Professional Portfolio

Jason Heesang Lee

List of Contents

- Timeline

- Previous Jobs & Career Transition

- Competitions & Notebooks

- Competition Review - CommonLit – Evaluate Student Summaries

Timeline

2015.08 ~ 2021.02

2021.04 ~ 2021.09

2021.09 ~ 2023.02

2023.04 ~ 2023.12

Bachelor of Business Administration
Hospitality Management
With Luxury Specialization

People & Culture
Coordinator

Associate

Sr. Associate

Team Lead

My First
Hello World!

My First
Kaggle Submit

Achieved
Notebook Expert

Achieved
Notebook Master

Apr

Sep

Jun

Aug

Mar

Apr

Aug

Oct

2020   2021   2022   2023

Previous Jobs & Career Transition

FOUR SEASONS
HOTEL
SEOUL

Atheneum

After graduating from Glion Institute of Higher Education, a Hotel School located in Switzerland, I joined at Four Seasons Hotel Seoul as a People & Culture (P&C) Coordinator.

I was mostly responsible for the Human Resources tasks :
General Administration, Enrollment, Termination, Interviews, organizing Employee Events and Job Fair.

I joined Atheneum Partners after being scouted by a director I met previously.

I was responsible for about 15 projects per month as a Project Manager.
Exceeded target every month by around 250%, best achievement being 425%.
Thankfully, I was able to be promoted as a Team Lead, which was the fastest promotion in all global Atheneum offices.

Main Industries Covered : Artificial Intelligence, Semiconductor and Digital Transformation.

fast campus ✕ KOSME 중소벤처기업진흥공단

이어드림스쿨
스타트업-청년인재 YEAR-DREAM SCHOOL

After being being responsible for more than ten Artificial-Intelligence-related projects at Atheneum Partners,
I got more interested in the developing AI itself than developing business strategies using AI.

Coincidently, I happened to bump into a news article that Korea SMEs and Startups Agency is collaborating with Fast Campus,
one of the top computer education corporate in Korea, to launch a government-funded AI education course : Year-Dream School.

At Year-Dream School, I learned the basics required for Machine Learning and Deep Learning.
Also, I was introduced to Kaggle, which I later found my enthusiasm in.

Competitions & Notebooks

AIFactory

- **Predicting Particular Matter Pollution Degree**
  - Date : 2023-04-24 ~ 2023-05-11
  - Type of Data : Tabular / Time-Series / Regression
  - Rank : Unranked
  - Used Models : None (EDA only)
  - Focus : Getting Familiar with Pandas

DACON

- **Sound Emotion Recognition**
  - Date : 2023-05-07 ~ 2023-06-05
  - Type : Acoustic / Emotion Recognition / Classification
  - Rank : Public : 43% / Private 40%
  - Used Models : Librosa / RandomForest / DecisionTree / XGBoost / LightGBM
  - Focus : Getting familiar with Acoustic data

- **Judicial Precedent Prediction**
  - Date : 2023-06-05 ~ 2023-07-03
  - Type : NLP / Classification
  - Rank : Public : 15% / Private 18%
  - Used Models : Sentence-BERT / Legal-BERT
  - Focus : Getting familiar with Text data

kaggle

- **ICR – Identifying Age-Related Conditions (Year-Dream First Group Project)**
  - Date : 2023-05-12 ~ 2023-08-11
  - Type : Tabular / Classification
  - Rank : Public : 7% / Private 48%
  - Used Models : Rule-Based / TabPFN / RandomForest / DecisionTree / XGBoost / LightGBM
  - Focus : Finding relations between each column and the meta data

- **CommonLit – Evaluate Student Summaries (Led NLP Group Study)**
  - Date : 2023-07-13 ~ 2023-10-12
  - Type of Data : NLP / LLM / Text Summary Evaluation
  - Rank : Public : 7% / Private 30%
  - Used Models : MobileBERT, DeBERTa, Numerous BERT family models
  - Focus : Transformer-based Deep Learning model compression

- **Kaggle – LLM Science Exam**
  - Date : 2023-07-12 ~ 2023-10-11
  - Type of Data : NLP / LLM / Question Answering
  - Rank : Public : 15% / Private 15%
  - Used Models : T5, DeBERTa, LLaMA2, Platypus2, Alpaca
  - Focus : Getting familiar dealing with Large Language Models.

