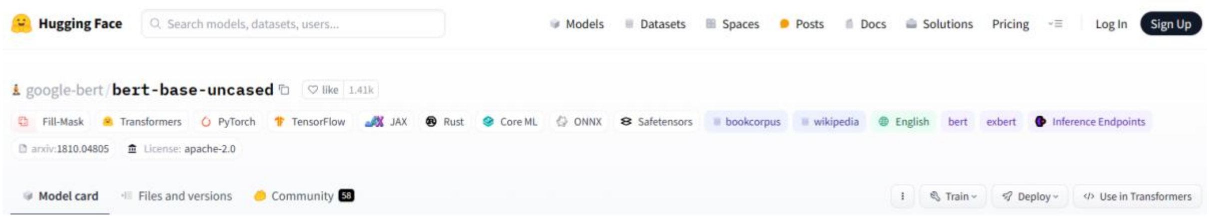
Phuvanach Phoemphol

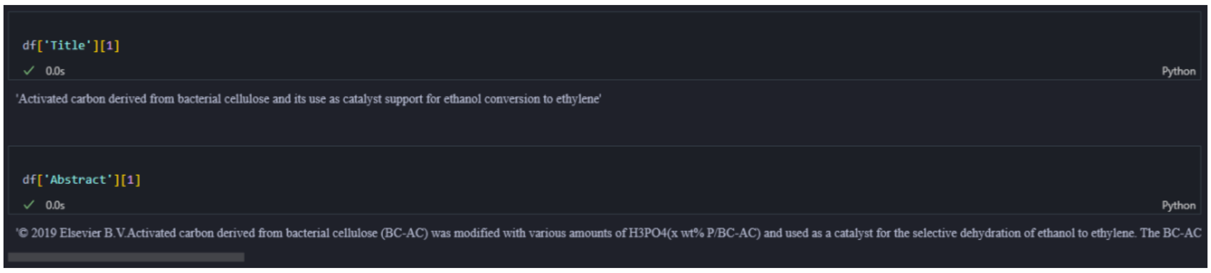
Multi-Label Classification: Kaggle

Chapter 1: Introduction

The problem is to find multi-label classification on 18 subject areas of engineering. Based on my knowledge, I don't have any experience in doing multi-label classification before. So I did some research and found a transformer-based model called BERT, developed by Google, which is known for its robustness and accuracy in performing NLP tasks, and it can also effectively in doing multi-label classification.



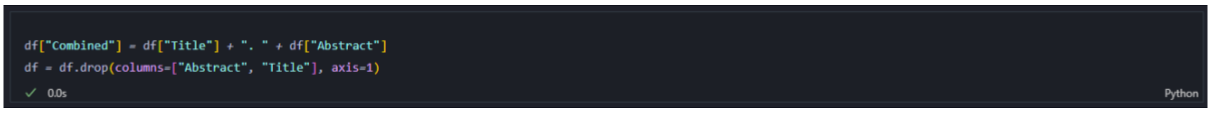
I decided to utilize bert-base-uncased from Hugging Face due to my limited resources and the dataset is not complex. I also inspected the title and abstract fields in the data, and it appears that they are case-insensitive, I've concluded that bert-base-uncased would be more suitable for me than bert-large-uncased or bert-large-cased.


```
df['Title'][1]
✓ 0.0s                                                                Python
'Activated carbon derived from bacterial cellulose and its use as catalyst support for ethanol conversion to ethylene'
```

```
df['Abstract'][1]
✓ 0.0s                                                                Python
'© 2019 Elsevier B.V.Activated carbon derived from bacterial cellulose (BC-AC) was modified with various amounts of H3PO4(x wt% P/BC-AC) and used as a catalyst for the selective dehydration of ethanol to ethylene. The BC-AC
```

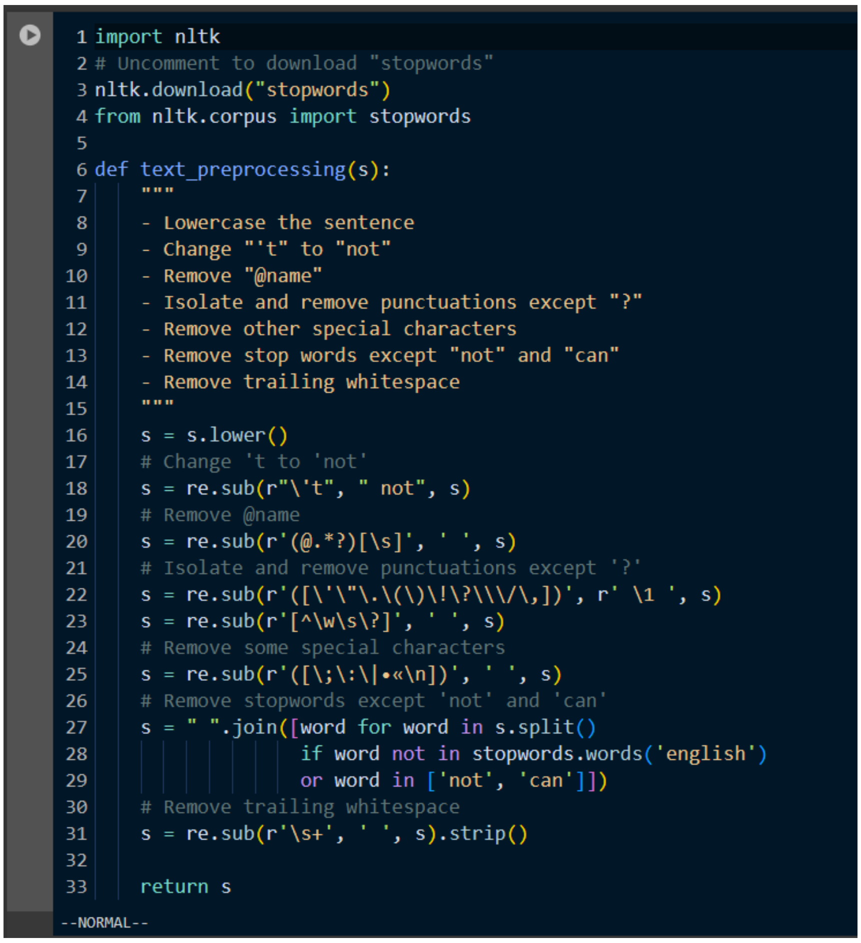
Chapter 2: Data preparation

First, I combined the Title field and Abstract field together to a new field called Combined to improve the effectiveness and reduce the dimensionality.

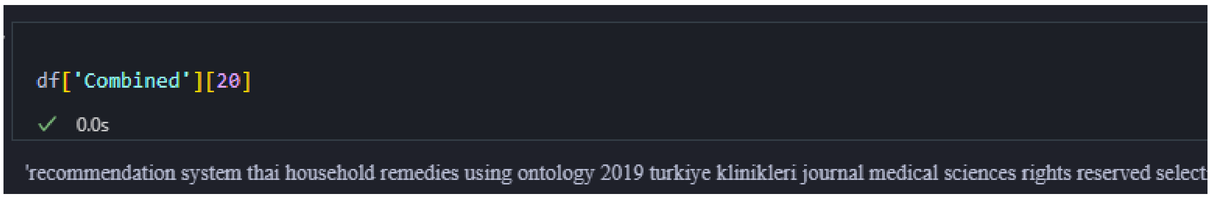


```python
df["Combined"] = df["Title"] + ". " + df["Abstract"]
df = df.drop(columns=["Abstract", "Title"], axis=1)
```

Next, I try to apply text preprocessing code from example code in GitHub classroom to my Combined filed, and the result gave me this

