

BERT Multi-label Engineer

Nopparuj Poonsubanan

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Data Preparation</b>	<b>3</b>
2.0.1	Import . . . . .	3
2.0.2	turn json to csv file . . . . .	3
2.0.3	Load Data . . . . .	4
2.0.4	Count the number of words in each article. . . . .	4
2.0.5	Plot histogram of word counts . . . . .	5
2.0.6	Plot the frequency of each label in the dataset . . . . .	6
2.0.7	Plot number of labels per articles . . . . .	7
2.0.8	Clean the data and save it to a CSV file. . . . .	8
<b>3</b>	<b>Model</b>	<b>11</b>
3.0.1	Connect to google drive . . . . .	11
3.0.2	Import & set seed . . . . .	11
3.0.3	Set hyperparameters . . . . .	12
3.0.4	Load model . . . . .	12
3.0.5	Prepare the data . . . . .	12
3.0.6	Set the computation metrics to be F1 score and Hamming loss. . . . .	13
3.0.7	Training . . . . .	13
<b>4</b>	<b>Results</b>	<b>15</b>
4.0.1	Save model . . . . .	15
4.0.2	Predictions . . . . .	16
<b>5</b>	<b>Discussion</b>	<b>18</b>
<b>6</b>	<b>Conclusion</b>	<b>19</b>

# Chapter 1

## Introduction

This project is part of the Data Science and Data Engineering midterm take-home exam. The objective is to develop a machine learning model that can multi label articles into 18 different classes according to titles and abstracts. This classification is based on a small dataset from Scopus that includes eighteen engineering subject areas: mechanical, electrical, computer, optical, nano, chemical, materials, biomedical, environmental, industrial, safety, mathematics and statistics, and materials science. The model's performance is evaluated using the macro F1 score.

## Chapter 2

# Data Preparation

Initially, we downloaded the data in json format which included id, titles, abstracts, and labels. Firstly we combined abstract and title into a single text column, followed by one-hot encoding the labels. Secondly, we examined the data to find and eliminate the outlier from the dataset. After that, the data was divided with a test ratio of 0.1 into training and test sets.

### 2.0.1 Import

```
[ ]: import os
import json
import csv
import pandas as pd
import seaborn as sns
```

### 2.0.2 turn json to csv file

```
[ ]: labels = ["CE", "ENV", "BME", "PE", "METAL", "ME", "EE", "CPE", "OPTIC",
              "NANO", "CHE", "MATENG", "AGRI", "EDU", "IE", "SAFETY", "MATH",
              ↪ "MATSCI"]

def train_to_csv(r, w):
    with open(r, "r") as f:
        data = json.load(f)

    with open(w, "w", newline='', encoding='utf-8') as csv_file:
        fieldnames = ["id", "text"] + labels
        writer = csv.DictWriter(csv_file, fieldnames=fieldnames)
        writer.writeheader()
        for key, value in data.items():
            row = {"id": key, "text": value['Title'] + ' ' +
            ↪ value["Abstract"]}
            for field in fieldnames[2:]:
                row[field] = 1 if field in value["Classes"] else 0
            writer.writerow(row)
```

```
def test_to_csv(r, w):
    with open(r, "r") as f:
        data = json.load(f)
    with open(w, "w", newline='', encoding='utf-8') as csv_file:
        fieldnames = ['id', 'text']
        writer = csv.DictWriter(csv_file, fieldnames=fieldnames)
        writer.writeheader()
        for key, value in data.items():
            row = {"id": key,
                  "text": value["Title"] + " " + value["Abstract"]}
            writer.writerow(row)
```

```
[ ]: os.chdir('..')
train_to_csv("data/student_json/train_for_student.json",
             "data/student_csv/train_for_student.csv")

test_to_csv("data/student_json/test_for_student.json",
            "data/student_csv/test_for_student.csv")
```

### 2.0.3 Load Data

```
[ ]: train_data = pd.read_csv('data/student_csv/train_for_student.csv')
test_data = pd.read_csv('data/student_csv/test_for_student.csv')
```

