COMMONLIT

Evaluate Student Summaries

Jason Heesang Lee

Competition Overview

- I participated in the CommonLit – Evaluating Student Summary Competition, where the objective was to develop an automated system for evaluating the quality of summaries written by students in grades 3-12.

- This involved building a Machine Learning / Deep Learning model capable of assessing how effectively a student captures the main idea and details of a prompt text, along with the clarity, precision, and fluency of their written summary.

  A comprehensive dataset of real student summaries were given to the participants to train and refine the model.

- This is the second Natural Language Processing competition I joined after Judicial Precedent Prediction by DACON.
  For the previous competition, the names of the first and second parties and brief facts about multiple cases were given, and the competitors had to predict whether the first party had won the case or not.

- I learned the basics of the NLP technique while competing in the first competition and found my interest in Natural Language Processing. Therefore, I have decided to join another NLP competition : CommonLit – Evaluating Student Summary.

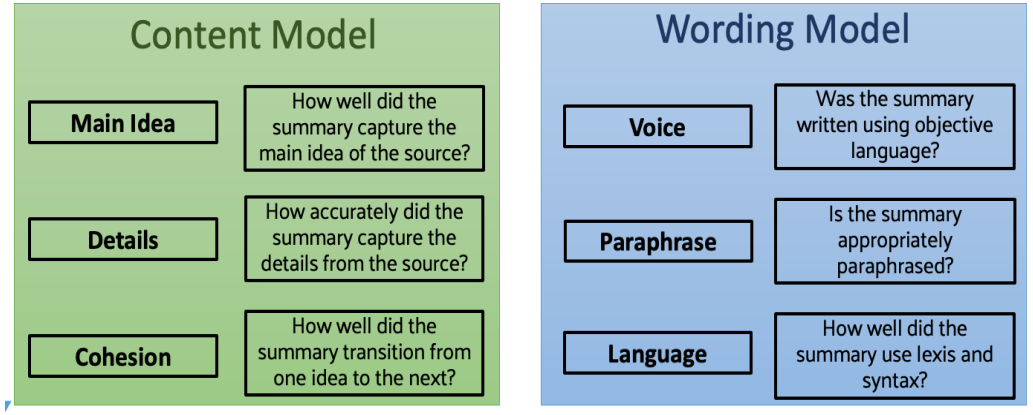
Scoring Metrics

- Submissions are scored using MCRMSE, mean column-wise root mean squared error.

  Where N_t is the number of scored ground truth target columns, and y and $\hat{y}$ are the actual and predicted values, respectively.

$$MCRMSE = \frac{1}{N_t}\sum_{j=1}^{N_t}\left(\frac{1}{n}\sum_{i=1}^{n}\left(y_{ij} - y_{\hat{i}j}\right)^2\right)^2$$

- We were asked to find content & wording scores.
- The competition host has explained the differences between these two scores in rubric items.



https://www.kaggle.com/competitions/commonlit-evaluate-student-summaries
https://www.kaggle.com/competitions/commonlit-evaluate-student-summaries/discussion/424402#2348464

Dataset

- The organizers have provided 2 different CSV files for each train & test data.
- summaries_train.csv
  - This file includes below columns:
    - student_id : The ID of the student, Unique to each student.
    - prompt_id : The ID of the prompt, Unique to each prompt.
    - text : Student composed summaries.
    - content_score : Content score, First Target.
    - wording_score : Wording score, Second Target.
- prompts_train.csv
  - This file includes below columns:
    - prompt_id : The ID of the prompt, Unique to each prompt.
    - prompt_question : The Questions unique to each prompt.
      Students have to compose summaries based on these questions.
    - prompt_title : Title of the prompt.
    - prompt_text : Full prompt text
- summaires_test.csv
  - Contains the same columns with summaries_train.
    But test.csv has no score (target) columns.
- prompts_test.csv
  - Contains the same columns with prompts_train but has only one example.
    A full test data is in the hidden dataset.
- sample_submission.csv
  - A submission file in the correct format.

summaries_train.csv (3.43 MB)

Detail Compact Column

student_id	prompt_id	text	content	wording
7165 unique values	39c16e 29% 3b9047 28% Other (3099) 43%	7165 unique values	-1.73 3.9	-1.96 4.31
000e8c3c7ddb	814d6b	The third wave was an experimentto see how people reacted to a new one leader government. It gained ...	0.205682506482641	0.380537638762288
0020ae56ffbf	ebad26	They would rub it up with soda to make the smell go away and it wouldnt be a bad smell. Some of the ...	-0.548304076980462	0.506755353548534
004e978e639e	3b9047	In Egypt, there were many occupations and social classes	3.12892846350062	4.23122555224945

prompts_train.csv (16.15 kB)