- **Linking Writing Process to Writing Quality**
  - Date : 2023-10-03 ~ 2024-01-10
  - Type of Data : Machine Learning / NLP / Classification
  - Rank : On-Going
  - Used Models : BERT family models
  - Focus : Reconstructing Essay by Keystroke.

Notebooks

- **Competition Related**
  - **ICR-Identifying Age-Related Conditions**
    🏅 EDA on Greeks
  - **Linking Writing Process to Writing Quality**
    🏅 Trying to Reveal Hidden Words
  - 🥉 EDA & Preprocessing
  - **RSNA 2023 Abdominal Trauma Detection**
    🏅 scale_h implementation
  - **CommonLit – Evaluating Student Summaries**
    🥈 EDA & SpellCheck Tool Comparison
    🥈 MobileBERT Implementation
    🥉 DeBERTa-V3-Large - Inference
  - **Kaggle – LLM Science Exam**
    🥈 DeBERTa-V3-Large Modification
  - **Google - Fast or Slow? Predict AI Model Runtime**
    ⚪ Checking Keys and Values – All Files
  - Stanford Ribonanza RNA Folding
    🥉 RNA Science Checking Files

- **Individual Research**
  🥈 Ideogram-based vs. Phogram-based Language
  🥉 BERT-base-uncased Implementation

- **Kaggle & Colab Utility**
  🥈 Email Notification & Colab Session AutoShutdown
  ⚪ Wheel Downloader

COMMONLIT

Evaluate Student Summaries

Competition Review

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.

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

- We were asked to find content & wording scores.

- The competition host has explained the differences between these two scores in rubric items.

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.

Exploratory Data Analysis

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


```
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
```
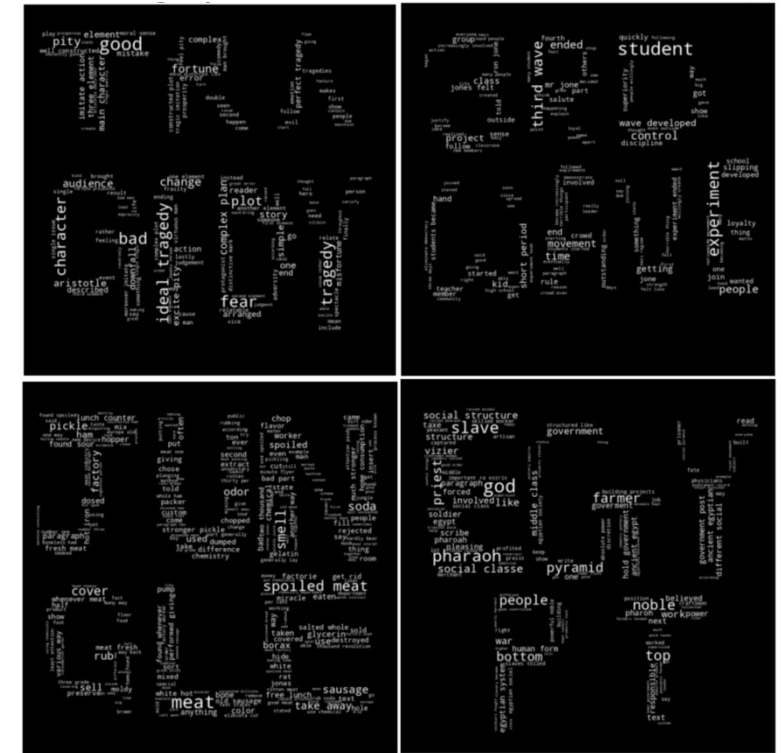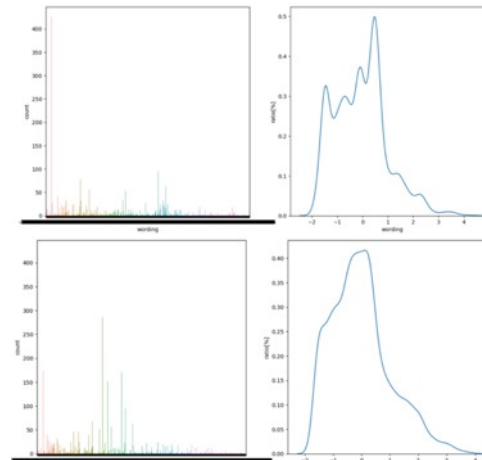
Preprocessing – Text Cleansing