### 2.0.4 Count the number of words in each article.

```
[ ]: train_data['count_word'] = train_data['text'].apply(lambda x: len(x.split()))
test_data['count_word'] = test_data['text'].apply(lambda x: len(x.split()))
```

```
[ ]: train_data['count_word'].describe()
```

```
[10]: count      454.000000
      mean      192.096916
      std       58.923375
      min       63.000000
      25%      149.250000
      50%      189.000000
      75%      224.750000
      max      464.000000
      Name: count_word, dtype: float64
```

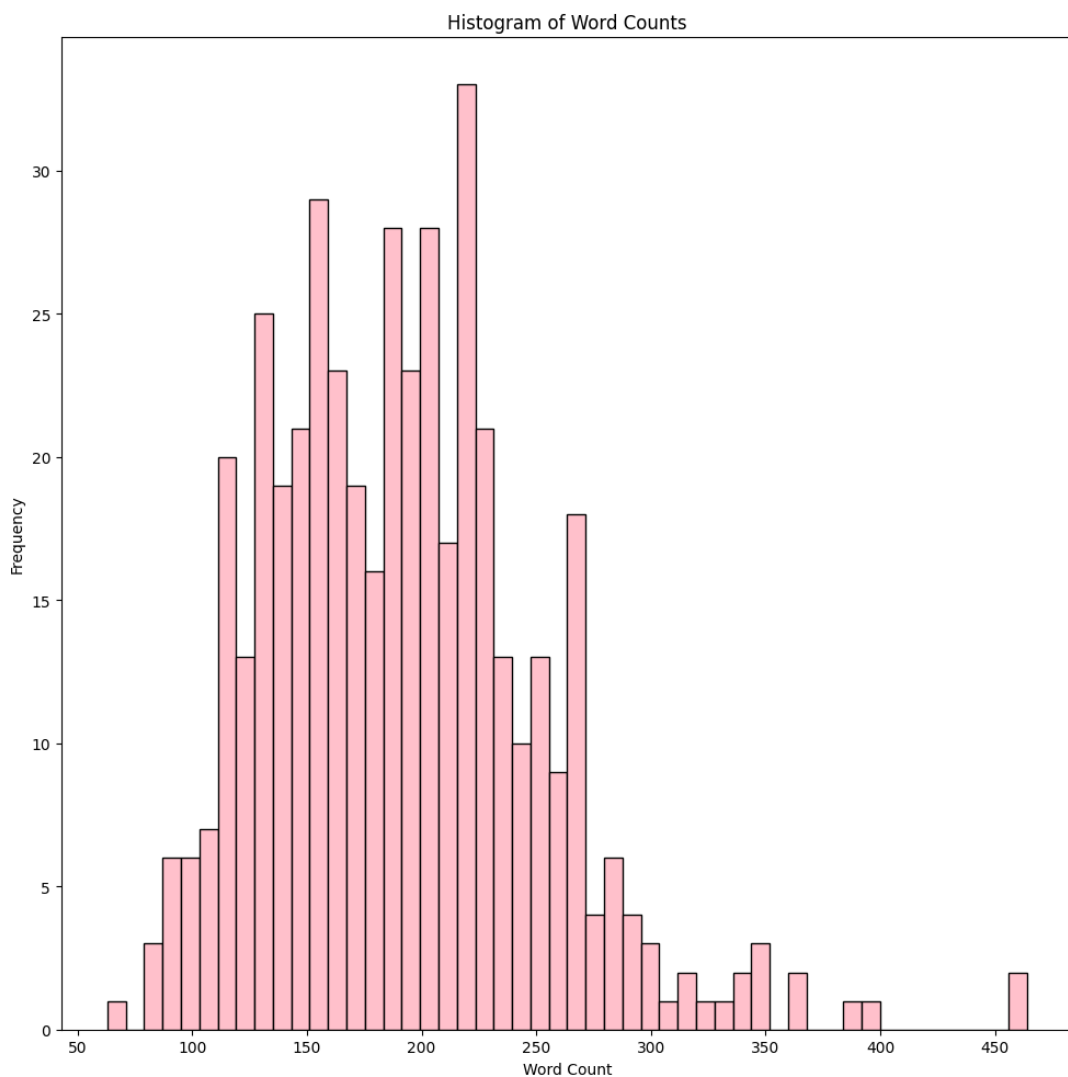
```
[ ]: test_data['count_word'].describe()
```

```
[ ]: count      151.000000
      mean      194.768212
      std       60.669866
      min       84.000000
      25%      146.500000
      50%      192.000000
      75%      232.000000
```

```
max      399.000000
Name: count_word, dtype: float64
```