Detail Compact Column

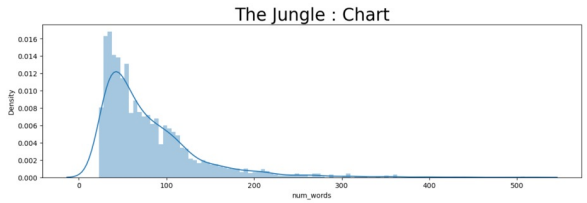
prompt_id	prompt_question	prompt_title	prompt_text
4 unique values	4 unique values	4 unique values	4 unique values
39c16e	Summarize at least 3 elements of an ideal tragedy, as described by Aristotle.	On Tragedy	Chapter 13 As the sequel to what has already been said, we must proceed to consider what the poet ...
3b9047	In complete sentences, summarize the structure of the ancient Egyptian system of government. How wer...	Egyptian Social Structure	Egyptian society was structured like a pyramid. At the top were the gods, such as Ra, Osiris, and Is...
814d6b	Summarize how the Third Wave developed over such a short period of time and why the experiment was e...	The Third Wave	Background The Third Wave experiment took place at Cubberley High School in Palo Alto, California ...
ebad26	Summarize the various ways the factory would use or cover up spoiled meat. Cite evidence in your ans...	Excerpt from The Jungle	With one member trimming beef in a cannery, and another working in a sausage factory, the family had...

https://www.kaggle.com/competitions/commonlit-evaluate-student-summaries

Exploratory Data Analysis

- Before diving into the competition, it is always recommended to take a look at the data.

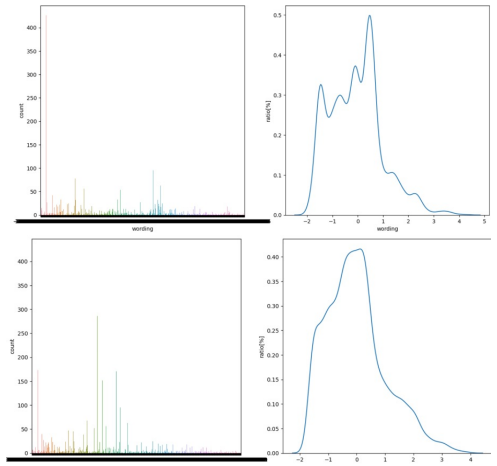
	prompt_id	prompt_question	prompt_title	prompt_text	student_id	text	content	wording
0	39c16e	Summarize at least 3 elements of an ideal trag...	On Tragedy	Chapter 13 \r\nAs the sequel to what has alrea...	00791789cc1f	1 element of an ideal tragedy is that it shoul...	-0.210614	-0.471415
1	39c16e	Summarize at least 3 elements of an ideal trag...	On Tragedy	Chapter 13 \r\nAs the sequel to what has alrea...	0086ef22de8f	The three elements of an ideal tragedy are: H...	-0.970237	-0.417058
2	39c16e	Summarize at least 3 elements of an ideal trag...	On Tragedy	Chapter 13 \r\nAs the sequel to what has alrea...	0094589c7a22	Aristotle states that an ideal tragedy should ...	-0.387791	-0.584181
3	39c16e	Summarize at least 3 elements of an ideal trag...	On Tragedy	Chapter 13 \r\nAs the sequel to what has alrea...	00cd5736026a	One element of an Ideal tragedy is having a co...	0.088882	-0.594710
4	39c16e	Summarize at least 3 elements of an ideal trag...	On Tragedy	Chapter 13 \r\nAs the sequel to what has alrea...	00d98b8ff756	The 3 ideal of tragedy is how complex you need...	-0.687288	-0.460886

