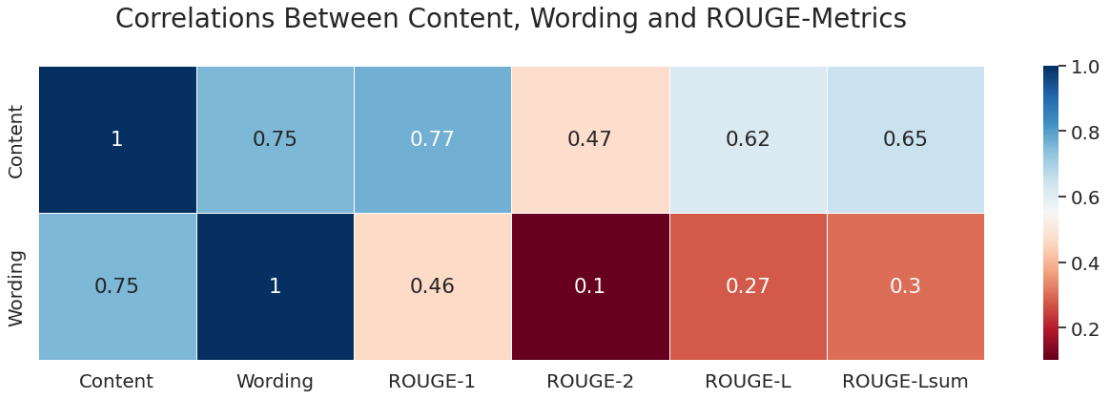

Figure 4.2: Correlations between Content and Wording scores and the used ROUGE Metrics. The higher correlation between the Content and ROUGE Meitics suggest that the ROUGE-Based Model will perform better on that metric.

4.1.2 Transformer

Motivation

In the past few years, transformers have risen in popularity in the NLP space. Student summaries have mostly long sequence lengths and require an efficient and capable architecture to fit the data. Transformers are a perfect fit for the challenge at hand.

Transformer selection

When choosing the most fitting and optimal transformer multiple factors must be taken into consideration. This would be the type of transformer for instance or how the attention mechanism was implemented. Nevertheless, from just observing the desired result, which is predicting two continuous values, several factors can already be decided on. Encoder-only models were chosen because there is no need for decoders, as there is nothing to decode. From that observation, the team searched for suitable models on HuggingFace[4], a platform listing various transformer implementations. The search focused on encoder-only models, sorted by popularity. Additionally, the complexity of the transformers was considered, along with trials of both uncased and cased models to assess their impact on performance. All tested implementations were variations of BERT, primarily due to the team's limited familiarity and expertise with non-BERT-based models.

The best performing implementation was found as follows: Since the public data only consists of four possible topics and the private data set of an unknown number of topics, a great importance on generalization was set. Due to this, the cross-validation technique Group-K Fold was used. With this method the generalization performance can be estimated on unseen topics. The models are trained on the training set using the HuggingFace Transformers library and the RMSE of the content and wording score, and the combination of MCRMSE were used to compute the metrics:

```
def compute_mcrmse(eval_pred):
    preds, labels = eval_pred

    col_rmse = np.sqrt(np.mean((preds - labels) ** 2, axis=0))
    mcrmse = np.mean(col_rmse)

    return {
        "content_rmse": col_rmse[0],
        "wording_rmse": col_rmse[1],
        "mcrmse": mcrmse,
    }
```

[4] HuggingFace - Website

When running the test all models had the exact same hyperparameter configuration:

- **Input:** The tokenized padded or truncated student-written summary.
- **Max length:** 512. This was chosen since most of the BERT models have 512 as maximum length.
- **Batch size:** 8
- **Epochs:** 8
- **Learning rate:** 0.0001
- **Weight decay:** 0.01

NOTE: Hyperparameters which are not listed above use the default settings from the HuggingFace Library.

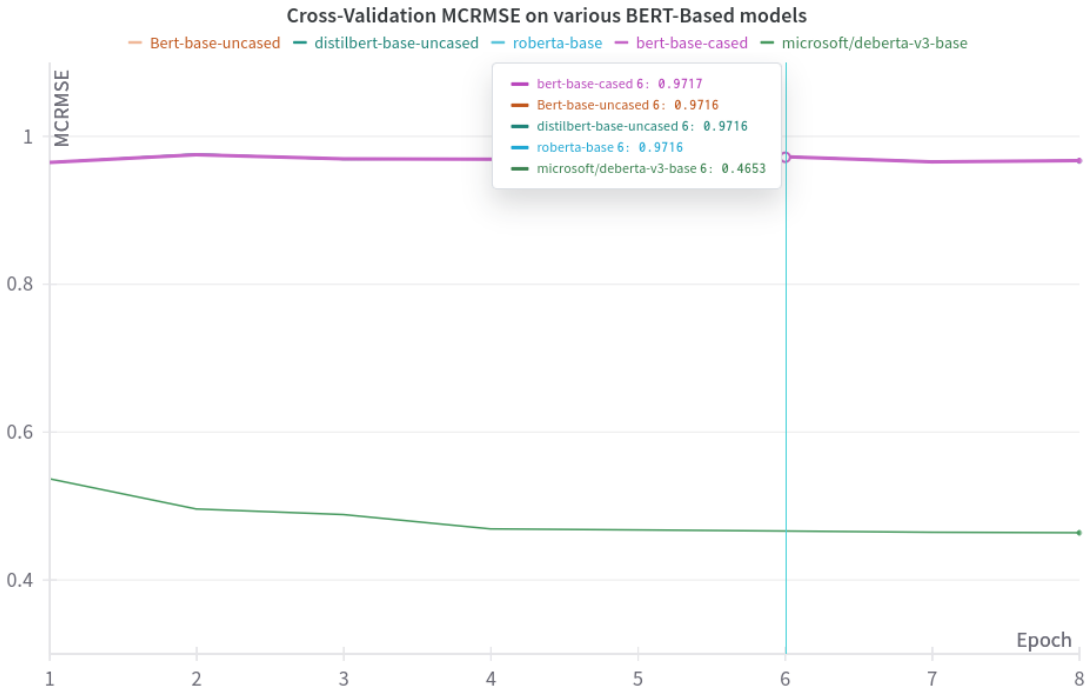

Figure 4.3: Cross-Validation MCRMSE Score Evolution on various BERT-Based models

The above visualization distinctly highlights a standout performer. All BERT-based models, except for DeBERTa fail to generalize effectively. It is noteworthy that the performance metrics of models other than DeBERTa are quite similar, suggesting minimal variance among them. Additionally, the comparison of uncased and cased BERT-based models indicates that