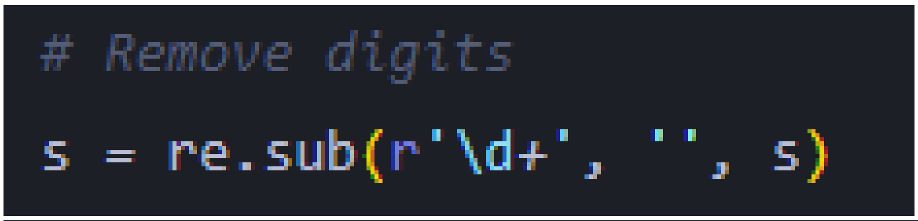


```python
import nltk
# Uncomment to download "stopwords"
nltk.download("stopwords")
from nltk.corpus import stopwords

def text_preprocessing(s):
    """
    - Lowercase the sentence
    - Change "'t" to "not"
    - Remove "@name"
    - Isolate and remove punctuations except "?"
    - Remove other special characters
    - Remove stop words except "not" and "can"
    - Remove trailing whitespace
    """
    s = s.lower()
    # Change 't to 'not'
    s = re.sub(r"\'t", " not", s)
    # Remove @name
    s = re.sub(r'(@.*?)[\s]', ' ', s)
    # Isolate and remove punctuations except '?'
    s = re.sub(r'([\'\"\.\(\)\!\?\\\/\,])', r' \1 ', s)
    s = re.sub(r'[^\w\s\?]', ' ', s)
    # Remove some special characters
    s = re.sub(r'([\;\:\|•«\n])', ' ', s)
    # Remove stopwords except 'not' and 'can'
    s = " ".join([word for word in s.split()
                  if word not in stopwords.words('english')
                  or word in ['not', 'can']])
    # Remove trailing whitespace
    s = re.sub(r'\s+', ' ', s).strip()

    return s
```

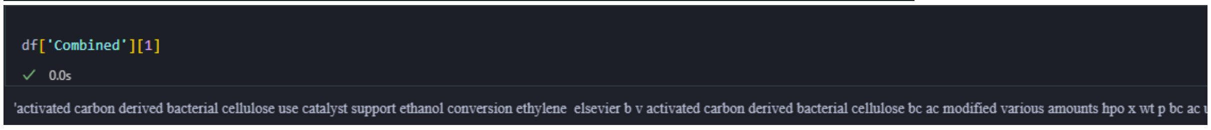


```
df['Combined'][20]
✓ 0.0s
'recommendation system thai household remedies using ontology 2019 turkiye klinikleri journal medical sciences rights reserved select
```

It works very well but, it still have year or irrelevant number that are not related to predicting class. Therefore, I decided to remove numbers from the label.

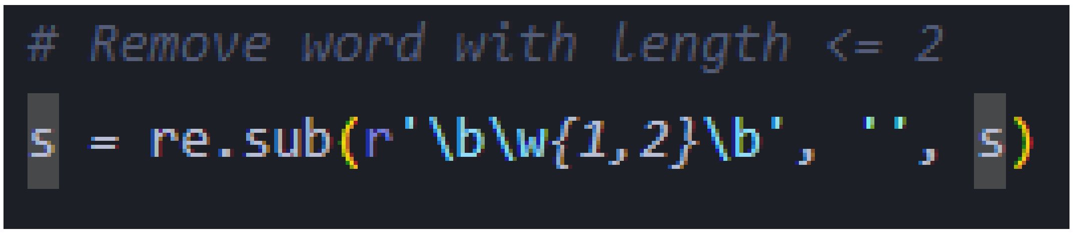


```python
# Remove digits
s = re.sub(r'\d+', '', s)
```



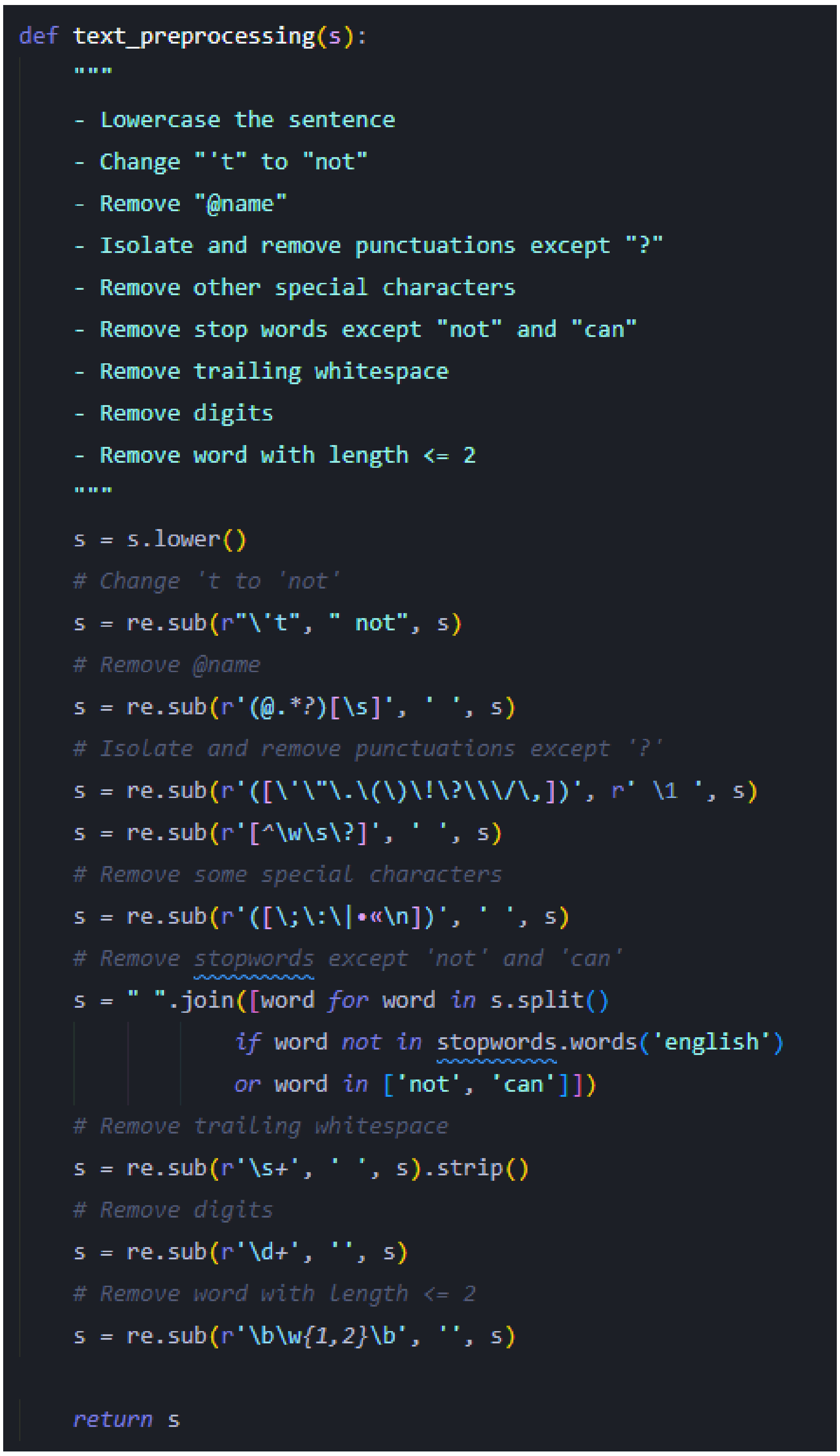
```
df['Combined'][1]
✓ 0.0s
'activated carbon derived bacterial cellulose use catalyst support ethanol conversion ethylene  elsevier b v activated carbon derived bacterial cellulose bc ac modified various amounts hpo x wt p bc ac
```

Because I remove all digits, it seems that some unknow words or units such as “b”, “v”, “bc”, “ac” appeared, which required me to remove words that length less than or equal to 2.

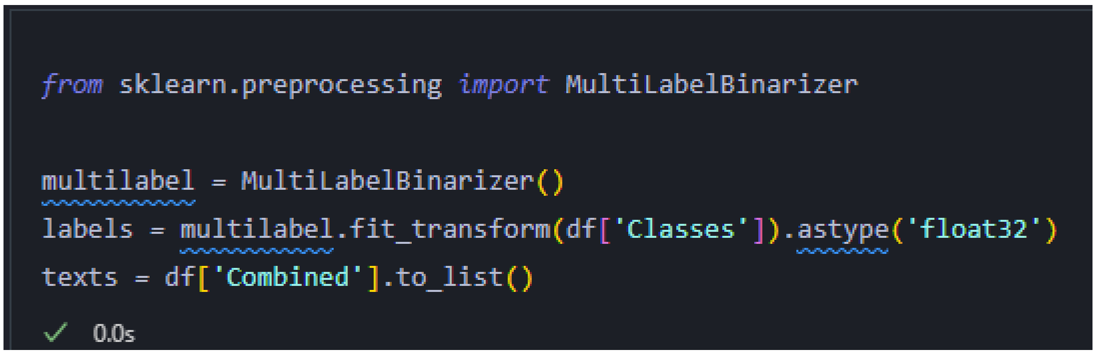


```python
# Remove word with length <= 2
s = re.sub(r'\b\w{1,2}\b', '', s)
```

Here is my final text_preprocessing function.