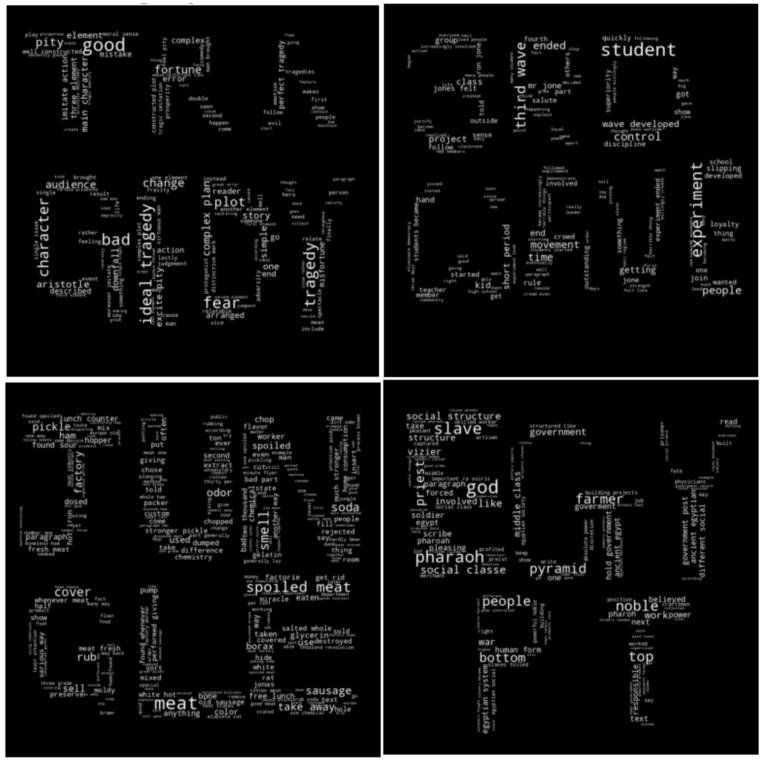


	num_words	num_unique_words	num_chars	num_stopwords	num_punctuations	num_words_upper	num_words_title	mean_word_len
count	1996.0000	1996.0000	1996.0000	1996.0000	1996.0000	1996.0000	1996.0000	1996.0000
mean	79.4509	54.5040	420.7560	42.1242	10.3472	0.0521	3.6844	4.2630
std	55.6905	30.2666	302.9265	29.4349	9.2133	0.3155	3.3914	0.3110
min	23.0000	18.0000	115.0000	8.0000	0.0000	0.0000	0.0000	3.2581
25%	41.0000	32.0000	214.0000	22.0000	4.0000	0.0000	2.0000	4.0522
50%	63.0000	47.0000	330.0000	33.0000	8.0000	0.0000	3.0000	4.2500
75%	100.0000	68.0000	527.2500	53.2500	14.0000	0.0000	5.0000	4.4458
max	509.0000	252.0000	2775.0000	257.0000	88.0000	8.0000	30.0000	5.8947

```
Train content description:
count    7165.000000
mean       -0.014853
std         1.043569
min        -1.729859
25%        -0.799545
50%        -0.093814
75%         0.499660
max         3.900326
Name: content, dtype: float64

Train wording description:
count    7165.000000
mean       -0.063072
std         1.036048
min        -1.962614
25%        -0.872720
50%        -0.081769
75%         0.503833
max         4.310693
Name: wording, dtype: float64
```





https://www.kaggle.com/code/jasonheesanglee/spellcheck-tool-comparison

Preprocessing – Text Cleansing

- Cleansing the text is a must to run the NLP model effectively.
- I first brought the `text_processor` module I composed while working on the previous competition.
  It includes
  - Regex pattern to replace all the words in all forms of brackets.
  - Converting `---n't` to `--- not` format.
  - Removing `stopwords`.
- Then, to compare the results, a controller was constructed to control whether to include each preprocess function.
- Eventually, I decided to exclude all the preprocess functions other than `stopwords` as this had the best Public Leaderboard score.
- On top of that, I decided to merge prompt_question and text columns so the model could better learn.

```
pattern = r'\([^)]*\)'  # ()
s = re.sub(pattern=pattern, repl='', string=s)
pattern = r'\[[^)]*\]'  # []
s = re.sub(pattern=pattern, repl='', string=s)
pattern = r'\<[^)]*\>'  # <>
s = re.sub(pattern=pattern, repl='', string=s)
pattern = r'\{[^)]*\}'  # {}
s = re.sub(pattern=pattern, repl='', string=s)

pattern = r'[^a-zA-Z0-9]'
s = re.sub(pattern=pattern, repl=' ', string=s)
s = s.replace("'", ' ')
s = s.replace(',', ' ')
s_split = s.split()

not_list0 = ['isnt', 'arent', 'wasnt','werent',
             'didnt', 'dont', 'doesnt',
             'hasnt', 'havent', 'hadnt',
             'neednt', 'darent',
             'oughtnt', 'mustnt',
             'wouldnt', 'couldnt', 'wouldnt', 'shouldnt']
not_list1 = ['cant', 'wont']

s_list = []
for word in s_split:
    if len(word) != 1:
        if word in not_list0:
            s_list.append(word[:-2])
            s_list.append('not')
        elif word == not_list1[0]:
            s_list.append('cannot')
        elif word == not_list1[1]:
            s_list.append('will')
            s_list.append('not')
        else:
            s_list.append(word)

s = " ".join(s_list)
s_split = s.split()

s_list = []
for word in s_split:
    if word in stop_words:
        continue
    else:
        s_list.append(word)

s = " ".join(s_list)
```

```
yes = True
no = False

preprocess = no ######## FIX AS YES ########
basic_preprocess = no
nt_preprocess = no
stop_words_switch = no ######## FIX AS NO ########

def text_processor_2(s, stop_words, nt_preprocess):
    """
    문장을 담고있는 variable을 넣어주면
    알파벳을 제외한 문장의 모든 기호, 숫자를 제거합니다.

    :param s: 문장을 담고있는 variable
    :return: 새로운 DataFrame안에 담긴 text_processor가 적용된 column
    """
    if basic_preprocess:
        pattern = r'\([^)]*\)'  # ()
        s = re.sub(pattern=pattern, repl='', string=s)
        pattern = r'\[[^)]*\]'  # []
        s = re.sub(pattern=pattern, repl='', string=s)
        pattern = r'\<[^)]*\>'  # <>
        s = re.sub(pattern=pattern, repl='', string=s)
        pattern = r'\{[^)]*\}'  # {}
        s = re.sub(pattern=pattern, repl='', string=s)

        pattern = r'[^a-zA-Z0-9]'
        s = re.sub(pattern=pattern, repl=' ', string=s)
        s = s.replace("'", ' ')
        s = s.replace(',', ' ')

        if nt_preprocess:
            s_split = s.split()

            not_list0 = ['isnt', 'arent', 'wasnt','werent',
                         'didnt', 'dont', 'doesnt',
                         'hasnt', 'havent', 'hadnt',
                         'neednt', 'darent',
                         'oughtnt', 'mustnt',
                         'wouldnt', 'couldnt', 'wouldnt', 'shouldnt']
            not_list1 = ['cant', 'wont']

            s_list = []
            for word in s_split:
                if len(word) != 1:
                    if word in not_list0:
                        s_list.append(word[:-2])
                        s_list.append('not')
                    elif word == not_list1[0]:
                        s_list.append('cannot')
                    elif word == not_list1[1]:
                        s_list.append('will')
                        s_list.append('not')
                    else:
                        s_list.append(word)

            s = " ".join(s_list)
```

```
if nt_preprocess and not basic_preprocess:
    s_split = s.split()

    not_list0 = ["isn't", "aren't", "wasn't","weren't",
                 "didn't", "don't", "doesn't",
                 "hasn't", "haven't", "hadn't",
                 "needn't", "daren't",
                 "oughtn't", "mustn't",
                 "wouldn't", "couldn't", "wouldn't", "shouldn't"]
    not_list1 = ["can't", "won't"]

    s_list = []
    for word in s_split:
        if len(word) != 1:
            if word in not_list0:
                s_list.append(word[:-3])
                s_list.append('not')
            elif word == not_list1[0]:
                s_list.append('cannot')
            elif word == not_list1[1]:
                s_list.append('will')
                s_list.append('not')
            else:
                s_list.append(word)

    s = " ".join(s_list)

if stop_words_switch:
    s_split = s.split()

    s_list = []
    for word in s_split:
        if word in stop_words:
            continue
        else:
            s_list.append(word)

    s = " ".join(s_list)

return s
```