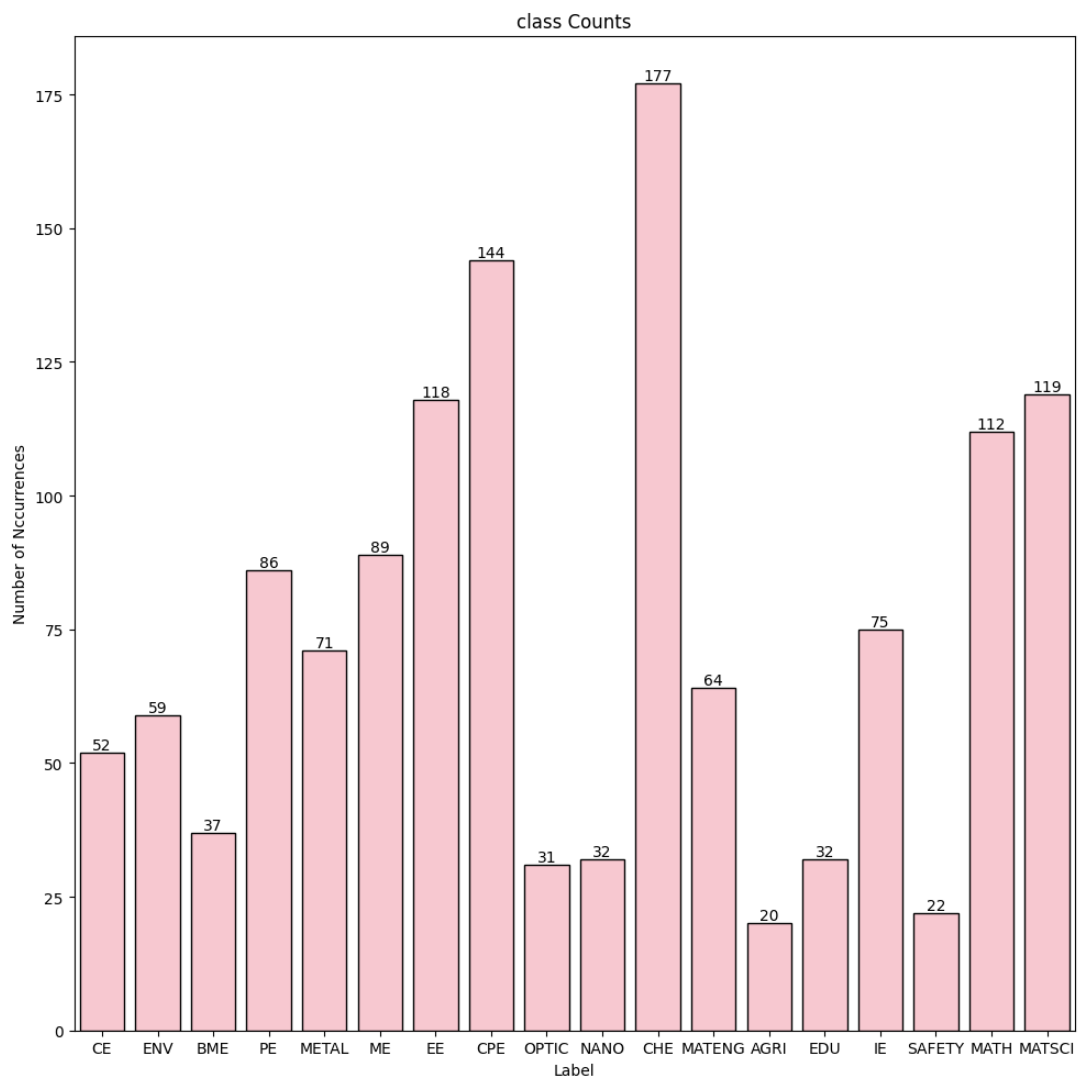
## 2.0.5 Plot histogram of word counts

```
[ ]: import matplotlib.pyplot as plt
x = train_data['count_word']
plt.figure(figsize=(10, 10))
plt.xlabel('Word Count')
plt.ylabel('Frequency')
plt.title('Histogram of Word Counts')
plt.hist(x, bins=50, color='pink', edgecolor='black')
plt.tight_layout()
plt.show()
```