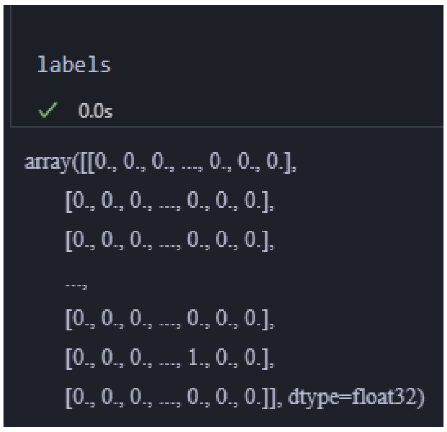


```python
def text_preprocessing(s):
    """

    - Lowercase the sentence

    - Change "'t" to "not"

    - Remove "@name"

    - Isolate and remove punctuations except "?"

    - Remove other special characters

    - Remove stop words except "not" and "can"

    - Remove trailing whitespace

    - Remove digits

    - Remove word with length <= 2
    """

    s = s.lower()
    # Change 't to 'not'
    s = re.sub(r"\'t", " not", s)
    # Remove @name
    s = re.sub(r'(@.*?)[\s]', ' ', s)
    # Isolate and remove punctuations except '?'
    s = re.sub(r'([\'\"\.\(\)\!\?\\\/\,])', r' \1 ', s)
    s = re.sub(r'[^\w\s\?]', ' ', s)
    # Remove some special characters
    s = re.sub(r'([\;\:\|•«\n])', ' ', s)
    # Remove stopwords except 'not' and 'can'
    s = " ".join([word for word in s.split()
                if word not in stopwords.words('english')
                or word in ['not', 'can']])
    # Remove trailing whitespace
    s = re.sub(r'\s+', ' ', s).strip()
    # Remove digits
    s = re.sub(r'\d+', '', s)
    # Remove word with length <= 2
    s = re.sub(r'\b\w{1,2}\b', '', s)

    return s
```



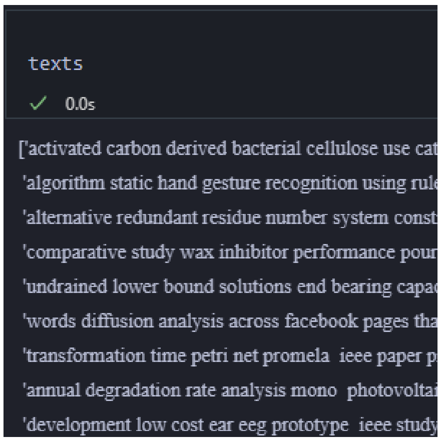
```
from sklearn.preprocessing import MultiLabelBinarizer

multilabel = MultiLabelBinarizer()
labels = multilabel.fit_transform(df['Classes']).astype('float32')
texts = df['Combined'].to_list()
```
✓ 0.0s

I used MultiLabelBinarizer to binarize multiple labels into a binary matrix in a very convincing way. The labels I got are binary column representing unique labels from Classes column for each row from in the dataframe, and the correspond to the Combined column for each row in the dataframe.



```
labels
```
✓ 0.0s

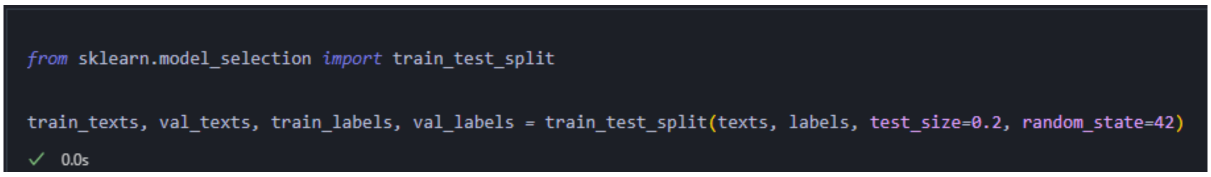
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]], dtype=float32)
```



```
texts
```
✓ 0.0s

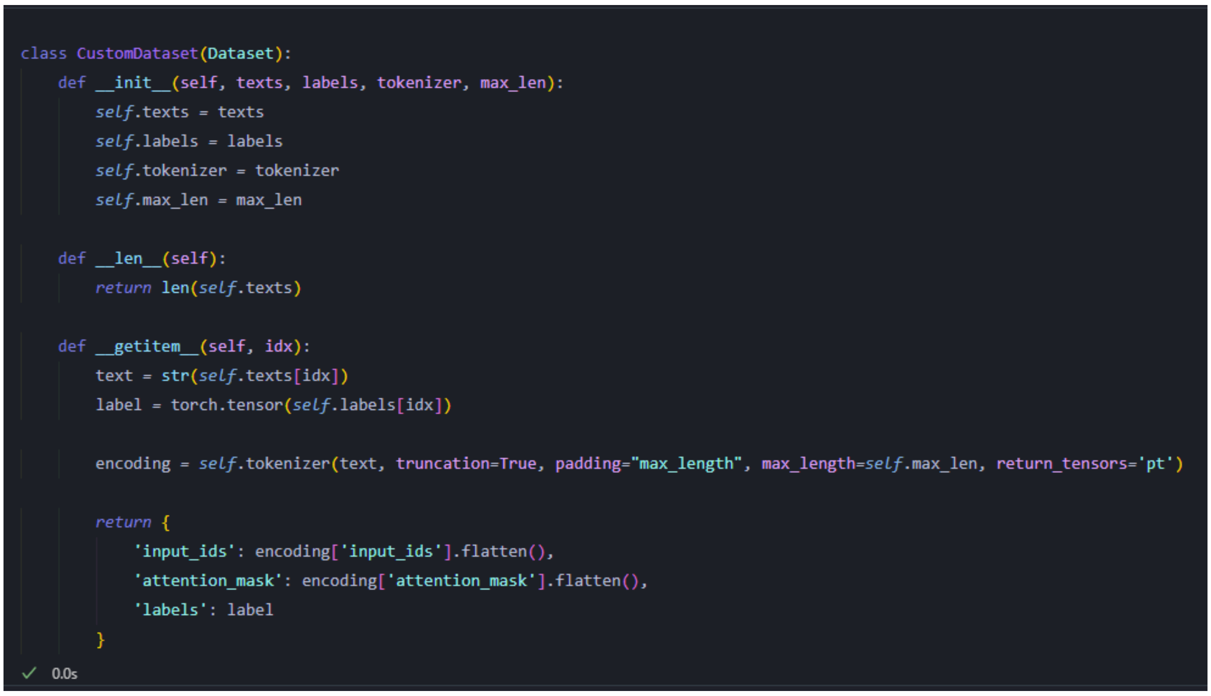
```
['activated carbon derived bacterial cellulose use cat
 'algorithm static hand gesture recognition using rule
 'alternative redundant residue number system const
 'comparative study wax inhibitor performance pour
 'undrained lower bound solutions end bearing capac
 'words diffusion analysis across facebook pages tha
 'transformation time petri net promela  ieee paper p
 'annual degradation rate analysis mono  photovoltai
 'development low cost ear eeg prototype  ieee study
```

Then, I split training set and validation set with size = 0.2 and random state = 42



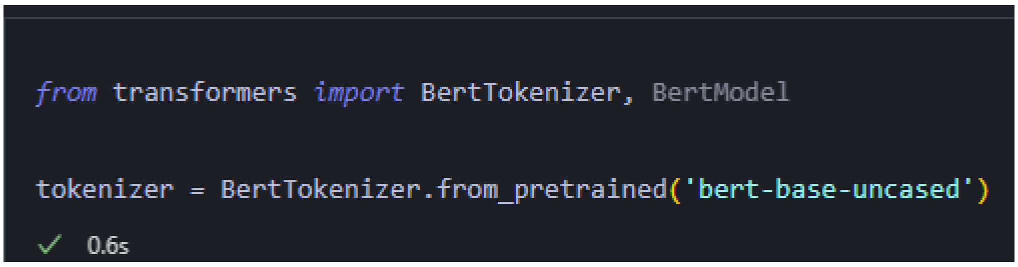
```
from sklearn.model_selection import train_test_split

