Midterm Model Report

Prepared By

Chanatip Kowsurat 6530075721

Proposed

Ass. Prof. Dr.	Veera	Muangsin
Asst. Prof. Dr.	Natawut	Nupairoj
Ass. Prof. Dr.	Peerapon	Vateekul

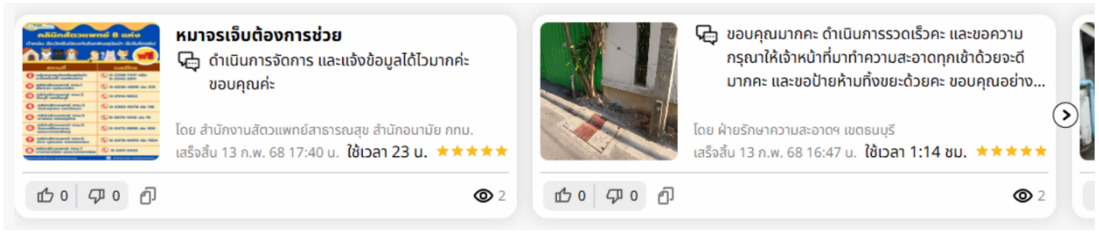
This report is a part of 2110446 Data Science and Data Engineering
Chulalongkorn University
Second Semester, Academic Year 2025

Table of content

Introduction

In Thailand, addressing public concerns through complaint reporting is vital for resolving various issues. The Traffy Fondue Dataset comprises citizen-submitted complaints, primarily from Bangkok, spanning multiple sectors. A key challenge is accurately classifying these complaints to determine the relevant organizations responsible for handling them.

This project focuses on multi-label classification of text-based complaints, mapping each complaint to one or more relevant organizations. The challenge lies in the fact that some complaints pertain to multiple entities, increasing the complexity of the classification process. The dataset consists of complaint descriptions along with their designated responsible organizations.



หมาจรเจ็บต้องการช่วย

ดำเนินการจัดการ และแจ้งข้อมูลได้ไวมากค่ะ ขอบคุณค่ะ

โดย สำนักงานสัตวแพทย์สาธารณสุข สำนักอนามัย กทม.

เสร็จสิ้น 13 ก.พ. 68 17:40 น. ใช้เวลา 23 น. ★★★★★

0 0 2

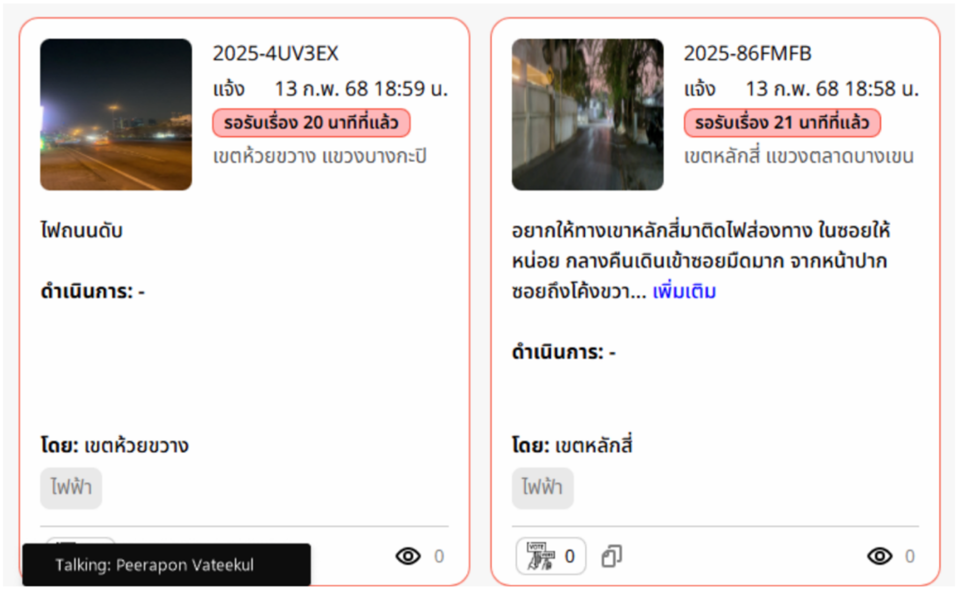
ขอบคุณมากคะ ดำเนินการรวดเร็วคะ และขอความกรุณาให้เจ้าหน้าที่มาทำความสะอาดทุกเช้าด้วยจะดีมากคะ และขอป้ายห้ามทิ้งขยะด้วยคะ ขอบคุณอย่าง...