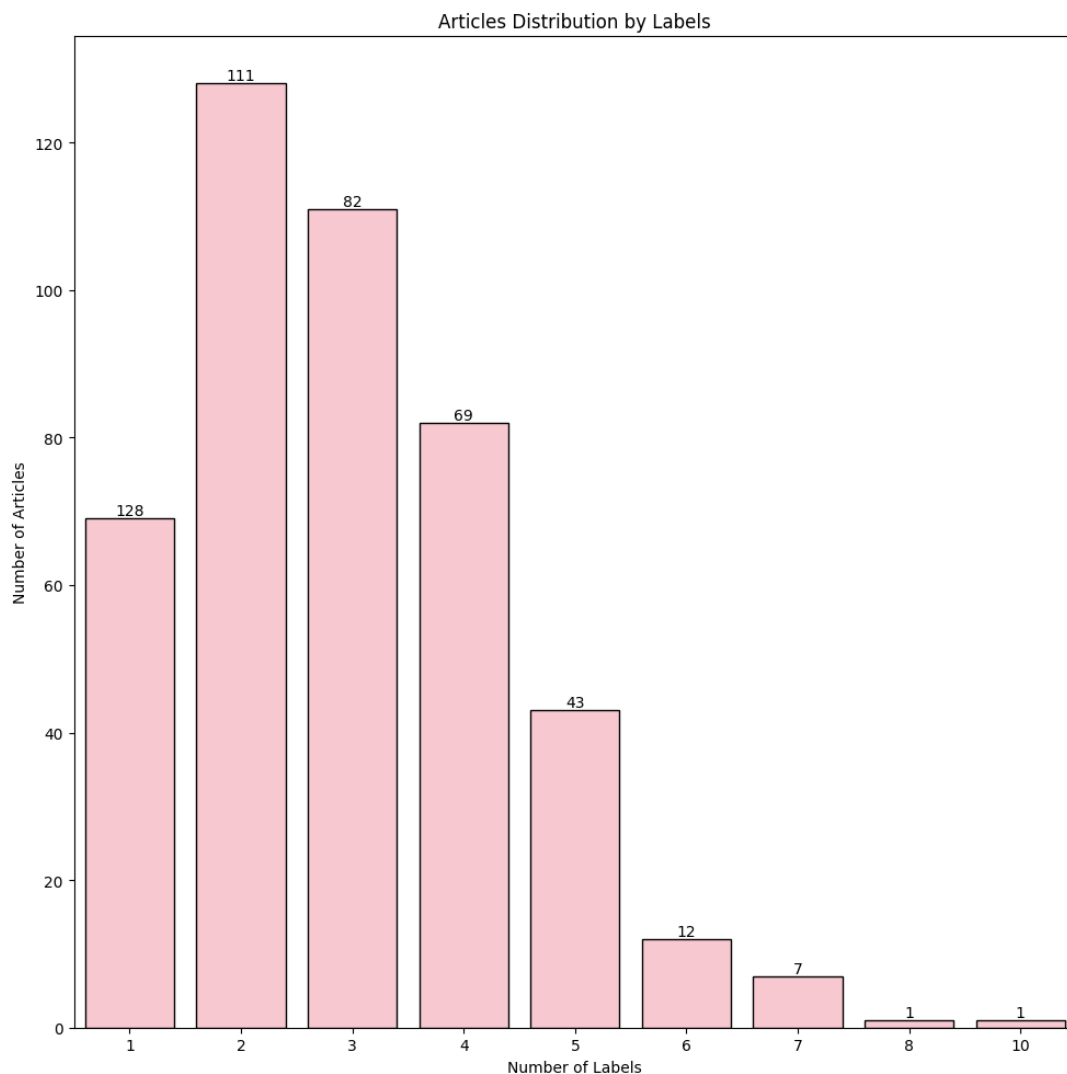
## 2.0.6 Plot the frequency of each label in the dataset

```
[ ]: x = train_data.iloc[:, 2:-1].sum()
plt.figure(figsize=(10, 10))
plt.title("class Counts")
plt.ylabel('Number of Nccurrences')
plt.xlabel('Label')
ax = sns.barplot(x=x.index, y=x.values, color='pink', edgecolor='black')
for rect, label in zip(ax.patches, x.values):
    ax.text(rect.get_x() + rect.get_width()/2,
            rect.get_height(), label, ha='center', va='bottom')
plt.tight_layout()
plt.show()
```