- 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.

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

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
```

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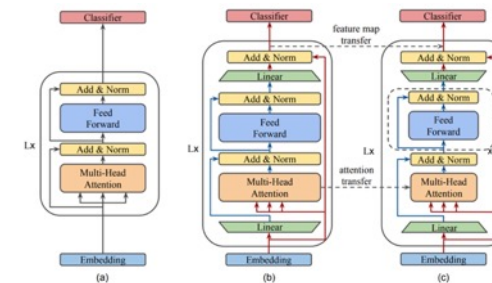
Figure 1: Illustration of three models: (a) BERT; (b) Inverted-Bottleneck BERT (IB-BERT); and (c) MobileBERT. In (b) and (c), red lines denote inter-block flows while blue lines intra-block flows. MobileBERT is trained by layer-to-layer imitating IB-BERT.

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

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]
         ]
```

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.
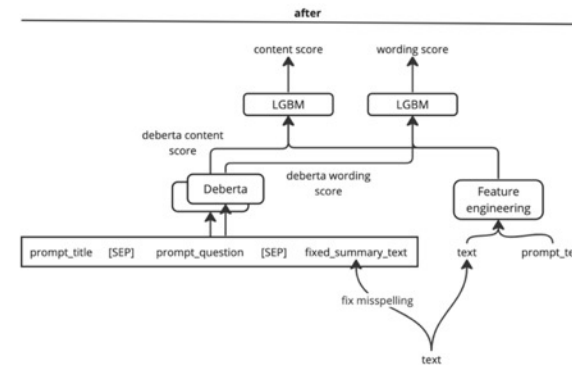

```python
for target in targets:
    models = []

    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
```

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,

Prompt used for LLM generate prompts:

"prompt":
Now you need to play as a writing teacher to prepare the topic materials and topic questions needed for grades 3 to 12 to exercise their ability to write answers and summaries based on the materials and questions.
Please write a topic material and a corresponding topic question according to the examples of the topic question and topic materials given, the topic question and topic materials you write should be beneficial for training students' writing ability, including represents the main idea and details of a source text, as well as the clarity, precision, and fluency of the language used in the summary.
Example topic question:
{ One prompt question in prompt_train.csv }

Prompt used for LLM generate summaries:

"prompt":
Now you need to write 10 texts of completely different quality based on the topic material I provided as well as a topic question. Please distinguish between the answer text quality from the two levels of content quality and wording quality, The following is a detailed description of these two levels:
1. Content quality: whether to express the main idea and details of the original text;
2. Word quality: The clarity, precision, and fluency of the language.

Related topic issues and topic materials are as follows:

Topic material：
{One Generated prompt text for Each Time}

The question:
{One correspond generated prompt question For each Time}

Please write 10 completely different answers of completely different quality in both content and wording according to the question and Topic material I provided you.
Please control the number of tokens per answer, no more than 30 tokens per answer.
Please use: "Answer1: Answer2: Answer3: " format gives your answers.
Please give your answers directly, no need to explain the specific quality and number of tokens and difference.