โดย ฝ่ายรักษาความสะอาดฯ เขตธนบุรี

เสร็จสิ้น 13 ก.พ. 68 16:47 น. ใช้เวลา 1:14 ชม. ★★★★★

0 0 2

Fig. 1: Complaints in Traffy Fondue



2025-4UV3EX

แจ้ง 13 ก.พ. 68 18:59 น.

รอรับเรื่อง 20 นาทีที่แล้ว

เขตห้วยขวาง แขวงบางกะปิ

ไฟถนนดับ

ดำเนินการ: -

โดย: เขตห้วยขวาง

ไฟฟ้า

0

Talking: Peerapon Vateekul

2025-86FMFB

แจ้ง 13 ก.พ. 68 18:58 น.

รอรับเรื่อง 21 นาทีที่แล้ว

เขตหลักสี่ แขวงตลาดบางเขน

อยากให้ทางเขตหลักสี่มาติดไฟส่องทาง ในซอยให้หน่อย กลางคืนเดินเข้าซอยมืดมาก จากหน้าปากซอยถึงโค้งขวา... เพิ่มเติม

ดำเนินการ: -

โดย: เขตหลักสี่

ไฟฟ้า

0 0

Fig. 2: Complaints in Traffy Fondue

Data preparation

I visualized the dataset's distribution using graphs to choose preprocess approach before feeding it into the model.

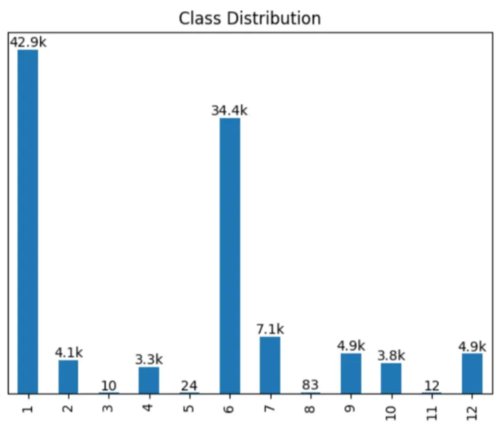


Fig. 3 : Class distribution graph before preprocess

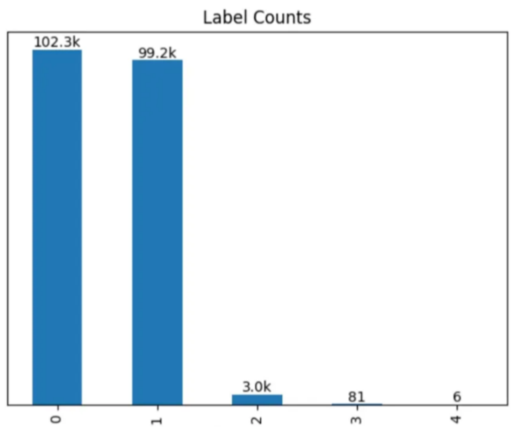


Fig. 4 : Label counts graph before preprocess

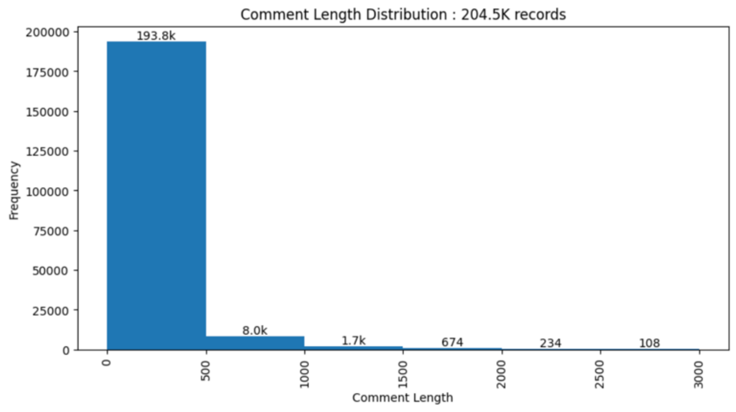


Fig. 5 : Comment length graph ranging from length 0 to 3,000

Due to the dataset's imbalance, as seen in Figures 1 and 2, I applied down sampling, reducing class 1 data by 60%, class 6 by 50%, and label 0 counts by 60%. Additionally, I performed up sampling, increasing class 3, 5, 8, and 11 data to 2,000 samples each and label counts of 2 to 5,000 samples.