## 2.0.7 Plot number of labels per articles

```
[ ]: x = train_data.iloc[:, 2:-1].sum(axis=1).value_counts()
plt.figure(figsize=(10, 10))
plt.title('Articles Distribution by Labels')
plt.ylabel('Number of Articles')
plt.xlabel('Number of Labels')
ax = sns.barplot(x=x.index, y=x.values, color="pink", edgecolor="black")
for rect, label in zip(ax.patches, x.values):
    ax.text(rect.get_x() + rect.get_width()/2,
            rect.get_height(), label, ha='center', va='bottom')
plt.tight_layout()
plt.show()
```



After investigating the data, we've identified outliers. Specifically, we need to remove articles that have more than seven labels and 256 words, remove numbers in the text, remove stop words, and lemmatize words. Upon reviewing the data, we noticed the presence of many specific words, so we will delete those as well.



## 2.0.8 Clean the data and save it to a CSV file.

```
[]: import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

def clean_text(text):
    sw = {'abstract', 'study', 'studied', 'research', 'publish', 'published',
    → 'association', 'iop', 'ltd', 'publishing', 'jsme', 'authors', 'author',
    → 'elsevier', 'present', 'presented', 'license', 'licence', 'institute',
    → 'moreover', 'assume', 'assumed', 'ieeever', 'b', 'v', 'ieice', 'asme',
    → 'iii', 'aidic', 'servizi', 'nd', 'st', 'th', 'congress',
    → 'international', 'project', 'acm', 'turkiye', 'bpmn', 'paper',
    → 'ltdnew', 'ltdhouse', 'Thomas', 'lorawan', 'fuji', 'anchale',
    → 'tresatayawed', 'john', 'wiley', 'ieee', 'ieeethe', 'ieeethis', 'recent',
    → 'america', 'american', 'thai', 'thailand', 'saraburi', 'province',
    → 'switzerland', 'phitsanulok', 'bangkok', 'provinces', 'nan', 'saraburi',
    → 'samutprakarn'}
    text = re.sub(r'[~a-zA-Z\s]', ' ', text)
    text = text.lower()
    tokens = word_tokenize(text)
    stop_words = set(stopwords.words('english'))
    stop_words.update(sw)
    tokens = [token for token in tokens if token not in stop_words]
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(token) for token in tokens]
    text = ' '.join(tokens)
    return text

def train_to_csv_clean(r, w):
    with open(r, "r") as f:
        data = json.load(f)

    with open(w, "w", newline='', encoding='utf-8') as csv_file:
        fieldnames = ["id", "text"] + labels
        writer = csv.DictWriter(csv_file, fieldnames=fieldnames)
        writer.writeheader()
        for key, value in data.items():
            label_count = sum( 1 for field in fieldnames[2:] if field in
    → value["Classes"])
            cleaned_text = clean_text(value["Title"] + " " +
    → value["Abstract"])
            if label_count <= 7 and len(cleaned_text.split()) <= 256:
                row = {"id": key,
                    "text": cleaned_text}
                for field in fieldnames[2:]:
                    row[field] = 1 if field in value["Classes"] else 0
                writer.writerow(row)
```

```
def test_to_csv_clean(r, w):
    with open(r, "r") as f:
        data = json.load(f)
    with open(w, "w", newline='', encoding='utf-8') as csv_file:
        fieldnames = ['id', 'text']
        writer = csv.DictWriter(csv_file, fieldnames=fieldnames)
        writer.writeheader()
        for key, value in data.items():
            row = {"id": key,
                  "text": clean_text(value["Title"] + " " +
→value["Abstract"])}
            writer.writerow(row)
```

```
[ ]: train_to_csv("data/student_json/train_for_student.json",
                  "data/student_csv/train_for_student.csv")

test_to_csv("data/student_json/test_for_student.json",
            "data/student_csv/test_for_student.csv")
```

```
[ ]: train_data_clean = pd.read_csv('data/clean/train_for_student_clean.csv')
test_data_clean = pd.read_csv('data/clean/test_for_student_clean.csv')
train_data_clean['count_word'] = train_data_clean['text'].apply(lambda x:
→len(x.split()))
test_data_clean['count_word'] = train_data_clean['text'].apply(lambda x:
→len(x.split()))
```

```
[ ]: train_data_clean['count_word'].describe()
```

```
[ ]: count      451.000000
mean      113.079823
std       33.427505
min       43.000000
25%       88.000000
50%      110.000000
75%      134.500000
max       246.000000
Name: count_word, dtype: float64
```

```
[ ]: test_data_clean['count_word'].describe()
```

```
[ ]: count      151.000000
mean      115.867550
std       35.171328
min       43.000000
25%       88.500000
50%      111.000000
75%      137.500000
max       239.000000
Name: count_word, dtype: float64
```