the casing does not significantly influence performance outcomes. A possible reason for DeBERTa outperforming, and also what it makes it different compared to the other BERT-Based models, is the disentangled attention mechanism.

Prompt Engineering

When using transformers, using an optimized prompt has been proven to achieve better results. Various experiments using different combinations of features from the data were conducted. A logical combination of the features would be to use the student-written text together with the prompt text. After further experimenting, adding the prompt question and using a natural language style prompt improved performance.

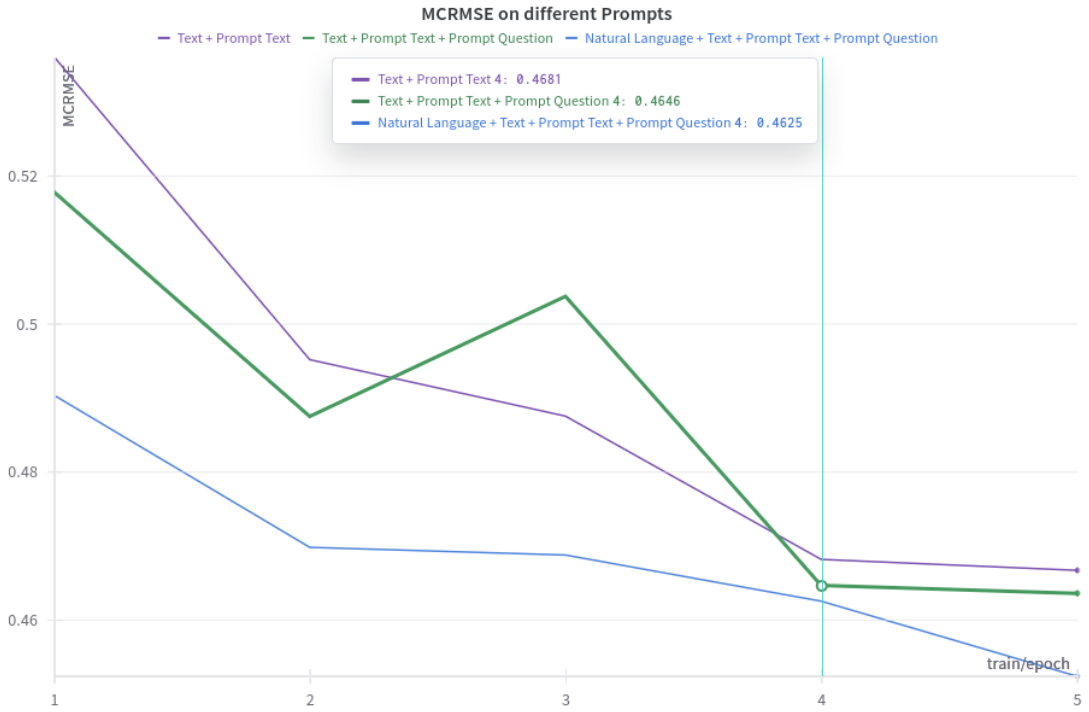

Figure 4.4: MCRMSE Score Evolution on different Prompts

By using a natural language style prompt, verbs and objectives were used to specify the goal of the problem at hand. The final prompt the transformer uses is the following:

> "Evaluate the content and wording score of this summary: [SEP] **text** [SEP] The summary must answer the following prompt: [SEP] **prompt question** [SEP] The prompt is related towards the following original text: [SEP] **prompt text**"

Model Architecture

The model uses three features from the dataset which are the student-written summary, the prompt text, and the prompt question. These get formatted into the aforementioned prompt and tokenized. The tokenizer pads or truncates the prompt to the max length. The network then outputs the content and wording scores for the summary.

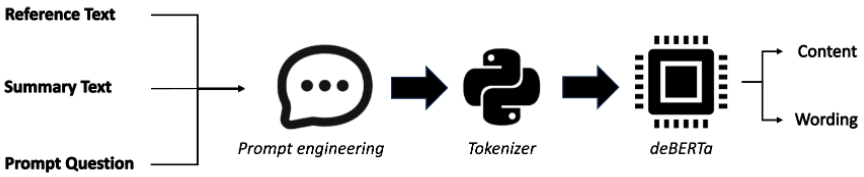

Figure 4.5: Transformer Pipeline

Hyperparameter tuning

Layer fine-tuning

DeBERTa contains twelve encoding layers. The Transformer model was downstreamed by freezing the bottom N layers. After iterating and freezing each layer, the best performing iteration was freezing at layer 9:

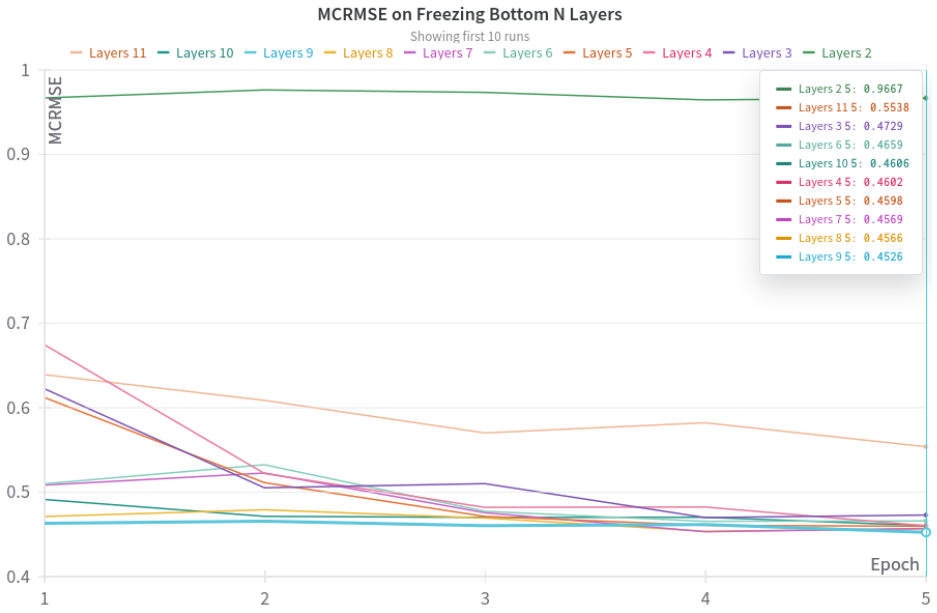

Figure 4.6: MCRMSE Score Evolution on freezing N Layers

Max length fine-tuning

Configuring the max length was essential, because if the max length is too short it could cut out important information of the prompt. A logical conclusion would be that a higher max length would lead to better performance. However, sequence lengths over 1024 led to worse performance. It is speculated that the text of the prompt exceeding 1024 tokens may become overly detailed, potentially impacting the results negatively.

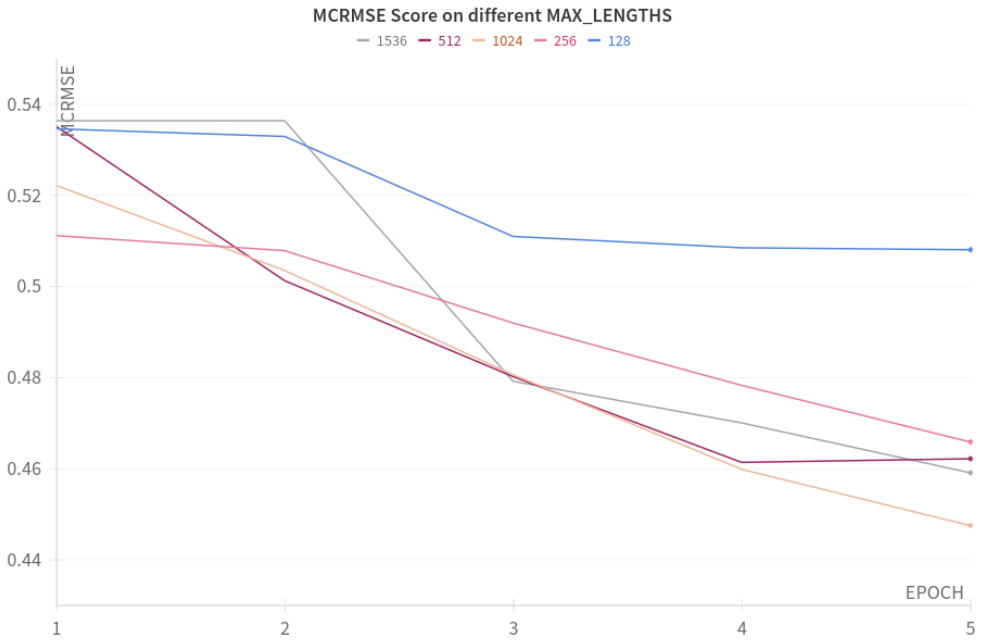

Figure 4.7: MCRMSE Score Evolution on different Max Lengths

Hyperparameter configuration

Below are the hyperparameters that were tuned and an explanation why the specific values were chosen.

- **Max length:** 1024. Best performing max length.
- **Batch size:** 4. Maximum allowed batch size due to hardware limitation.
- **Epochs:** 4. MCRMSE Score converges at four epochs. Also, having too many epochs could lead to overfitting.
- **Learning rate:** 0.0001. Automatically fine-tuned using optuna[5].
- **Weight decay:** 0.01. Automatically fine-tuned using optuna.

[5]Optuna - Automatic Hyperparameter Optimizer

- **Hidden dropout prob:** 0.07. Automatically fine-tuned using optuna.
- **Attention probs dropout prob:** 0.07. Automatically fine-tuned using optuna.

NOTE: Hyperparameters which are not listed above use the default settings from the HuggingFace Library.

Transformer result

As seen in Figure 4.8, the performance on the content score is high. However, improvements can be made on the wording score.