Public LB	CFG	Preprocess	stop_words_switch	nt_preprocess	manage_missing_words
0.498	LR = 1.5e-5 wd = 0.02 hdp = 0.005 apdp = 0.005 n_epochs = 3 n_splits = 4 batch_size = 12	yes	yes	yes	-
0.822	LR = 1.1e-9 wd = 0.02 hdp = 0.005 apdp = 0.005 n_epochs = 10 n_splits = 4 batch_size = 4	-	-	-	-
1.290	LR = 1.5e-5 wd = 0.02 hdp = 0.005 apdp = 0.005 n_epochs = 3 n_splits = 4 batch_size = 12	yes	yes	yes	yes
0.853	LR = 1.9e-1 wd = 0.2 hdp = 0.005 apdp = 0.005 n_epochs = 4 n_splits = 4 batch_size = 12	yes	no	yes	yes
0.852	LR = 1.9e-1 wd = 0.2 hdp = 0.005 apdp = 0.005 n_epochs = 4 n_splits = 4 batch_size = 12	yes	yes with additional split words	yes	yes

https://www.kaggle.com/competitions/commonlit-evaluate-student-summaries
https://www.kaggle.com/code/jasonheesanglee/en-mobilebert-implementation

Preprocessing - SpellCheck

- After going through the frequently used words on the word cloud, I realized that there were many occasions where the students had misspelled words.

  For example, many students have written “Pharaoh” as “Pharoh” or “Pharoah”.

- Considering that the task was to evaluate the summary, I believed that the misspelled words would affect the score.

  Therefore, I decided not to replace the misspelled words with the correct word.

- However, after finding out that the competition organizer mentioned that spelling is not the evaluation criteria, I have compared different spell-checking tools with Grammarly to find out which tool performs the best.

- Then SymSpellPy caught my attention as it was the only tool that correctly replaced “thenwent” with “then went”.

- The only struggle using SymSpellPy was that it had the slowest processing time among all the tools.
  Therefore, I have decided to collaborate two tools : PySpellChecker for the misspelled word detecting tool and SymSpellPy as a correction tool.

- This as well was also included in the controller and changed the configuration per different training experiments.


```
method = ['py_and_sym', 'pyspell_only', 'symspell_only']

freq_dict_list = ["/kaggle/input/symspell-677/symspell_freq_dict.txt",
            "/kaggle/input/symspell-677/frequency_dictionary_en_82_765.txt",
            "/kaggle/input/symspell-677/frequency_bigramdictionary_en_243_342.txt"]

yes = True
no = False

preprocess = no ####### FIX AS YES #######
basic_preprocess = no
nt_preprocess = no
stop_words_switch = no ####### FIX AS NO #######

manage_misspelled_words = no
misspelled_word_method = method[0]
if (manage_misspelled_words==yes) & (misspelled_word_method == 'py_and_sym'):
    pyspell_detector = yes
    symspell_corrector = yes
    misspell_counter = 0

if (manage_misspelled_words==yes) & (misspelled_word_method == 'pyspell_only'):
    pyspell_detector = yes
    symspell_corrector = no
    misspell_counter = 1

if (manage_misspelled_words==yes) & (misspelled_word_method == 'symspell_only'):
    pyspell_detector = no
    symspell_corrector = yes
    freq_dict = freq_dict_list[0]
    misspell_counter = 2
```

Tool	Detected	Correctly Replaced	Misjudged
Grammarly	16 words	16 words	Less 1 word
TextBlob	20 words	14 words	2 words 2 names
PySpellChecker	18 words	16 words	2 words
SymSpellPy	18 words	17 words	1 name
AutoCorrect	15 words	12 words	2 words 1 name

https://www.kaggle.com/competitions/commonlit-evaluate-student-summaries
https://www.kaggle.com/code/jasonheesanglee/eda-spellcheck-tool-comparison