Upload the cleaned file to Google Drive to train it on Colab.

## Chapter 3

# Model

For this project, we utilized the bert-base-uncased model from Google. BERT, short for Bidirectional Encoder Representations from Transformers, is a model pre-trained on a vast corpus of English data in a self-supervised manner. This pre-training involved learning from raw text without any explicit human labeling. The choice of BERT is predicated on its ability to understand the context of words in a sentence by considering the words that come before and after, making it highly effective for tasks requiring a deep understanding of language nuances. The model's performance was measured using the macro F1 score.

```
[ ]: !pip install -U accelerate
      !pip install -U transformers
      !pip install datasets
```

### 3.0.1 Connect to google drive

```
[ ]: from google.colab import drive
      import os
      drive.mount('/content/drive')
      os.chdir('drive/My Drive/contents')
```

Mounted at /content/drive

### 3.0.2 Import & set seed

```
[ ]: seed = 21

      import torch
      torch.manual_seed(seed)
      torch.use_deterministic_algorithms(True)

      import random
      random.seed(seed)

      import numpy as np
      np.random.seed(seed)
```

```
import transformers
transformers.enable_full_determinism(seed)
```

```
[ ]: import pandas as pd
from datasets import load_dataset
from sklearn.metrics import f1_score, hamming_loss
from transformers import BertForSequenceClassification, BertTokenizer, TrainingArguments, Trainer
```

### 3.0.3 Set hyperparameters

```
[ ]: checkpoint = "bert-base-uncased"
data_path = "train_for_student_clean.csv"
labels = [ "CE", "ENV", "BME", "PE", "METAL", "ME", "EE", "CPE",
    ↳ "OPTIC", "NANO", "CHE", "MATENG", "AGRI", "EDU", "IE", "SAFETY", "MATH",
    ↳ "MATSCI"]
MAX_LEN = 246
TRAIN_BATCH_SIZE = 2
VALID_BATCH_SIZE = 2
TEST_BATCH_SIZE = 2
EPOCHS = 21
LEARNING_RATE = 5e-05
THRESHOLD = 0.3
WEIGHT_DECAY = 0.001
```

### 3.0.4 Load model

```
[ ]: tokenizer = BertTokenizer.from_pretrained(checkpoint)
model = BertForSequenceClassification.from_pretrained(checkpoint,
    ↳ num_labels=len(labels),
    ↳ problem_type="multi_label_classification")
```