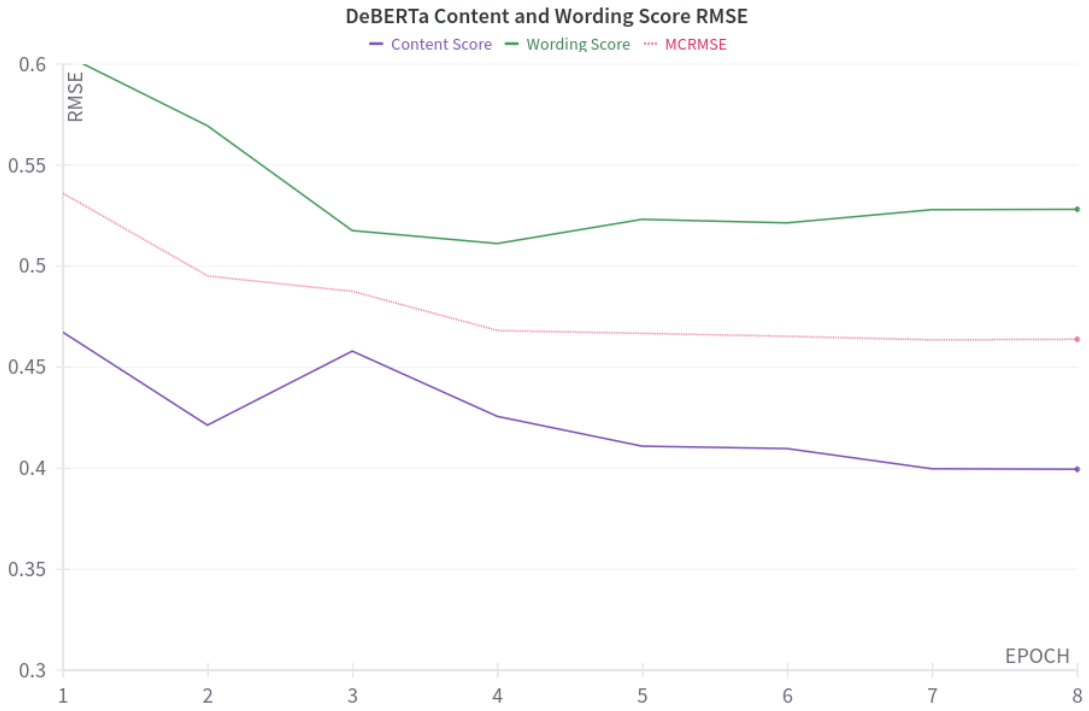

Figure 4.8: Deberta Performance on Wording and Content

4.1.3 LightGBM

Motivation

Being a GBDT-based model, LightGBM models are highly accurate without the need for powerful GPUs. Amongst contestants of this and various other Kaggle challenges, this machine learning algorithm is one of the most popular ones, often achieving the top rank on the Kaggle leaderboards of said challenges. Given that the prediction values, 'content' and 'wording', are continuous, numeric values it made sense to use a regression-based approach as well. However, unlike other approaches used in this project, this approach requires more preprocessing and engineering of the data to get the best results.

Preprocessing

Like other traditional machine learning approaches, LightGBM does not directly work with text data. Thus features like the student summaries themselves, could not be used as an input. Instead, the creation of new numeric features containing different information about the summaries and the prompts was necessary. Many of the new features in the final dataset were created using a library called 'Textstat' (Shivam Bansal, 2015). The Textstat library calculates different statistics from text. It helps determine readability, complexity, and grade level of the given student summaries.
Additionally, the number of overlapping N-grams between a reference text and a student summary were calculated and a separate ROUGE-score was calculated as well. The motivation in using Textstat came from a discussion of a previous CommonLit challenge called the "CommonLit Readability Prize". The notebook on that discussion can be found under Yhirakawa, 2021. By using this library, the wording score improved compared to the other models used in this project.

Architecture

The LightGBM model only uses the student summaries and the reference texts. New features are created from those two input features using Textstat, as described above. The data then gets split into three sets: 80% for training, 10% for validation and 10% for testing. This choice was made due to this split being very common in the field of machine learning. The hyperparameters were tuned on the validation set using Optuna and finally everything was evaluated on the test set. The final output consisted of the wording and content scores of all the student summaries. A graphical representation of this process can be found in Figure 4.9.

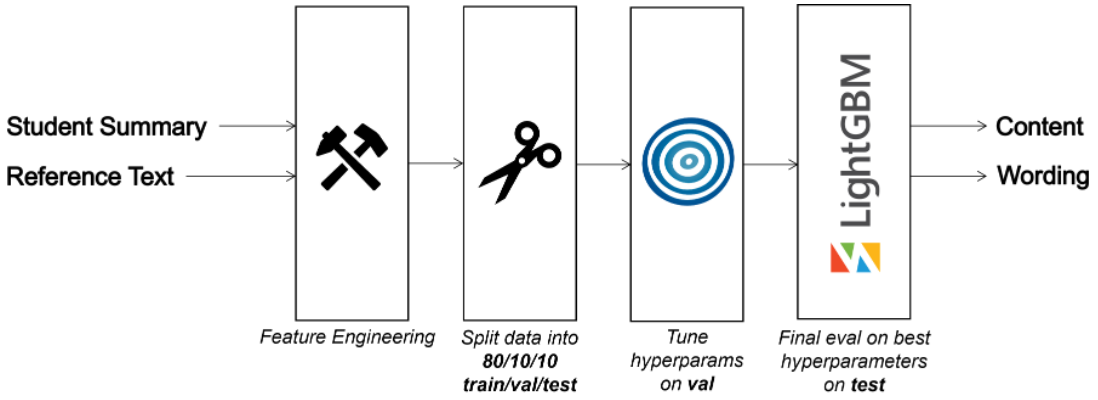

Figure 4.9: Pipeline of the LightGBM model, the feature engineering was done using the aforementioned Textstat library and the hyperparameters were tuned with the help of Optuna

Hyperparameters

For the LightGBM model, Optuna was used. Optuna allows for efficient hyperparameter tuning by optimizing an objective function. Additional details can be found by referring to the original article under Akiba et al., 2019.

These are the final hyperparameters that have been used for this model:

- **Number of Estimators :** 778
- **Learning Rate :** 0.015
- **Colsample Bytree :** 0.761
- **Reg Alpha :** 0.002

Those final hyperparameters have been used after using Optuna to determine the importance of every hyperparameter and only using those four, which had the highest importance. This was done to maximize the hyperparameter search speed for any subsequent training. Any definitions can be found in the glossary.

Results

The final MCRMSE scores of the LightGBM model were as follows:

- **Content RMSE:** 0.66
- **Wording RMSE:** 0.51
- **MCRMSE:** 0.59

This model performs better on the wording score than the other approaches in this project. This was one of the reasons for choosing an ensemble-based approach to improve the final score.

4.1.4 Ensemble

The Ensemble Model merges the predictions from the three models above: the ROUGE-Based Model, the LightGBM Model and the Transformer, as shown in Figure 4.10. Each of these models independently predicts both content and wording scores, which the Ensemble Model combines into the two final scores.