Preprocessing - Masking

- After reading some BERT-related research papers and watching videos about Natural Language Processing, I have learned that masking was a huge part of dealing with Natural Language data.
- When I tried to implement masking, I found that the tokens used for masking differed from model to model, although most of the models I tried were BERT-descendants.
  Therefore, I decided only to include masking when using `debertav3base`.
- I have separated the masking module into two and let each module mask Keywords and Frequently-appeared words.
- However, when I asked for feedback from the Kaggle community, they told me that masking doesn't work this way.
- Then, I tried to learn more and implemented in the other code, which I will introduce later in this document.

class transformers.AlbertTokenizer <source>

```
( vocab_file, do_lower_case = True, remove_space = True, keep_accents = False, bos_token = '[CLS]',
eos_token = '[SEP]', unk_token = '<unk>', sep_token = '[SEP]', pad_token = '<pad>', cls_token =
'[CLS]', mask_token = '[MASK]', sp_model_kwargs: typing.Union[typing.Dict[str, typing.Any], NoneType]
= None, **kwargs )
```

class transformers.DebertaTokenizer <source>

```
( vocab_file, merges_file, errors = 'replace', bos_token = '[CLS]', eos_token = '[SEP]', sep_token =
'[SEP]', cls_token = '[CLS]', unk_token = '[UNK]', pad_token = '[PAD]', mask_token = '[MASK]',
add_prefix_space = False, add_bos_token = False, **kwargs )
```

class transformers.RobertaTokenizer <source>

```
( vocab_file, merges_file, errors = 'replace', bos_token = '<s>', eos_token = '</s>', sep_token =
'</s>', cls_token = '<s>', unk_token = '<unk>', pad_token = '<pad>', mask_token = '<mask>',
add_prefix_space = False, **kwargs )
```

class transformers.MobileBertTokenizer <source>

```
( vocab_file, do_lower_case = True, do_basic_tokenize = True, never_split = None, unk_token = '[UNK]',
sep_token = '[SEP]', pad_token = '[PAD]', cls_token = '[CLS]', mask_token = '[MASK]',
tokenize_chinese_chars = True, strip_accents = None, **kwargs )
```

```python
def keyword_masking_debertav3base(df, base_column, target_column, stop_words):
    temp_1 = []

    for row in tqdm(range(df.shape[0])):
        sentence = df[target_column][row]
        keywords = [word.replace(',', '') for word in df[base_column][row].lower().split() if len(word) > 3 and word not in stop_words]
        words = [word.strip() for word in sentence.split(' ') if word != '']
        temp = []
        for word in words:
            if word.endswith(','):
                temp.append(word.replace(',', ''))
                temp.append('[SEP]')
                continue
            elif word.endswith('.'):
                temp.append(word.replace('.', ''))
                temp.append('[SEP]')
                continue
            elif word.endswith(';'):
                temp.append(word.replace(';', ''))
                temp.append('[SEP]')
                continue
            if word in keywords:
                temp.append('[MASK]')
            else:
                temp.append(word)
        sentence = ' '.join(temp)
        temp_1.append(sentence)
    temp_Series = pd.Series(temp_1)
    df[target_column] = temp_Series

    return df
```

```python
def unique_list(list_):
    unique_list = []
    for x in list_:
        if x not in unique_list:
            unique_list.append(x)
    return unique_list

def freq_word_masking_debertav3base(df, target_column, stop_words):
    temp_1 = []


    for row in tqdm(range(df.shape[0])):
        sentence = df[target_column][row]
        words = [word for word in sentence.split(' ')]

        temp_dict = {}
        for word in unique_list(words):
            temp_dict[word] = words.count(word)

        mask_list = []
        for word in temp_dict.keys():
            if len(word) > 3:
                if temp_dict.get(word) > 4:
                    if word not in stop_words:
                        mask_list.append(word)

        temp = []
        for word in words:
            if word in mask_list:
                word = '[MASK]'
                temp.append(word)
            else:
                temp.append(word)

        sentence = ' '.join(temp)
        temp_1.append(sentence)

    temp_Series = pd.Series(temp_1)
    df[target_column] = temp_Series

    return df
```

https://huggingface.co/docs/transformers/model_doc/deberta
https://www.kaggle.com/code/jasonheesanglee/en-mobilebert-implementation#Masking

Train & Infer - MobileBERT

- As the purpose of this competition is to enhance the teaching and scoring experience of the teachers, I believed making this model lightweight is the key.
  By implementing MobileBERT, the model would run on individual mobile devices; teachers at isolated locations with poor or no internet connection could also use this.
  It would not need a simultaneous connection to the internet.
- Unfortunately, I had to bring a training code from the community, as the one I composed didn't run successfully.
- When learning about Deep Learning modeling, one thing I learned was that, in many cases, it is better to have more data for the model to train on.
- Therefore, for the occasions where I turned the spellchecking module on, I made the model to be trained on the non-spellchecked data and the spellchecked data.
  Which later, I found out that this tactic made the model overfitted to the training data.
- However, by doing so, I had a chance to look closer into the codes and to understand the logic.
- Then, I tried to implement MobileBERT to the code by simply changing the name of the model in `.from_pretrained()`, but it was giving an out-of-range Public LB score.