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.
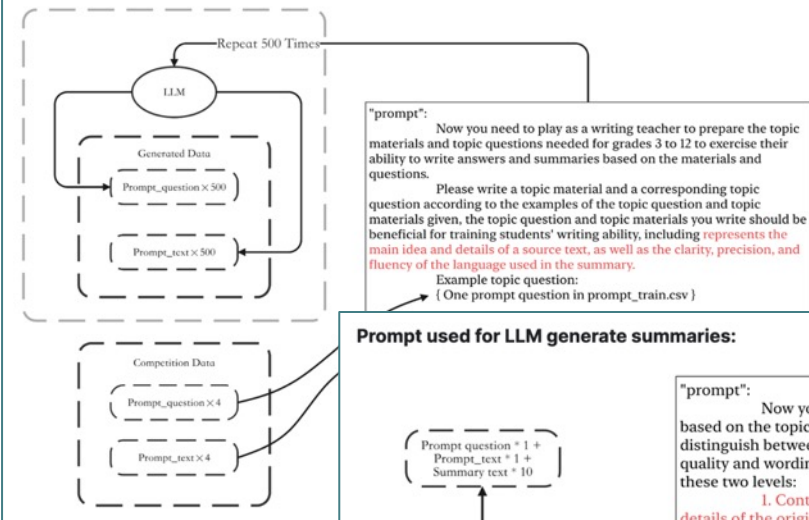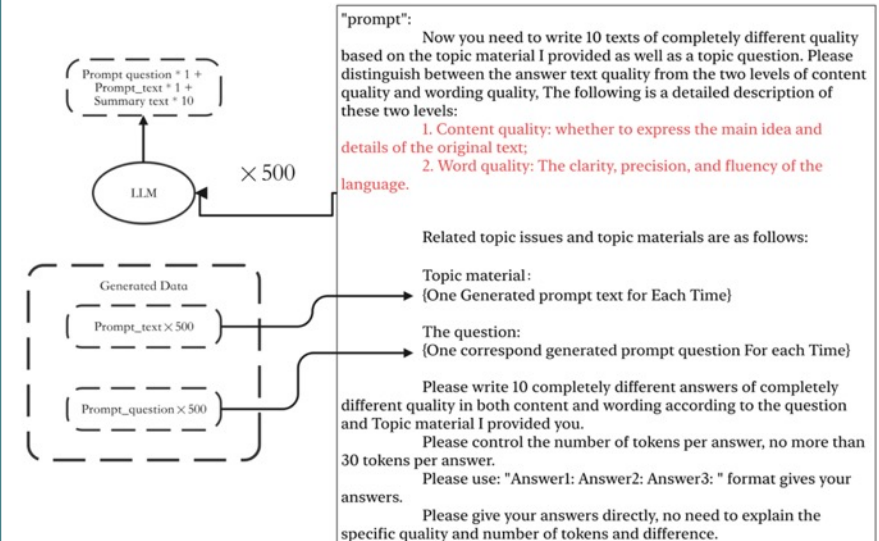
Result & Reflection

- 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.

| 143 | Jason Heesang Lee | | 0.44061 | 274 | 12d | |
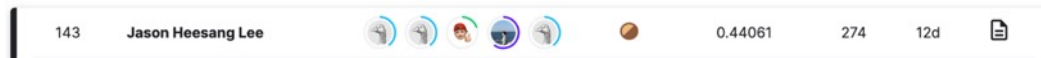
- When the Private Leaderboard was published, we were pretty shocked with the result.

| 627 | ▼ 484 | Jason Heesang Lee | | 0.48785 | 274 | 12d | |
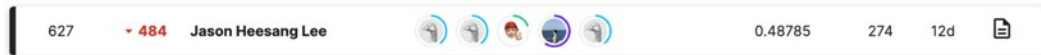
- We were expecting at least the top 170, which is the lower limit for the bronze medal.

- 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.