Model Architecture

The summary, reference text, and the prompt are provided as raw text inputs for this ensemble. Each of the three individual models perform their own preprocessing steps on the texts. (These steps are explained in their respective sections above). Following preprocessing, the models make their predictions, which the ensemble aggregates using a single linear layer, down-projecting them into the two final predictions. The linear layer is trained to predict scores based on the outputs of the three models, effectively learning how to weigh each model's contribution. Instead of a simple (weighted) average, the ensemble learns to leverage the strengths of each model: the ROUGE-Based Model performs well on the content score, LGBM on the wording score, and the Transformer is a great generalizer that also takes the specific question of the task into consideration.

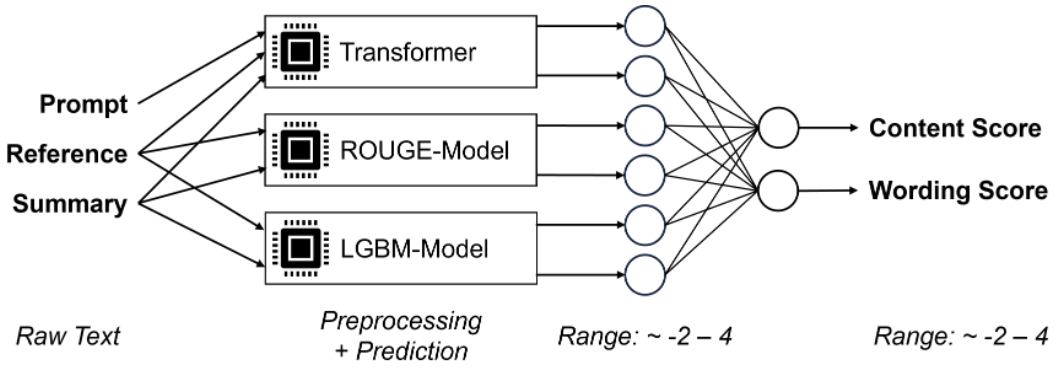

Figure 4.10: Ensemble Model Architecture: Three different models preprocess raw text inputs independently, make predictions, and contribute to the final scores through a trained linear layer.

Hyperparameter Tuning

The "learned weighted average" approach was the only one investigated for this model. Since the model consists of just one linear layer, there were only a few parameters to test. While experimenting with hidden layers for this model, it became clear quite quickly that the

added non-linearity, no matter the activation function, reduced performance. However, like the ROUGE-Based Model, the simplicity meant that the Ensemble Model was robust against overfitting as well as changes to the Learning rate and the number of training Epochs. Leading to the following hyperparameters:

- **Hidden-Layers:** 0 Hidden Layers
- **Hidden-Dim:** 0 Neurons
- **Activation Function:** No Activation Function
- **Learning Rate:** 0.01
- **Epochs:** 10 Epochs

4.2 Model Evaluation

4.2.1 Results

The performances of the developed models were assessed using the MCRMSE. The final Kaggle scores, representing the Ensemble Models' performance and generalization capabilities, are as follows:

- **Final Public Score:** 0.4956
- **Final Private Score:** 0.4977

Furthermore, the Ensemble Model was evaluated on a separate test set, with scores during training and testing phases:

- **Final Training-Set Score:** 0.3687
- **Final Test-Set Score:** 0.4981

Evolution of Kaggle Scores

The following graphs (Figure 4.11 and Figure 4.12) illustrate the evolution of Kaggle scores throughout the development process. Some models, like the RNN and LSTM are not listed as they were not tested thoroughly, and those with multiple entries have some variation in their hyperparameters. The full range and a clipped version of the graph - highlighting the lowest (best) scores - show the progress of the models and their performances.

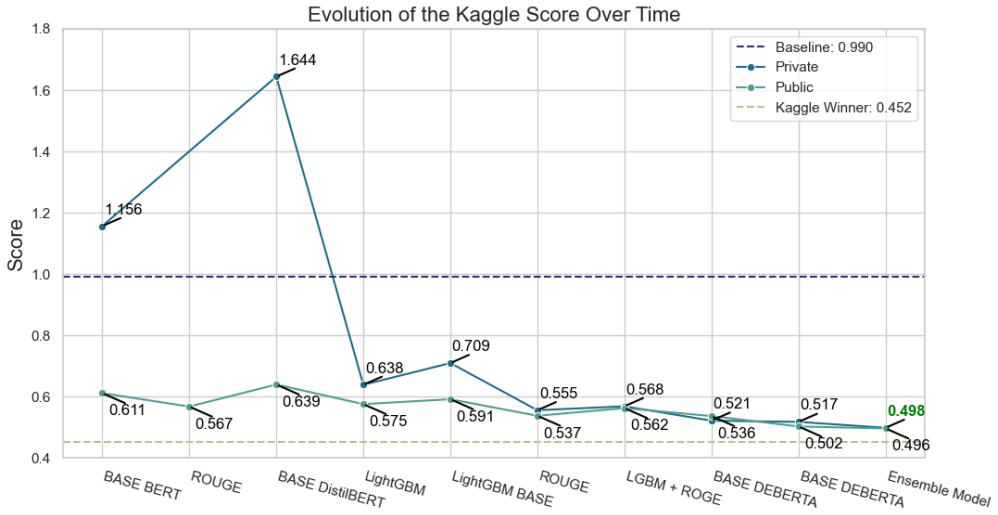

Figure 4.11: Kaggle Score Evolution over time. Models with repeating entries have different Hyperparameters.