- I realized something wrong when I saw the score and thought this is not the way how MobileBERT is implemented in the code.
- I read the research paper of MobileBERT to find the solution.
  Then, I realized that a teacher model needed to be taught before implementing MobileBERT, and the teacher model needs to teach MobileBERT.
- At first, I didn't really understand the logic behind it.
  Still, then I realized that the teacher model was the one containing the information, and MobileBERT only took the necessary information from the previous model (teacher model).
- The Public Leaderboard score after properly implementing the MobileBERT has been better than before, but it was still unsatisfying.
  Which led me to revert back to other BERT-descendant models.

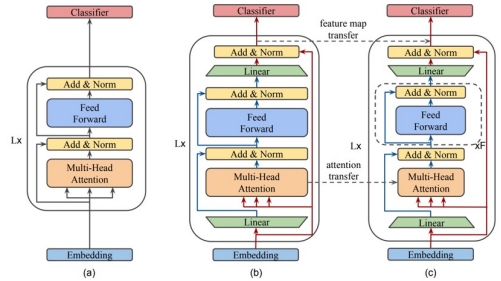


Figure 1: Illustration of three models: (a) BERT; (b) Inverted-Bottleneck BERT (IB-BERT); and (c) MobileBERT. In (b) and (c), red lines denote inter-block flows while blue lines intra-block flows. MobileBERT is trained by layer-to-layer imitating IB-BERT.

MobileBERT: a Compact Task-Agnostic BERT for Resource-Limited Devices (Sun et al., 2020)

https://arxiv.org/pdf/2004.02984.pdf

https://www.kaggle.com/code/jasonheesanglee/en-mobilebert-implementation#Masking

Train & Infer – All The Other Models

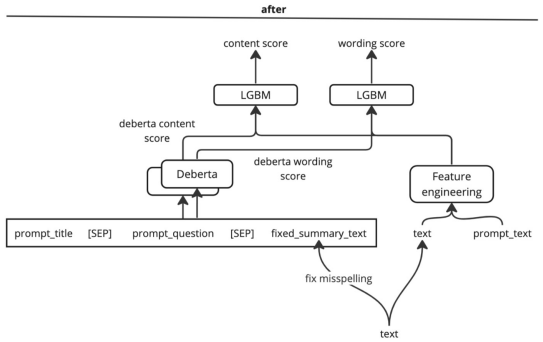
- Once the MobileBERT moved out of my scope, I wanted to try all the other available models in this Dataset. Which are:
  - albert-large-v2
  - bert-base-uncased
  - bert-large-uncased
  - distilroberta-base
  - distilbert-base-uncased
  - electra-base-discriminator
  - bart-base
  - bart-large
  - roberta-base
  - roberta-large
  - t5-base
  - t5-large
  - xlnet-base-cased
  - xlnet-large-cased
- I have tried all the non-large models above, but BART & T5.
  When I tried to implement BART and T5 as I have done for other models, a large number of error messages occurred.
  Later, I learned from the T5 research paper that unlike other encoder-only models listed above, T5 is an encoder-decoder model, and so is BART.
  Therefore, I had to construct from scratch.
- However, when I realized that I had to build a new set of code, it was already 3 days to the deadline, and I could only give up using BART and T5 for this time.
- For other models than BART and T5, I have built a list to store the model location and retrieved the model's name & location within the training & infer code structure I used for DeBERTa-v3-Base.
- I understand that it wouldn't be the best way to try out multiple models in a single structure, but as aforementioned, I did not have enough time to build again from ground zero.
- After several experiments on Google Colab, I have figured out that DistilRoBERTa-base scored the best amongst the models.
- Therefore, I have selected the DistilRoBERTa-version of the notebooks as one of my final submission.

```
files = ['debertav3base', # files[0]
         'albert-large-v2', # files[1]
         'bert-base-uncased', # files[2]
         'bert-large-uncased', # files[3]
         'distilroberta-base', # files[4]
         'distilbert-base-uncased', # files[5]
         'google-electra-base-discriminator', # files[6]
         'facebook-bart-base', # files[7] # Not working
         'facebook-bart-large', # files[8]
         'funnel-transformer-small', # files[9]
         'funnel-transformer-large', # files[10]
         'roberta-base', # files[11]
         'roberta-large', # files[12]
         't5-base', # files[13] # don't use
         't5-large', # files[14] # don't use
         'xlnet-base-cased', # files[15]
         'xlnet-large-cased' # files[16]
         ]
```

https://arxiv.org/pdf/1910.10683.pdf
https://www.kaggle.com/code/jasonheesanglee/lgbm-autocorrect-allmodels