From the comment length distribution, I observed that comment lengths range from 0 to 7,000 characters, with 95% of comments being under 1,000 characters. Therefore, I removed entries where the comment length exceeded 2,000 characters.

For text preprocessing, I removed emojis, URLs, repeated characters, and Thai stop words (e.g., ครับ, นะครับ, ค่ะ, นะคะ, น่ะคะ). However, this significantly reduced accuracy, causing the model to predict zero for all test data. As a result, I decided to keep the original text.

I structured the class label columns into a 12-element array containing 0s and 1s and added a label_counts column for further usage.

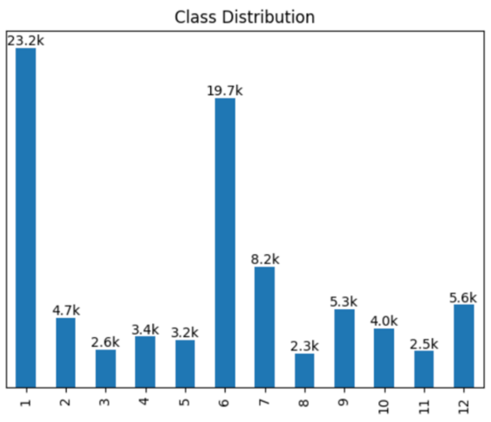


Fig. 6 : Class distribution graph after preprocess

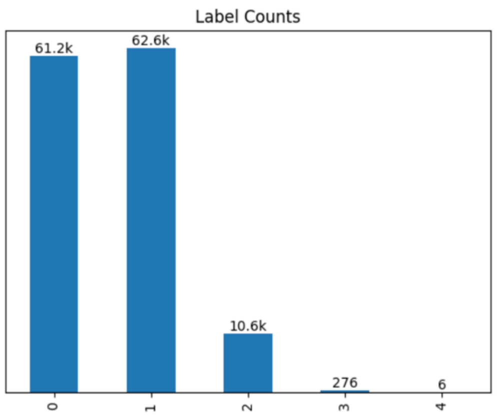
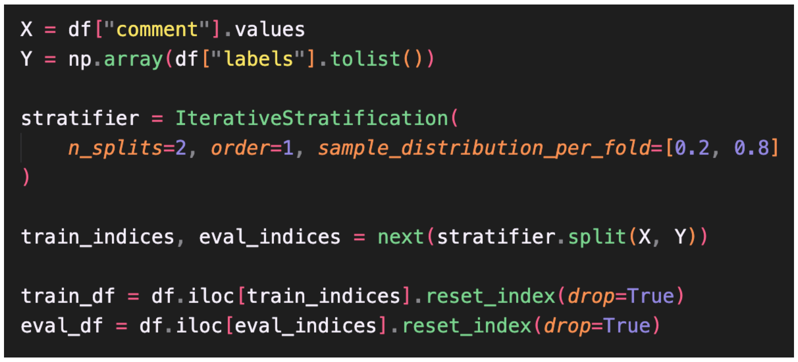


Fig. 7 : Label counts graph after preprocess

Finally, I divided the dataset into 80% for training and 20% for evaluation using IterativeStratification. Following this, the input comments were tokenized and transformed into a numerical format compatible with BERT. The processed dataset is then wrapped in a custom PyTorch dataset class and loaded into a DataLoader.