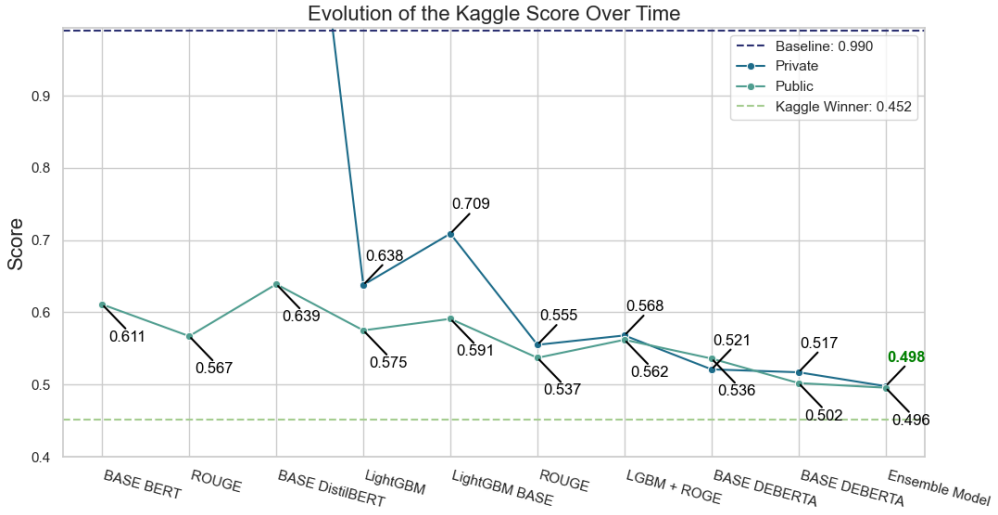

Figure 4.12: Kaggle Score Evolution over time. (Clipped at the Baseline score, to highlight the best scores.) Models with repeating entries have different Hyperparameters.

4.2.2 Error Analysis

The final evaluation of the project involved assessing the performance of the ensemble model on the test set, which was 20% of the provided data. The final score was 0.498, consistent with the scores achieved in the Kaggle competition, suggesting a representative outcome.

Error distribution

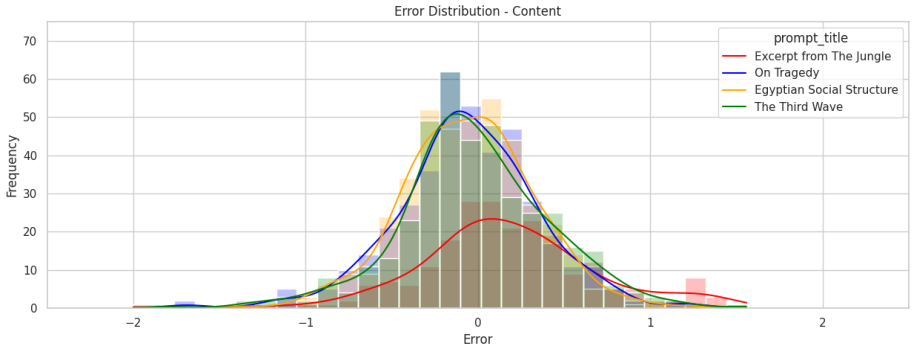

Figure 4.13: Error Distribution - Content

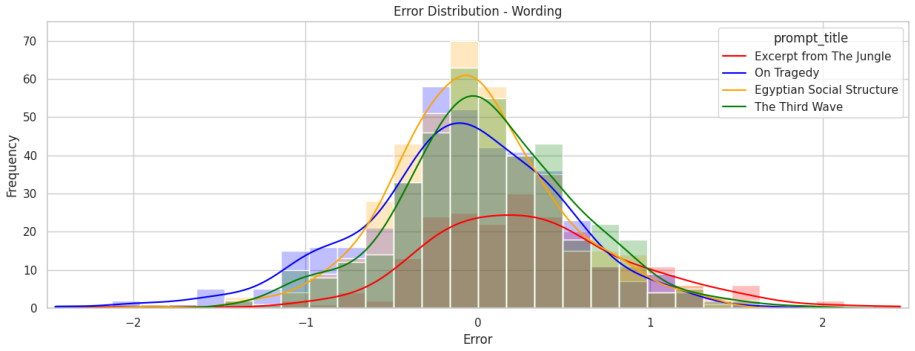

Figure 4.14: Error Distribution - Wording

The analysis of the error distribution for content and wording scores, as illustrated in Figures 4.13 and 4.14, reveals a gaussian distribution peak around zero, with most errors being smaller than 1. This suggests that the model's errors are generally small in magnitude. The even distribution also shows that the model does not tend to predict to high nor too low.