Train & Infer – LightGBM

- After getting the result with NLP Deep Learning models, I used LightGBM to improve the score by feature engineering, as introduced in this notebook.
- The original composer of the notebook has found the optimal hyperparameters through their experiment.
  Still, as I have combined some columns on the way, I needed to find a better hyperparameter that would suit my version of the work.
- To do so, I added Optuna on LightGBM and ran multiple experiments on Google Colab.
  - This made me think of sending the result to my email, then auto shut-down the session (to save some quota). Details on this are explained in this notebook.
- However, it seemed like it didn't work since the RMSE got higher (worse) than before running LightGBM.
- Then, I decided not to fix onto one set of hyperparameter values but let the Optuna do its work for the hidden test data.




```python
for target in targets:
    models = []

    for fold in range(CFG.n_splits):

        X_train_cv = train[train["fold"] != fold].drop(columns=drop_columns)
        y_train_cv = train[train["fold"] != fold][target]

        X_eval_cv = train[train["fold"] == fold].drop(columns=drop_columns)
        y_eval_cv = train[train["fold"] == fold][target]

        dtrain = lgb.Dataset(X_train_cv, label=y_train_cv)
        dval = lgb.Dataset(X_eval_cv, label=y_eval_cv)


        def objective(trial):
            param_space = {'boosting_type': 'gbdt',
                'random_state': 42,
                'objective': 'regression',
                'metric': 'rmse',
                'learning_rate' : trial.suggest_float('learning_rate', 1e-7, 0.1, log=True),
                'max_depth' : 2 if IS_DEBUG else trial.suggest_int('max_depth', 3, 10),
                'num_leaves': 2 if IS_DEBUG else trial.suggest_int('num_leaves', 2, 1024),
                'lambda_l1' : trial.suggest_float('lambda_l1', 1e-7, 0.1, log=True),
                'lambda_l2' : trial.suggest_float('lambda_l2', 1e-7, 0.1, log=True)
            }
            params = param_space.copy()
            for param, value in trial.params.items():
                params[param] = value

            evaluation_results = {}
            model = lgb.train(params,
                              num_boost_round= 2 if IS_DEBUG else 10000,
                                #categorical_feature = categorical_features,
                              valid_names=['train', 'valid'],
                              train_set=dtrain,
                              valid_sets=dval,
                              callbacks=[
                                  lgb.early_stopping(stopping_rounds=30, verbose=True),
                                   lgb.log_evaluation(100),
                                  lgb.callback.record_evaluation(evaluation_results)
                                ],
                              )
            y_pred = model.predict(X_eval_cv)
            rmse = np.sqrt(mean_squared_error(y_eval_cv, y_pred))
            return rmse

        study = optuna.create_study(direction='minimize')
        study.optimize(objective, n_trials= 2 if IS_DEBUG else 500)

        best_params = study.best_trial.params
        evaluation_results = {}
        best_model = lgb.train(best_params,
                               num_boost_round = 2 if IS_DEBUG else 1000,
                               train_set=dtrain,
                               valid_sets=dval,
                               callbacks=[
                               lgb.early_stopping(stopping_rounds=30, verbose=True),
                                lgb.log_evaluation(100),
                               lgb.callback.record_evaluation(evaluation_results)
                               ])


        models.append(best_model)

    model_dict[target] = models
```

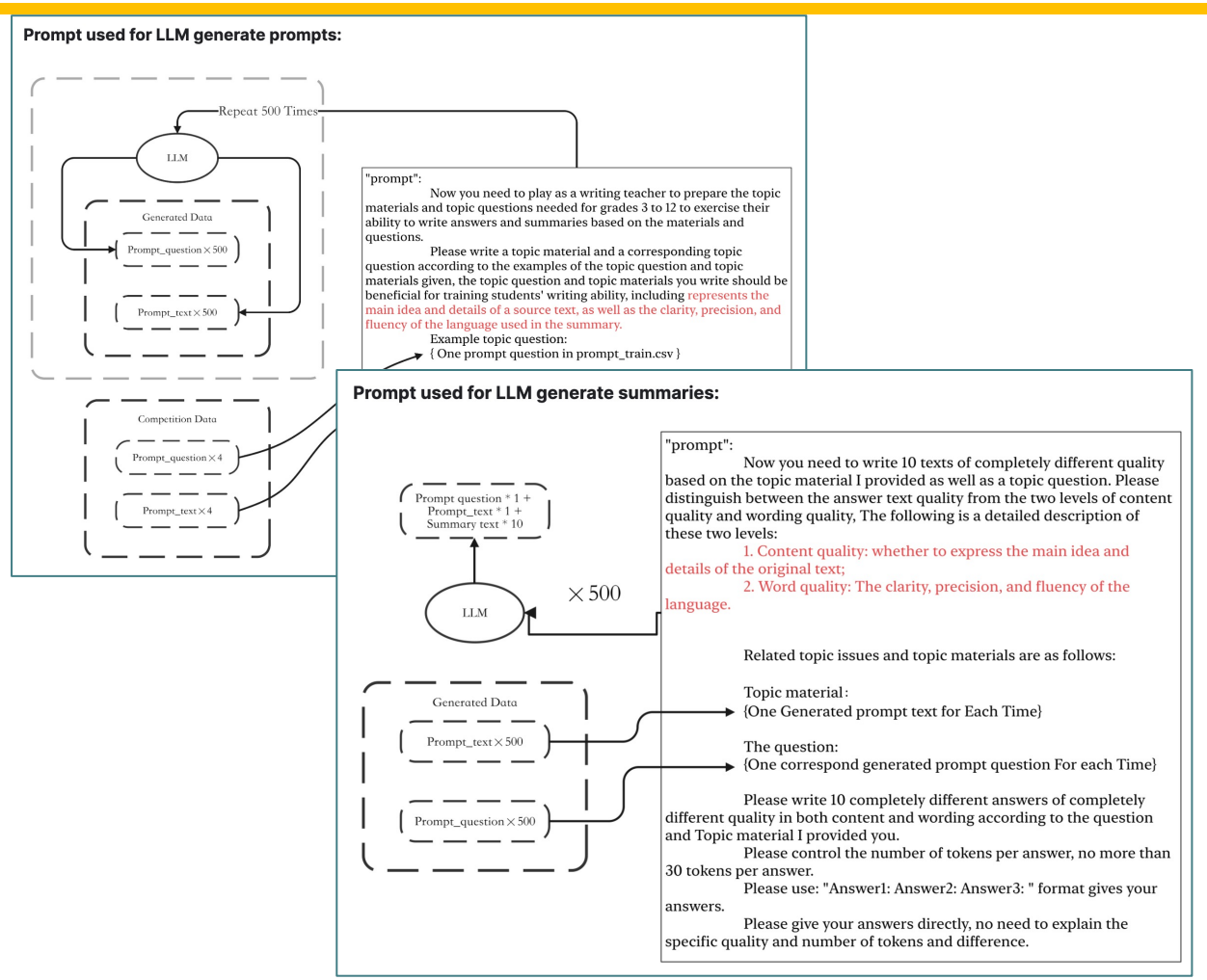
https://www.kaggle.com/code/jasonheesanglee/lgbm-autocorrect-allmodels
https://www.kaggle.com/code/tsunotsuno/updated-debertav3-lgbm-with-spell-autocorrect/notebook