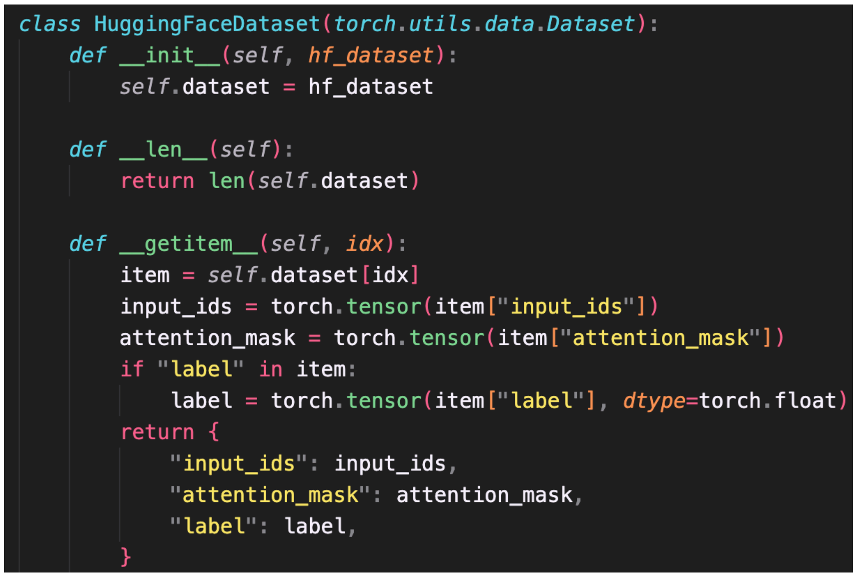
```
X = df["comment"].values
Y = np.array(df["labels"].tolist())

stratifier = IterativeStratification(
    n_splits=2, order=1, sample_distribution_per_fold=[0.2, 0.8]
)

train_indices, eval_indices = next(stratifier.split(X, Y))

train_df = df.iloc[train_indices].reset_index(drop=True)
eval_df = df.iloc[eval_indices].reset_index(drop=True)
```

Fig. 8 : Split dataset

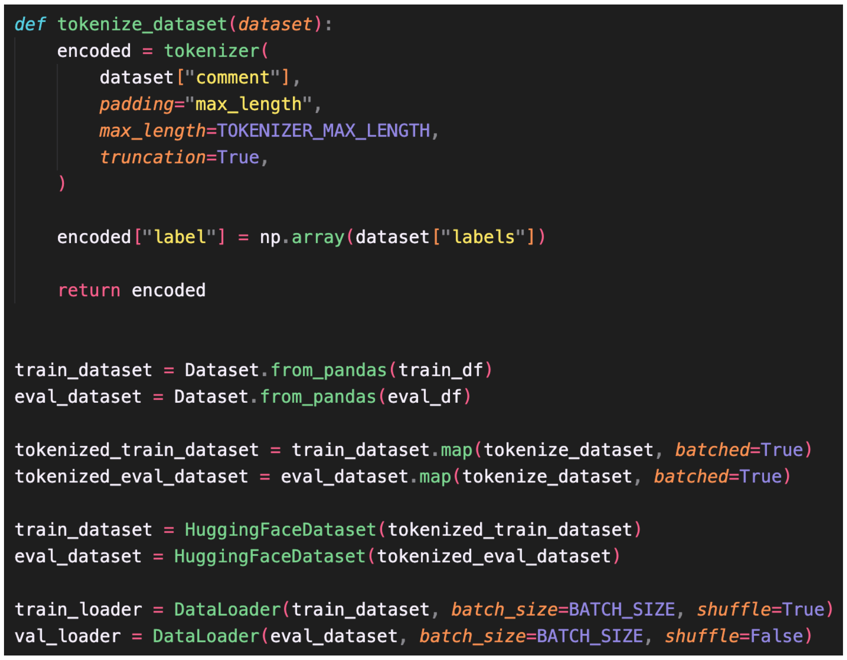


```python
class HuggingFaceDataset(torch.utils.data.Dataset):
    def __init__(self, hf_dataset):
        self.dataset = hf_dataset

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

    def __getitem__(self, idx):
        item = self.dataset[idx]
        input_ids = torch.tensor(item["input_ids"])
        attention_mask = torch.tensor(item["attention_mask"])
        if "label" in item:
            label = torch.tensor(item["label"], dtype=torch.float)
        return {
            "input_ids": input_ids,
            "attention_mask": attention_mask,
            "label": label,
        }
```