Mean absolute error

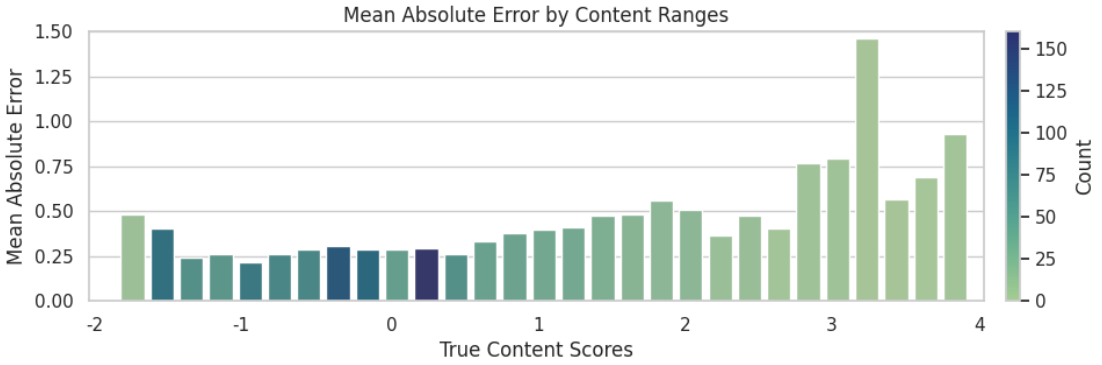

Figure 4.15: Mean Absolute Error - Content

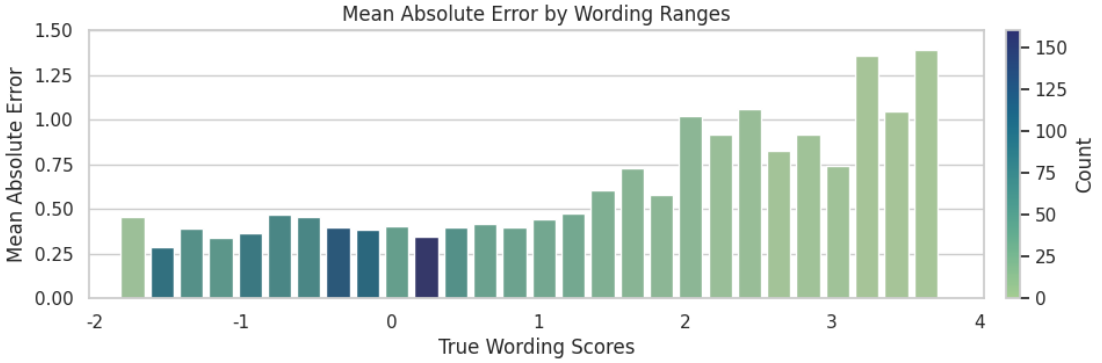

Figure 4.16: Mean Absolute Error - Wording

Displayed above are plots illustrating the Mean Absolute Error (MAE) for the true content and wording scores. (Figure 4.15 and Figure 4.16) These plots use color intensity to represent data density: lighter shades indicate less training data in the dataset. These plots reveal a tendency for higher MAEs in areas with lighter colors, suggesting that the model has difficulties with less frequent data. It can be inferred that an increase in data quantity within these lighter ranges would likely enhance the models performance.

4.3 Discarded Approaches

4.3.1 RNN & LSTM

Since RNNs and LSTMs are specifically designed for processing sequential data, making them particularly effective for handling text and tasks involving natural language, they served

as ideal starting points for early prototyping.

RNNs and LSTMs formed the initial approach, primarily exploring the wording metric, in the hopes that these models could assess the use of objective language and evaluate the use of lexis and syntax. This approach also allowed the team to get comfortable with the dataset, preprocessing techniques, as well as model implementation and training with PyTorch[6], tracking with Weights & Biases (WandB)[7] and a models evaluation processes. However, this first approach was discarded in favor of other models that had better performance for comparable efforts.

Model Architectures

Both the RNN and LSTM models used a similar bidirectional architecture featuring a linear classifier as output. Stopwords were removed from the summaries, the remaining words were uncapitalized and stemmed to their base forms. Each of these word-tokens were vectorized using Word2Vec. These encoded strings then got passed into the model sequentially. After processing each token, the classifier generated the prediction for the wording score of that summary.

Hyperparameters

- **Model Type:** Whether the Model is an RNN or an LSTM.
- **Embedding-dim.:** Dimensionality of the word embeddings.
- **Hidden-dim.:** Dimensionality of the hidden layers.
- **Num Layers:** Number of layers in the RNN or LSTM.
- **Epochs:** Number of training Epochs.

Limitations

These models struggled with overfitting. Notably, RNNs achieved a suspiciously impressive RMSE score of 0.03 for wording on the training set. This success did not generalize well to the validation set, where RMSE scores got as high as 2.25, well above the baseline of 0.99. The complex nature of the natural language in the summaries posed challenges for these models. Also, considering that the models only looked at the summaries themself, without comparing them to the reference texts, it was decided to no longer pursue this approach.

[6] PyTorch - Website
[7] Weights & Biases - Website

4.3.2 DeBERTa Large

After having success with the DeBERTa Base model, there was hope that using the DeBERTa Large model would lead to a better performance. However, after experimenting with the large model there was an overfitting problem. The initial intention was to allocate more time to solving this problem. However, the extensive number of parameters resulted in longer training times. Additionally, hardware limitations and time constraints necessitated the decision to discard this approach. Potentially, with extended time and enhanced computing power, better performance could be achieved.

5 Conclusion

Overall, the team is satisfied with the project result. This was the first Kaggle challenge for all four members. The team was able to place 975th out of a total 2065 participants, with a private score of 0.4977 on the already finished Kaggle competition leaderboard:

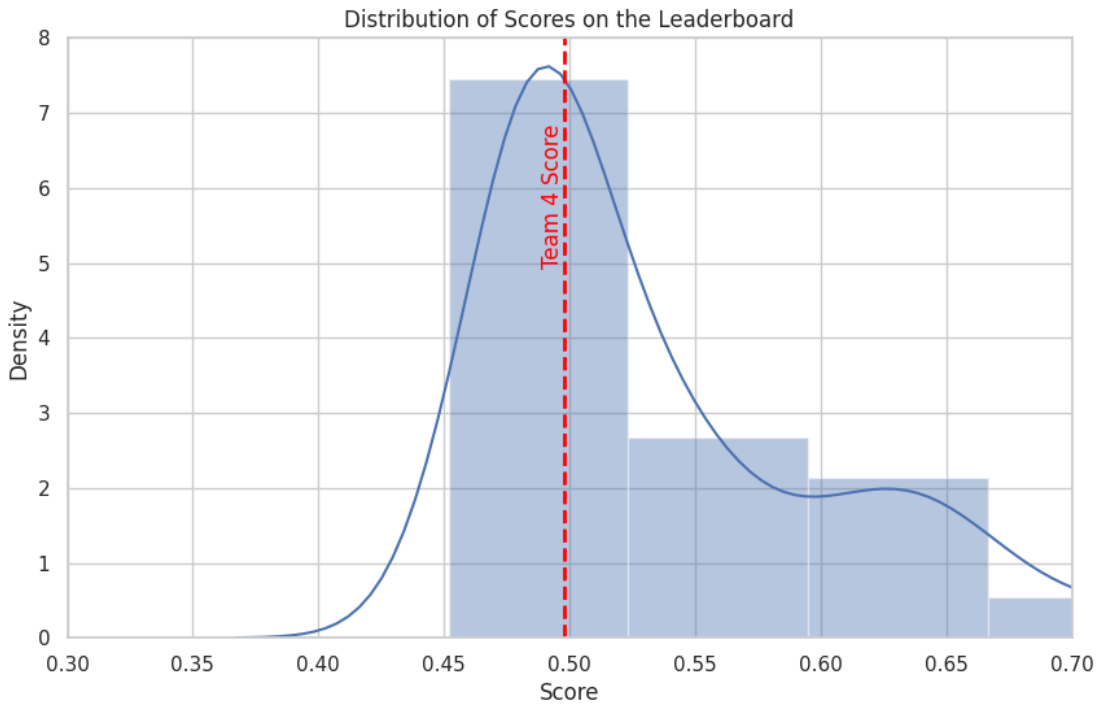

Figure 5.1: Kaggle Score Distribution

When looking at the winning solution, it became apparent that developing a similar approach would be challenging due to the complexity and depth of the winner's strategy. In summary, the leader used Large Language Models with specific prompts to augment data, coupled with a four-fold transformer ensemble.

With the absence of time constraints and hardware limitations, the team could have explored additional approaches. This might have involved using a larger DeBERTa model or dedicating more time to the development of the ensemble. Nevertheless, the achieved results are considered satisfactory, given the unique solution employed. The solution combines a deep-

learning approach such as transformers and a linguistic approach such as ROUGE-Based Model and LightGBM with linguistic features.

5.1 Challenges and Lessons Learned

At the beginning of the project, none of the team members had completed the Natural Language Processing course. Instead, the NLP course was taken in parallel with the AI Challenge course. This made it especially difficult to work with transformers, since they are quite an large topic to understand. While on the topic on transformers, they are computationally intensive models and, even with the provided hardware from the university, computing limitations were still an issue. This was not the only challenge; the team also never worked with or implemented an ensemble. Nevertheless, with these challenges the team has definitely learned a lot in the field of NLP and how machine learning in general can be used in real life settings.

6 Reflection

At the beginning of the challenge there was not a lot of motivation coming into the project. When selecting the competition, we did not find a topic that seemed interesting to us and just settled for "Evaluate Student Summaries". But as time went by, we grew to enjoy the challenge, especially after implementing our own ideas to solve the problem. Communication within the team and with our coach was a strong point. We consistently received valuable feedback and could freely discuss machine learning queries. Our approach to planning was systematic, with weekly goals and milestones that kept us on track. Notably, our presentation skills noticeably improved with each successive presentation.

At the end of the day, we feel that the AI Challenge course served as a good preparation for the upcoming Bachelor Thesis and are thankful for the opportunity as we are now more confident to write and present any technical project.

Glossary

activation function A non-inear function which calculates the output of a neuron. 3, 14, 25

Batch size Specifies the number of data samples processed simultaneously in one iteration. 17

BLEU Bilingual Evaluation Understudy. 13

Colsample Bytree Fracture of features used to train model (0.75 for example means that 75% of the features are used for training). 23

EFB Exclusive Feature Bundling. 5

Epoch Denotes the number of times the entire training dataset is passed through the model. 14, 15, 17, 25

GBDT Gradient-Boosted Decision Trees . 5

GOSS Gradient-based One Side Sampling. 5

hidden layer The layer(s) of neurons that are between the input and output layers. 14, 25

hidden-dim The number of neurons in a hidden layer. 14, 25

LCS Longest Common Subsequence. 6

Learning Rate Controls the learning speed. 23

Learning rate Controls the size of the steps taken during the optimization process. 14, 17, 25

lemmatization Breaking down a word to its root meaning. 14

LSTM Long Short-Term Memory. 3, 28, 29

MAE Mean Absolute Error. 28

Max length Upper limit on the number of tokens of sequence. 17

MCRMSE Mean Columnwise Root Mean Squared Error. 2, 16, 23, 25

METEOR Metric for Evaluation of Translation with Explicit Ordering. 13

MSE Mean Squared Error. 15

n-gram A contiguous sequence of n items (words, characters, etc.) in a text. 6

NN Neural Network. 3, 13, 14

Number of Estimators Controls the number of decision trees in the model. 23

Perplexity A quantitative measure that assesses how well a language model predicts a given set of text. 13

Reg Alpha Used to penalize features. 23

RMSE Root Mean Squared Error. 2, 16, 29

RNN Recurrent Neural Network. 3, 28, 29

ROUGE Recall-Oriented Understudy for Gisting Evaluation. 6, 13–15, 22, 32

skip-bigram A pair of items in a sequence where some elements are skipped. 6

stemming Reducing a word to its word stem. 14

token Individual units, such as words or subwords, extracted from a given text or sequence. 4, 6, 13, 14, 19, 20, 29

Weight decay Regularization technique that applies a penalty to the magnitude of the model's weights. 17

Word2Vec A technique for obtaining vector representations of words. 29

List of Figures

Table of Equations

Bibliography

Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2623–2631. https://doi.org/10.1145/3292500.3330701

He, P., Liu, X., Gao, J., & Chen, W. (2021). Deberta: Decoding-enhanced bert with disentangled attention. https://github.com/microsoft/DeBERTa

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 30). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf

Lin, C.-Y. (2004). Rouge: A package for automatic evaluation of summaries. *Text summarization branches out*, 74–81.

Schmidt, R. M. (2019). Recurrent neural networks (rnns): A gentle introduction and overview. *CoRR*, *abs/1912.05911*. http://arxiv.org/abs/1912.05911

Shivam Bansal, C. A. (2015). Textstat/textstat: Python package to calculate readability statistics of a text object - paragraphs, sentences, articles. *GitHub*. Retrieved January 8, 2024, from https://github.com/textstat/textstat

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2023). Attention is all you need. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Yhirakawa. (2021). [textstat] how to evaluate readability? *Kaggle.com*. Retrieved January 9, 2024, from https://www.kaggle.com/code/yhirakawa/textstat-how-to-evaluate-readability?scriptVersionId=66150847