1st Place Solution

- Surprisingly or not, the first-place holder turned out to be using the same baseline that we were using.
- The differences between our solution and the first-place solution are:
  1. They have barely modified the logic of the model.
  2. They not only have used the provided four prompts and generated even more by effectively using Large Language Model.
  3. They have excluded using LightGBM and solved the problem solely with DeBERTa-v3-Large.
- Per their [discussion](), they have spent more time generating new prompts and summaries with LLM than tuning the DeBERTa model.
- I believe this was quite a smart move, as the competition overview – context has already mentioned utilizing LLM.
  - Even though I don't think this is not the way the organizer expected the LLM to be used, they were the only team that actually used LLM for this competition,

Context

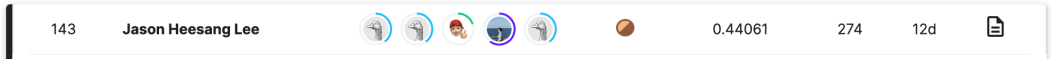
Summary writing is an important skill for learners of all ages. Summarization enhances reading comprehension, particularly among second language learners and students with learning disabilities. Summary writing also promotes critical thinking, and it's one of the most effective ways to improve writing abilities. However, students rarely have enough opportunities to practice this skill, as evaluating and providing feedback on summaries can be a time-intensive process for teachers. Innovative technology like large language models (LLMs) could help change this, as teachers could employ these solutions to assess summaries quickly.



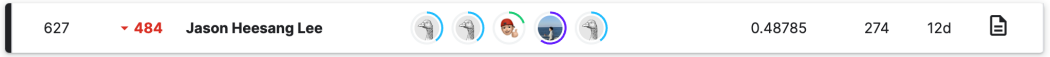
www.kaggle.com/competitions/commonlit-evaluate-student-summaries/overview/description
https://www.kaggle.com/competitions/commonlit-evaluate-student-summaries/discussion/447293

Result & Reflection

- I have teamed up with the study-group members who study NLP together.
- As we were placed at the top 143 in the Public Leaderboard, we believed that we could make it to the Bronze medal when the Private Leaderboard was published.
- Therefore, we tried our best until the very end to make minor changes to our existing solution.



- When the Private Leaderboard was published, we were pretty shocked with the result.



- We were expecting at least the top 170, which is the lower limit for the bronze medal.
- After being depressed for 30 minutes, we started analyzing what have we done wrong.
- There were quite some reasons we have detected.

- First : The proportion of the Public Data.
  The organizer stated that the test data provided to the competitors is only 13% of the whole test data.

> This leaderboard is calculated with approximately 13% of the test data.
>
> The final results will be based on the other 87%, so the final standings may be different.

  Overfitting was unavoidable as we focused on getting a better score on the Public Leaderboard.
- Second : Using SymSpellPy for spellchecking.
  I wouldn't say this is totally wrong, but most competitors used PySpellCheck and AutoCorrect.
  It was mainly AutoCorrect.
- Third : Focused too much on MobileBERT.
  This is more likely to self-reflection.
  I have spent about 2 weeks implementing MobileBERT.
  Obviously, it has been a great practice for me, but that was it.
  It didn't help in scoring.
- Fourth : Need more and better modeling techniques.
  It would have been better if we could also try T5 and BART, as no one else was trying that measure.
  Only if we had a better modeling technique, we could have also tried these two models.

https://www.kaggle.com/competitions/commonlit-evaluate-student-summaries/leaderboard