train_texts, val_texts, train_labels, val_labels = train_test_split(texts, labels, test_size=0.2, random_state=42)
```
✓ 0.0s

Next, I create CustomDataset



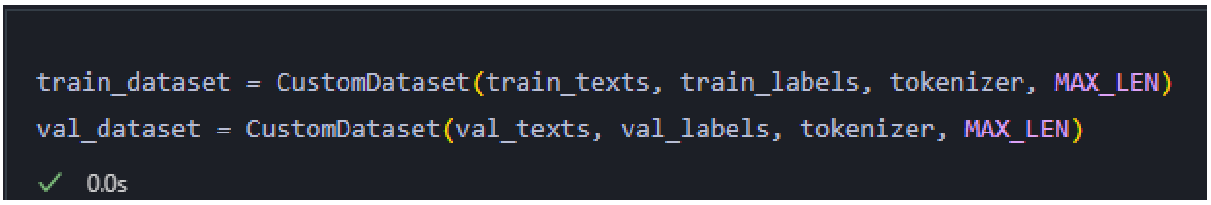
```
class CustomDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_len):
        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_len = max_len

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, idx):
        text = str(self.texts[idx])
        label = torch.tensor(self.labels[idx])

        encoding = self.tokenizer(text, truncation=True, padding="max_length", max_length=self.max_len, return_tensors='pt')

        return {
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten(),
            'labels': label
        }
✓ 0.0s
```

I also use tokenizer with the bert-base-uncased configuration, which is important and required for converting the text into a format suitable for the model’s input requirements

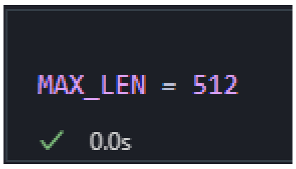


```
from transformers import BertTokenizer, BertModel

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
✓ 0.6s
```

Then, I created training and validation dataset from my CustomDataset using the train and val sets obtained earlier.



```
train_dataset = CustomDataset(train_texts, train_labels, tokenizer, MAX_LEN)
val_dataset = CustomDataset(val_texts, val_labels, tokenizer, MAX_LEN)
✓ 0.0s
```

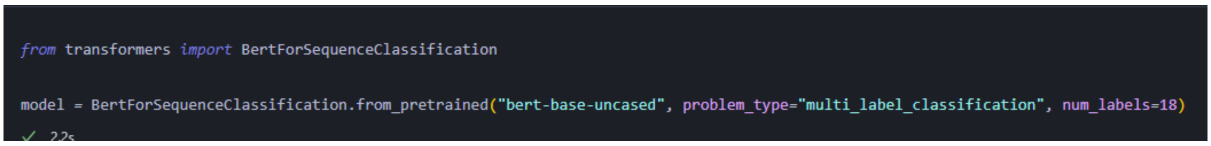


```
MAX_LEN = 512
✓ 0.0s
```

maximum number of tokens is 512 (bert-base-uncased)

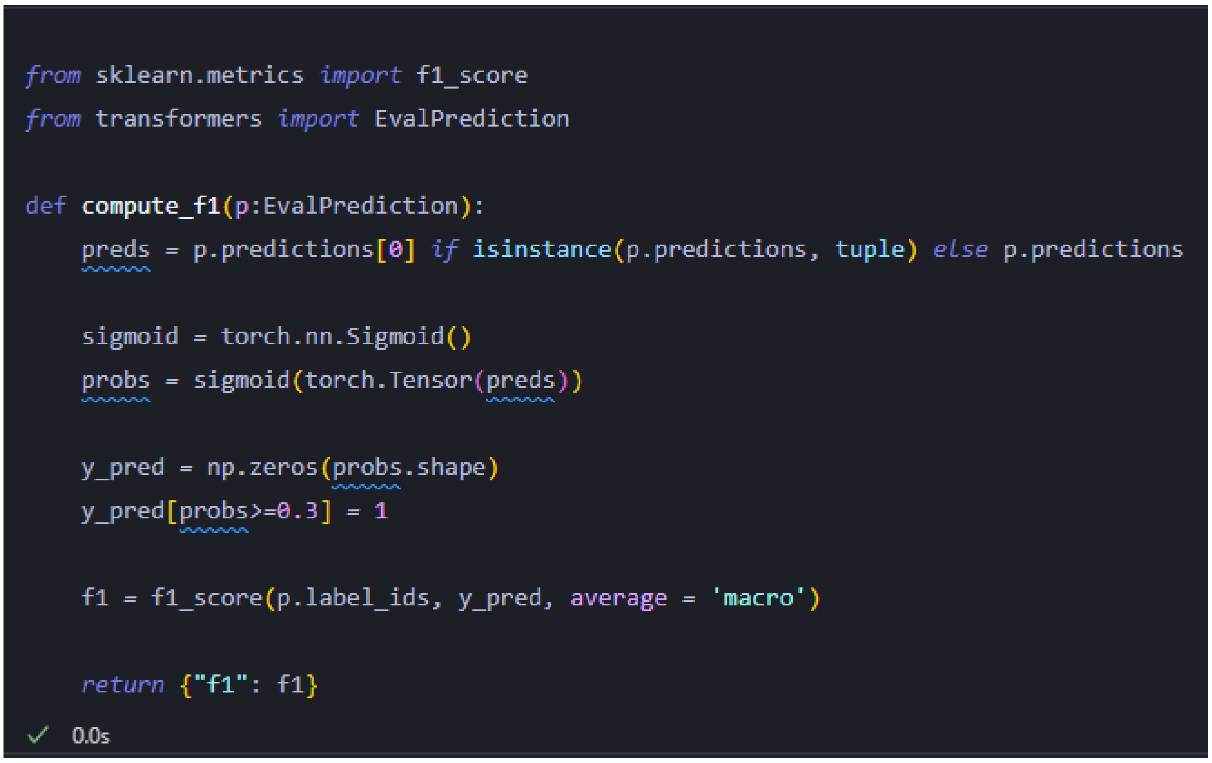
Chapter 3: Model

As I mentioned before, I used bert-base-uncased. Therefore, I declare a model using BertForSequenceClassification, specifying the problem_type as multi_label_classification and num_labels as 18, which represent the 18 subject areas of engineering.



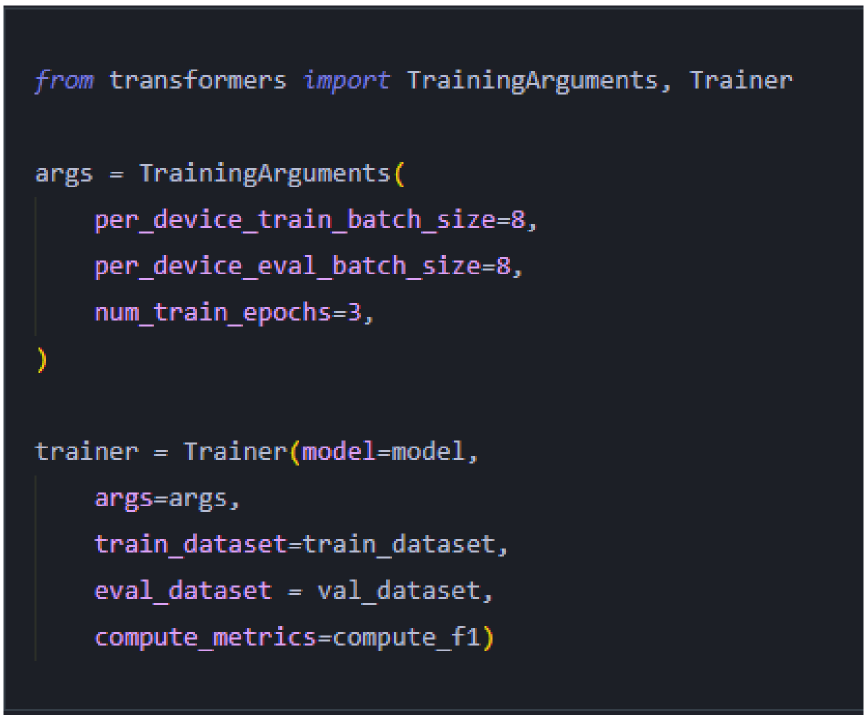
```
from transformers import BertForSequenceClassification