### 3.0.5 Prepare the data

```
[ ]: data = load_dataset("csv", data_files=data_path)
data = data["train"].train_test_split(test_size=0.1, seed=seed)

[ ]: def preprocess_function(examples):
    tokenized_inputs = tokenizer(examples["text"], padding="max_length",
    ↳ truncation=True, max_length=MAX_LEN)
    label_dict = {}
    for label in labels:
        label_dict[label] = examples[label]
    tokenized_inputs["labels"] = [[float(label_dict[label][i]) for label in
    ↳ labels] for i in range(len(examples["text"]))]
    return tokenized_inputs

tokenized_data = data.map(preprocess_function, batched=True)
```

### 3.0.6 Set the computation metrics to be F1 score and Hamming loss.

```
[ ]: def compute_metrics(eval_pred):
    predictions, ref = eval_pred
    sigmoid = torch.nn.Sigmoid()
    probs = sigmoid(torch.Tensor(predictions))

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

    metrics = {
        "f1": f1_score(ref, y_pred, average = 'macro', zero_division=0),
        "hamming_loss": hamming_loss(ref, y_pred),
    }
    return metrics
```

### 3.0.7 Training

```
[ ]: training_args = TrainingArguments(
    output_dir = './results',
    evaluation_strategy="epoch",
    learning_rate=LEARNING_RATE,
    per_device_train_batch_size=TRAIN_BATCH_SIZE,
    per_device_eval_batch_size=VALID_BATCH_SIZE,
    num_train_epochs=EPOCHS,
    save_total_limit=1,
    fp16=True,
)
```

```
[ ]: trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_data["train"],
    eval_dataset=tokenized_data["test"],
    tokenizer=tokenizer,
    compute_metrics=compute_metrics,
)
```

```
[ ]: trainer.train()
```

```
[ ]: TrainOutput(global_step=4263, training_loss=0.1410284964042232,
metrics={'train_runtime': 1054.9406, 'train_samples_per_second': 8.062,
'train_steps_per_second': 4.041, 'total_flos': 1075327978966920.0,
→ 'train_loss':
0.1410284964042232, 'epoch': 21.0})
```

Epoch	Training Loss	Validation Loss	F1	Hamming Loss
1	No log	0.314494	0.224889	0.150966
2	No log	0.290898	0.297800	0.143720
3	0.368000	0.264773	0.466495	0.117150
4	0.368000	0.247246	0.479391	0.108696
5	0.266800	0.227199	0.570766	0.101449
6	0.266800	0.239932	0.577966	0.121981
7	0.266800	0.231628	0.592698	0.099034
8	0.189300	0.239384	0.637211	0.103865
9	0.189300	0.236545	0.608009	0.100242
10	0.129300	0.241901	0.557948	0.106280
11	0.129300	0.236051	0.620834	0.092995
12	0.129300	0.242032	0.586037	0.100242
13	0.085600	0.242005	0.590316	0.103865
14	0.085600	0.240049	0.620831	0.094203
15	0.059900	0.242787	0.582370	0.096618
16	0.059900	0.241507	0.620282	0.096618
17	0.059900	0.239307	0.569547	0.095411
18	0.046200	0.243935	0.602460	0.092995
19	0.046200	0.243870	0.608433	0.090580
20	0.038500	0.247260	0.597618	0.095411
21	0.038500	0.246206	0.606484	0.091787

## Chapter 4

# Results

Upon training, the model achieved a macro F1 score of 0.6008 on the test set. The model was then used to predict classifications on a real testing dataset, which was submitted to Kaggle, and received a macro F1 score of 0.60388.

```
[ ]: trainer.evaluate()

[ ]: {'eval_loss': 0.23503780364990234,
      'eval_f1': 0.6008195667678425,
      'eval_hamming_loss': 0.08695652173913043,
      'eval_runtime': 1.3919,
      'eval_samples_per_second': 33.048,
      'eval_steps_per_second': 16.524,
      'epoch': 21.0}
```

### 4.0.1 Save model

```
[ ]: trainer.save_model("bert-multilabel-engineer")

[ ]: tokenizer.save_pretrained('bert-tokenizer')
```