model = BertForSequenceClassification.from_pretrained("bert-base-uncased", problem_type="multi_label_classification", num_labels=18)
✓ 2.2s
```

Next, I created compute_f1 function that calculates F1 Score from the prediction, which is also used for public and private ranking in Kaggle. This is for model evaluation.



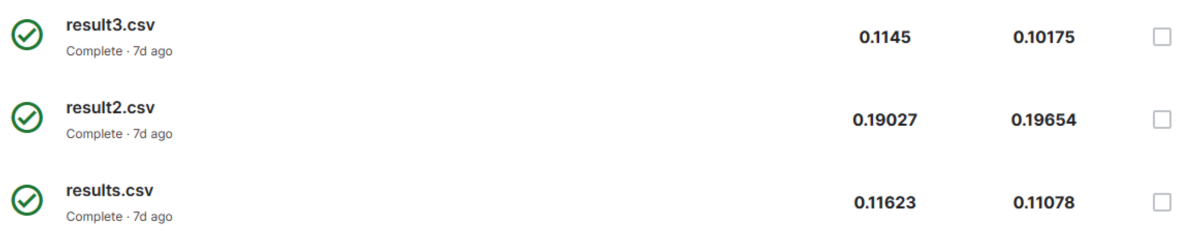
```
from sklearn.metrics import f1_score
from transformers import EvalPrediction

def compute_f1(p:EvalPrediction):
    preds = p.predictions[0] if isinstance(p.predictions, tuple) else p.predictions

    sigmoid = torch.nn.Sigmoid()
    probs = sigmoid(torch.Tensor(preds))

    y_pred = np.zeros(probs.shape)
    y_pred[probs>=0.3] = 1

    f1 = f1_score(p.label_ids, y_pred, average = 'macro')

    return {"f1": f1}
✓ 0.0s
```

After everything was ready, I configured the arguments for training and declared the trainer.



```
from transformers import TrainingArguments, Trainer

args = TrainingArguments(
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=3,
)

trainer = Trainer(model=model,
    args=args,
    train_dataset=train_dataset,
    eval_dataset = val_dataset,
    compute_metrics=compute_f1)
```

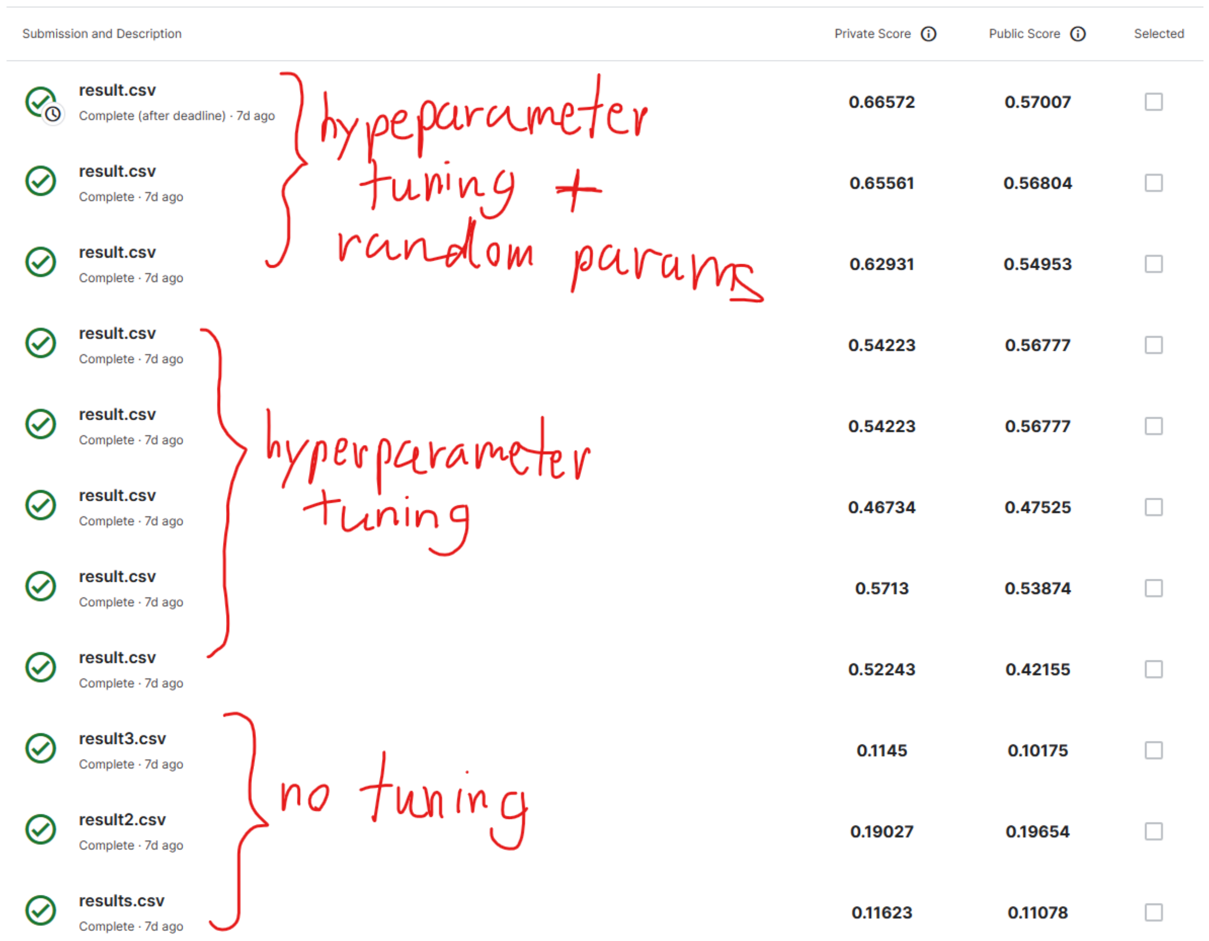
In my very first attempt, I declared the arguments as mentioned above, and here are the results.



result3.csv Complete · 7d ago	0.1145	0.10175	☐
result2.csv Complete · 7d ago	0.19027	0.19654	☐
results.csv Complete · 7d ago	0.11623	0.11078	☐

Chapter 4: Results

I can separate my results into 3 different sections as below.

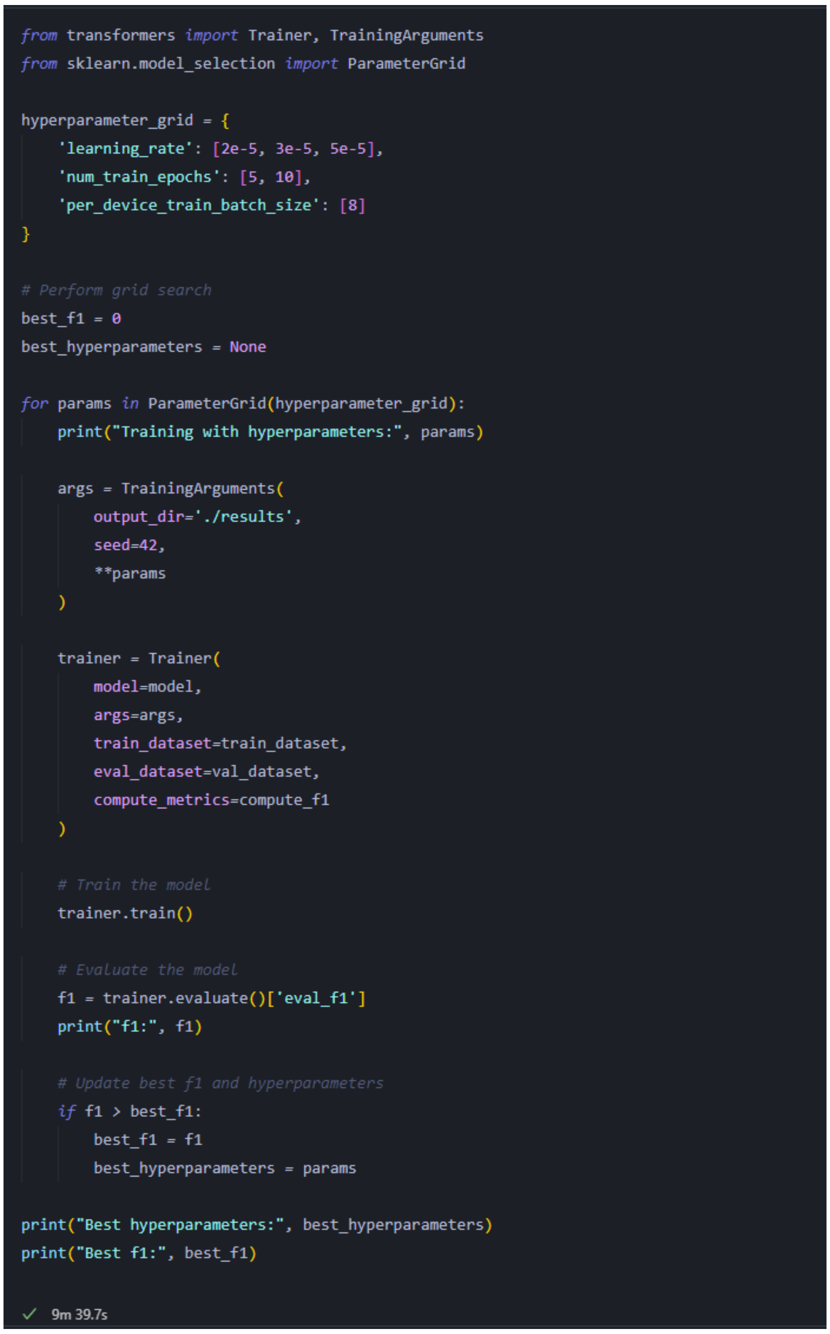


Submission and Description		Private Score	Public Score	Selected
result.csv Complete (after deadline) · 7d ago	hyperparameter tuning + random params	0.66572	0.57007	☐
result.csv Complete · 7d ago		0.65561	0.56804	☐
result.csv Complete · 7d ago		0.62931	0.54953	☐
result.csv Complete · 7d ago	hyperparameter tuning	0.54223	0.56777	☐
result.csv Complete · 7d ago		0.54223	0.56777	☐
result.csv Complete · 7d ago		0.46734	0.47525	☐
result.csv Complete · 7d ago		0.5713	0.53874	☐
result.csv Complete · 7d ago		0.52243	0.42155	☐
result3.csv Complete · 7d ago	no tuning	0.1145	0.10175	☐
result2.csv Complete · 7d ago		0.19027	0.19654	☐
results.csv Complete · 7d ago		0.11623	0.11078	☐

Chapter 5: Discussion

As I mentioned before, in my very first attempt, I didn't perform any hyperparameter tuning, led to a low f1-score (first section).

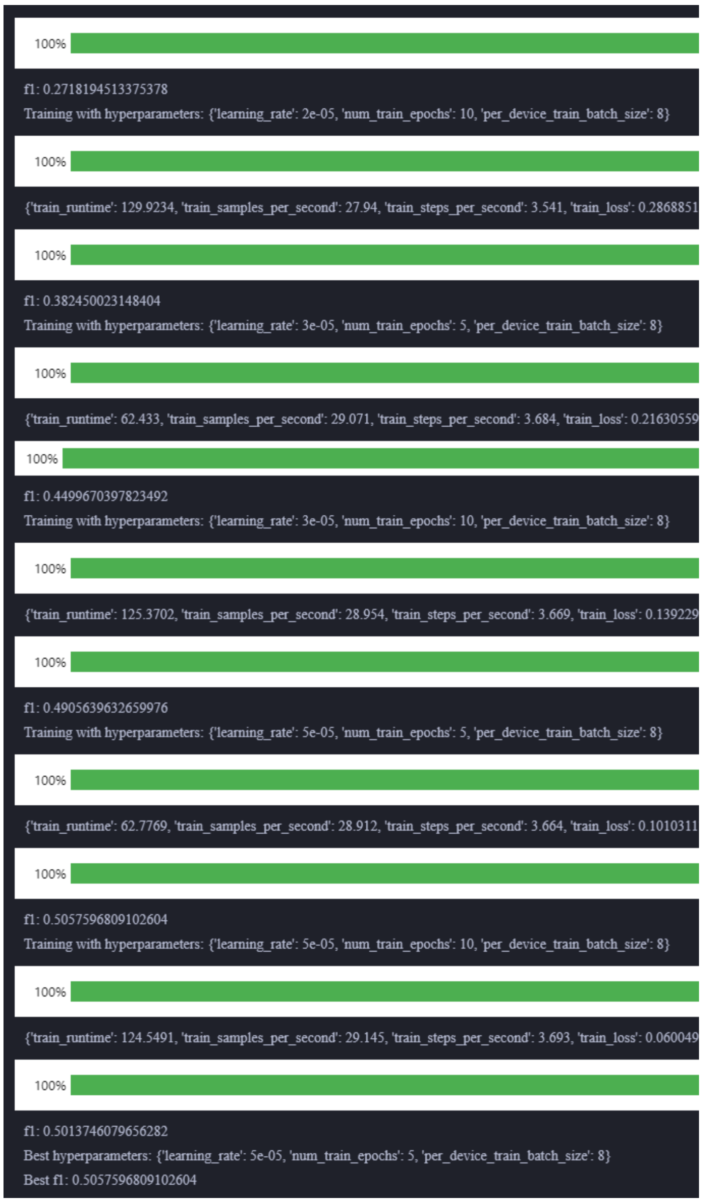
Afterward, I attempted hyperparameter tuning by performing a grid search. So I coded the following.



```python
from transformers import Trainer, TrainingArguments
from sklearn.model_selection import ParameterGrid

hyperparameter_grid = {
    'learning_rate': [2e-5, 3e-5, 5e-5],
    'num_train_epochs': [5, 10],
    'per_device_train_batch_size': [8]
}

# Perform grid search
best_f1 = 0
best_hyperparameters = None

for params in ParameterGrid(hyperparameter_grid):
    print("Training with hyperparameters:", params)

    args = TrainingArguments(
        output_dir='./results',
        seed=42,
        **params
    )

    trainer = Trainer(
        model=model,
        args=args,
        train_dataset=train_dataset,
        eval_dataset=val_dataset,
        compute_metrics=compute_f1
    )

    # Train the model
    trainer.train()

    # Evaluate the model
    f1 = trainer.evaluate()['eval_f1']
    print("f1:", f1)

    # Update best f1 and hyperparameters
    if f1 > best_f1:
        best_f1 = f1
        best_hyperparameters = params

print("Best hyperparameters:", best_hyperparameters)
print("Best f1:", best_f1)
```

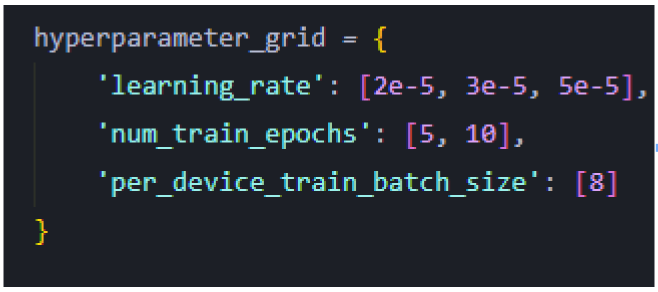
✓ 9m 39.7s

For each combination, I will calculate the F1 Score and store the best F1 Score to best_f1 and its parameters to best_hyperparameters. After iterating through all possible parameter combinations, it will print out the best one.

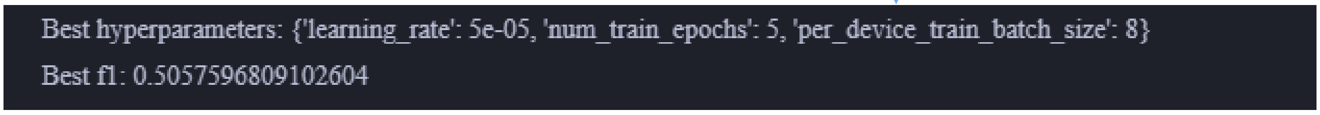