## 4.0.2 Predictions

```
[ ]: text = "Comparative Electrical Energy Yield Performance of Micro-Inverter PV_
→Systems Using a Machine Learning Approach Based on a Mixed-Effect Model of_
→Real Datasets @ 2013 IEEE.Long-term energy evaluation of PV systems that_
→use micro-inverter configuration (micro-inverter PV systems) is currently_
→unclear due to the lacking of sufficient longitudinal measurement data and_
→appropriate analysis method. The poor knowledge about impact and aging of_
→micro-inverter PV system affects the comprehension and accuracy of PV_
→design and simulation tools. In this paper, we propose a machine learning_
→approach based on the mixed-effect model to compare and evaluate the_
→electrical energy yield of micro-inverter PV systems. The analyzed results_
→using a 5-year period data of PV stations located at Concord,_
→Massachusetts, USA showed that there is no significant difference in yearly_
→electrical energy yield of micro-inverter PV systems under shading and_
→non-shading condition. This finding has confirmed the advantage of_
→micro-inverter PV system over the other PV types. In addition, our work is_
→the first study that identified the average degradation rate of_
→micro-inverter PV of 3% per year (95% confidence intervals: 2%-4.3%). Our_
→findings in this study have extended substantially the comprehensive_
→understanding of micro-inverter PV system."

encoding = tokenizer(text, return_tensors='pt')
encoding.to(trainer.model.device)

outputs = trainer.model(**encoding)

[ ]: sigmoid = torch.nn.Sigmoid()
probs = sigmoid(outputs.logits[0].cpu())
preds = np.zeros(probs.shape)
preds[np.where(probs>=0.3)] = 1
preds

[ ]: array([0., 0., 0., 1., 0., 0., 1., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
0.])
```


```
[ ]: test_data = pd.read_csv('./test_for_student_clean.csv')

def predict_labels(text):
    encoding = tokenizer(text,
                          return_tensors='pt',
                          padding="max_length",
                          max_length=MAX_LEN,
                          truncation=True
                          )
    encoding.to(trainer.model.device)

    outputs = trainer.model(**encoding)
    sigmoid = torch.nn.Sigmoid()
    probs = sigmoid(outputs.logits[0].cpu())
    preds = np.zeros(probs.shape)
    preds[np.where(probs >= 0.3)] = 1
    return preds.astype(int)

prediction_data = []
for idx, row in test_data.iterrows():
    id_ = row['id']
    text = row['text']
    predictions = predict_labels(text).tolist()
    prediction_data.append([id_] + predictions)

prediction_df = pd.DataFrame(prediction_data, columns=['id'] + labels)
prediction_path = 'prediction.csv'
prediction_df.to_csv(prediction_path, index=False)
```

#	△	Team	Members	Score	Entries	Last	Solution
51	→ 30	6330261921_Nopparuj_Poons ubana		0.60388	31	5d	

## Chapter 5

# Discussion

After we reviewed the dataset, we can see several issues. Notably, numerous articles were assigned more than four labels, which is an unlikely scenario for one article to cover that many fields. Furthermore, some articles were mislabeled, such as those discussing herbal studies but classified under mechanical engineering, highlighting inconsistencies and irrelevance in the labeling. These observations suggest that the dataset requires further cleaning and accurate relabeling to improve model performance. Furthermore, our current approach utilizes the bert-base-uncased model, which is not case-sensitive. In future iterations, it might be advantageous to utilize bert-large-cased, which is pre-trained on a larger dataset and retains case sensitivity. This could be beneficial for datasets containing a lot of specific names and terms where capitalization could provide additional context or distinguish between acronyms and common words, potentially improving the model's accuracy and understanding of the subject matter.

## Chapter 6

# Conclusion

This project demonstrates the potential of advanced machine learning models like BERT for classifying engineering articles. Despite data quality challenges, the model achieved respectable performance. Additionally, exploring more sophisticated data preprocessing and model tuning could further enhance the model's ability to understand and classify complex engineering topics accurately. Finally, this project underscores the importance of both high-quality data and advanced machine learning techniques in the developing field of text classification.