```
100%
f1: 0.2718194513375378
Training with hyperparameters: {'learning_rate': 2e-05, 'num_train_epochs': 10, 'per_device_train_batch_size': 8}
100%
{'train_runtime': 129.9234, 'train_samples_per_second': 27.94, 'train_steps_per_second': 3.541, 'train_loss': 0.2868851
100%
f1: 0.382450023148404
Training with hyperparameters: {'learning_rate': 3e-05, 'num_train_epochs': 5, 'per_device_train_batch_size': 8}
100%
{'train_runtime': 62.433, 'train_samples_per_second': 29.071, 'train_steps_per_second': 3.684, 'train_loss': 0.21630559
100%
f1: 0.4499670397823492
Training with hyperparameters: {'learning_rate': 3e-05, 'num_train_epochs': 10, 'per_device_train_batch_size': 8}
100%
{'train_runtime': 125.3702, 'train_samples_per_second': 28.954, 'train_steps_per_second': 3.669, 'train_loss': 0.139229
100%
f1: 0.4905639632659976
Training with hyperparameters: {'learning_rate': 5e-05, 'num_train_epochs': 5, 'per_device_train_batch_size': 8}
100%
{'train_runtime': 62.7769, 'train_samples_per_second': 28.912, 'train_steps_per_second': 3.664, 'train_loss': 0.1010311
100%
f1: 0.5057596809102604
Training with hyperparameters: {'learning_rate': 5e-05, 'num_train_epochs': 10, 'per_device_train_batch_size': 8}
100%
{'train_runtime': 124.5491, 'train_samples_per_second': 29.145, 'train_steps_per_second': 3.693, 'train_loss': 0.060049
100%
f1: 0.5013746079656282
Best hyperparameters: {'learning_rate': 5e-05, 'num_train_epochs': 5, 'per_device_train_batch_size': 8}
Best f1: 0.5057596809102604
```

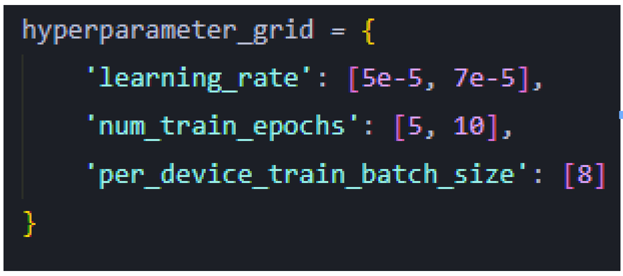
After this, I try to performing hyperparameter tuning multiple times by changing the hyperparameters configuration. Instead of putting all possible parameter into a list, this approach helps to avoid taking a long time to iterate through all combinations. Here are some of my result.



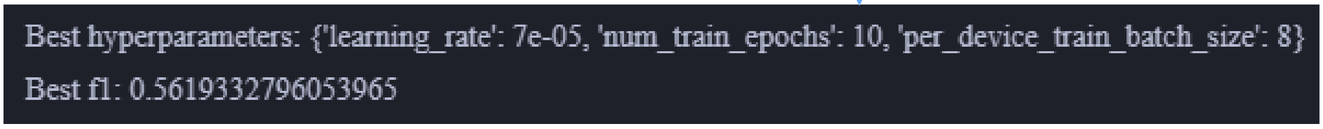
```
hyperparameter_grid = {
    'learning_rate': [2e-5, 3e-5, 5e-5],
    'num_train_epochs': [5, 10],
    'per_device_train_batch_size': [8]
}
```



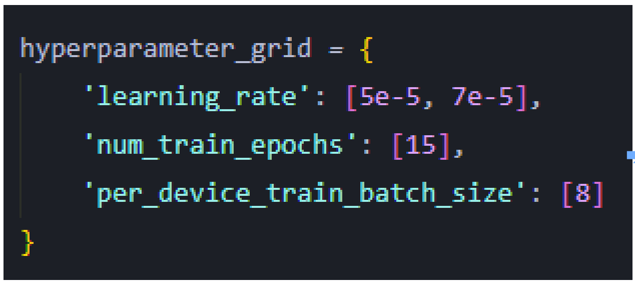
```
Best hyperparameters: {'learning_rate': 5e-05, 'num_train_epochs': 5, 'per_device_train_batch_size': 8}
Best f1: 0.5057596809102604
```



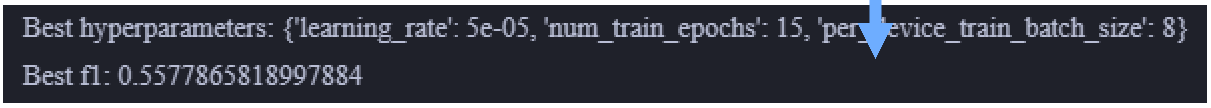
```
hyperparameter_grid = {
    'learning_rate': [5e-5, 7e-5],
    'num_train_epochs': [5, 10],
    'per_device_train_batch_size': [8]
}
```



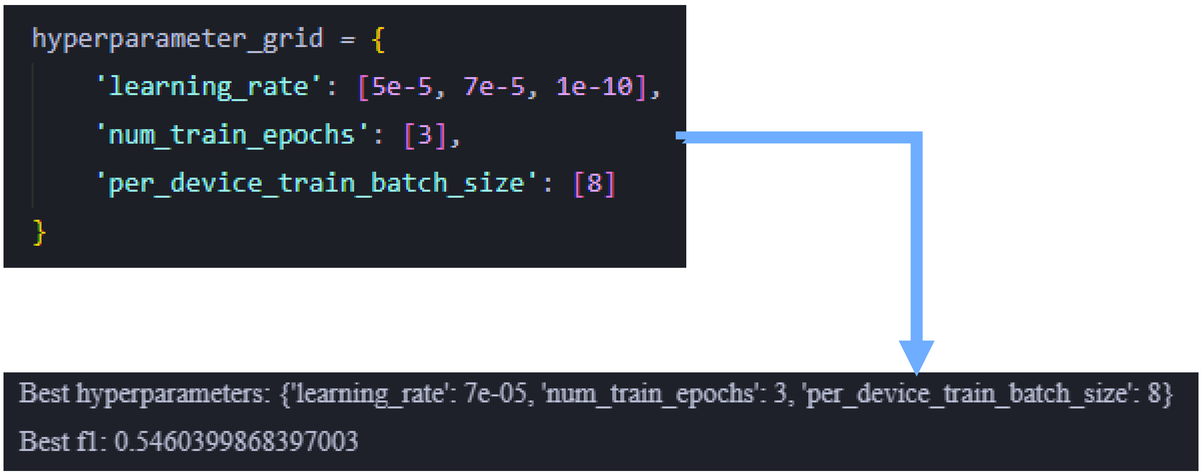
```
Best hyperparameters: {'learning_rate': 7e-05, 'num_train_epochs': 10, 'per_device_train_batch_size': 8}
Best f1: 0.5619332796053965
```



```
hyperparameter_grid = {
    'learning_rate': [5e-5, 7e-5],
    'num_train_epochs': [15],
    'per_device_train_batch_size': [8]
}
```



```
Best hyperparameters: {'learning_rate': 5e-05, 'num_train_epochs': 15, 'per_device_train_batch_size': 8}
Best f1: 0.5577865818997884
```



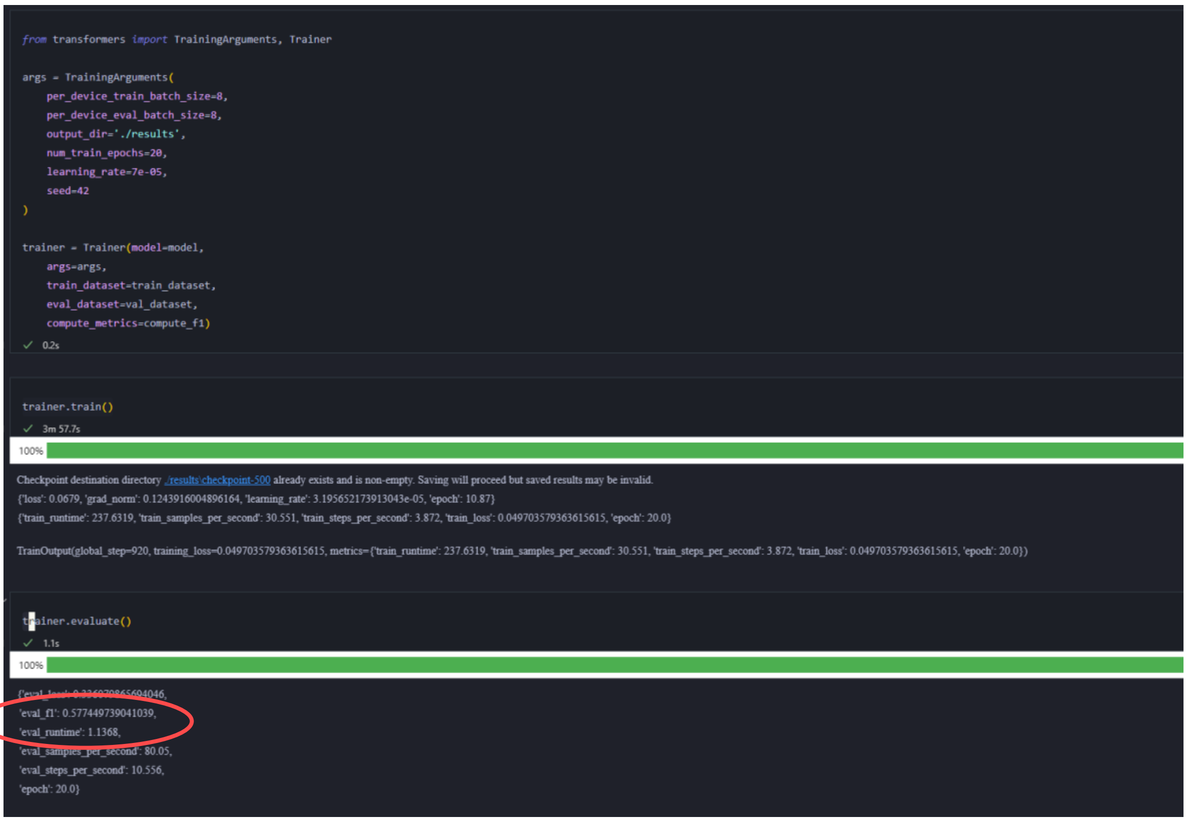
```
hyperparameter_grid = {
    'learning_rate': [5e-5, 7e-5, 1e-10],
    'num_train_epochs': [3],
    'per_device_train_batch_size': [8]
}
```

```
Best hyperparameters: {'learning_rate': 7e-05, 'num_train_epochs': 3, 'per_device_train_batch_size': 8}
Best f1: 0.5460399868397003
```

After performing multiple grid searches, I gained a lot of information about hyperparameters, which led me to section 3.

Because I didn't have enough time (it was 23:30 at that moment), I randomly used information based on my previous result of the hyperparameters. it suggested that num_train_epochs should be between 15 and 20 while learning rate should be between 5e-5 and 10e-5, and the batch_size should be fix at 8 instead of 16. With that information, I set train_epochs to 20 and tried many learning rate values.

After trying multiple learning rates, I found that a learning rate of 7e-5 gave me the highest F1-Score, as shown in the result.

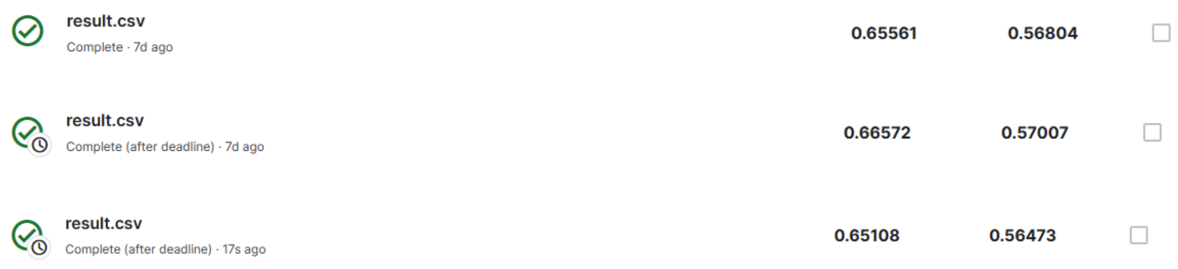


```
from transformers import TrainingArguments, Trainer

args = TrainingArguments(
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    output_dir='./results',
    num_train_epochs=20,
    learning_rate=7e-05,
    seed=42
)

trainer = Trainer(model=model,
    args=args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    compute_metrics=compute_f1)
✓ 0.2s
```

```
trainer.train()
✓ 3m 57.7s
100%
Checkpoint destination directory ./results\checkpoint-500 already exists and is non-empty. Saving will proceed but saved results may be invalid.
{'loss': 0.0679, 'grad_norm': 0.1243916004896164, 'learning_rate': 3.195652173913043e-05, 'epoch': 10.87}
{'train_runtime': 237.6319, 'train_samples_per_second': 30.551, 'train_steps_per_second': 3.872, 'train_loss': 0.049703579363615615, 'epoch': 20.0}

TrainOutput(global_step=920, training_loss=0.049703579363615615, metrics={'train_runtime': 237.6319, 'train_samples_per_second': 30.551, 'train_steps_per_second': 3.872, 'train_loss': 0.049703579363615615, 'epoch': 20.0})
```

```
trainer.evaluate()
✓ 1.1s
100%
{'eval_loss': 0.336070865694046,
'eval_f1': 0.577449739041039,
'eval_runtime': 1.1368,
'eval_samples_per_second': 80.05,
'eval_steps_per_second': 10.556,
'epoch': 20.0}
```

Then I trained my model several time with those hyperparameters and submit it to Kaggle.

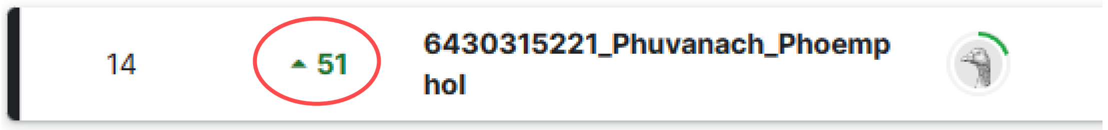


result.csv Complete · 7d ago	0.65561	0.56804	☐
result.csv Complete (after deadline) · 7d ago	0.66572	0.57007	☐
result.csv Complete (after deadline) · 17s ago	0.65108	0.56473	☐

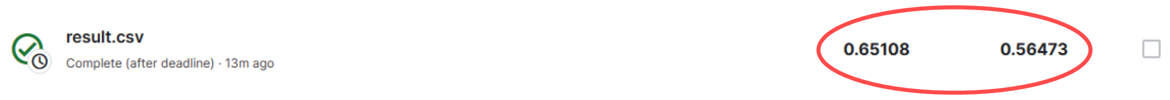
Chapter 6: Conclusion

BERT can perform robustly and effective in multi-label classification tasks. However, this does not mean only pre-trained model is sufficient. It still requires data processing and hyperparameters tuning as well.

I must admit that in this task, I have some luck in predicting the test data as well because there was a pretty large gap between my public and private F1-score. Additionally, I also have some luck to find appropriate learning rate value for the corresponding epochs and batch_size in time.



14 ▲ 51 6430315221_Phuvanach_Phoemphol



result.csv
Complete (after deadline) · 13m ago 0.65108 0.56473

However, if I had more time, I believe I could find even better hyperparameter suitable for this task, but it would still take a lot of time for Grid Search as well.