Fig. 9 : Custom PyTorch dataset class



```python
def tokenize_dataset(dataset):
    encoded = tokenizer(
        dataset["comment"],
        padding="max_length",
        max_length=TOKENIZER_MAX_LENGTH,
        truncation=True,
    )

    encoded["label"] = np.array(dataset["labels"])

    return encoded


train_dataset = Dataset.from_pandas(train_df)
eval_dataset = Dataset.from_pandas(eval_df)

tokenized_train_dataset = train_dataset.map(tokenize_dataset, batched=True)
tokenized_eval_dataset = eval_dataset.map(tokenize_dataset, batched=True)

train_dataset = HuggingFaceDataset(tokenized_train_dataset)
eval_dataset = HuggingFaceDataset(tokenized_eval_dataset)

train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
val_loader = DataLoader(eval_dataset, batch_size=BATCH_SIZE, shuffle=False)
```

Fig. 10 : Data preparation pipeline

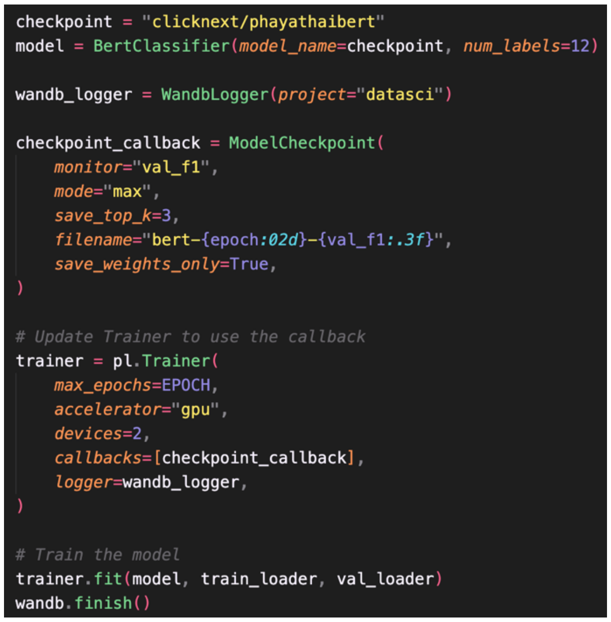
Model

I use pretrained PhayaThaiBERT, a pre-trained BERT model designed for Thai language processing. It is fine-tuned for a multi-label text classification task, The implementation leverages Hugging Face Transformers, PyTorch Lightning, and Weights & Biases (WanDB).

The model architecture is built using *AutoModelForSequenceClassification* with multi label classification problem type. The model's input consists of tokenized text representations using the *AutoTokenizer*, which applies padding and truncation.

The training process is handled using BertClassifier class. The training step computes loss using binary cross-entropy with logits, logs training loss using wandb.log(), and appends loss values for tracking. The validation step applies a sigmoid activation function and converts probability scores into binary predictions using a 0.5 threshold. Evaluation metrics such as accuracy and macro F1-score are computed to assess model performance on the validation set.

For optimization, the model uses the AdamW optimizer. The training is executed using PyTorch Lightning's Trainer, with GPU acceleration enabled. The training is performed for one epoch at a time, where after each epoch, validation metrics such as accuracy and macro F1-score are computed. The process also incorporates ModelCheckpoint to save the top-performing.



```python
checkpoint = "clicknext/phayathaibert"
model = BertClassifier(model_name=checkpoint, num_labels=12)

wandb_logger = WandbLogger(project="datasci")

checkpoint_callback = ModelCheckpoint(
    monitor="val_f1",
    mode="max",
    save_top_k=3,
    filename="bert-{epoch:02d}-{val_f1:.3f}",
    save_weights_only=True,
)

# Update Trainer to use the callback
trainer = pl.Trainer(
    max_epochs=EPOCH,
    accelerator="gpu",
    devices=2,
    callbacks=[checkpoint_callback],
    logger=wandb_logger,
)

# Train the model
trainer.fit(model, train_loader, val_loader)
wandb.finish()
```

Fig. 11 : Training process

Results

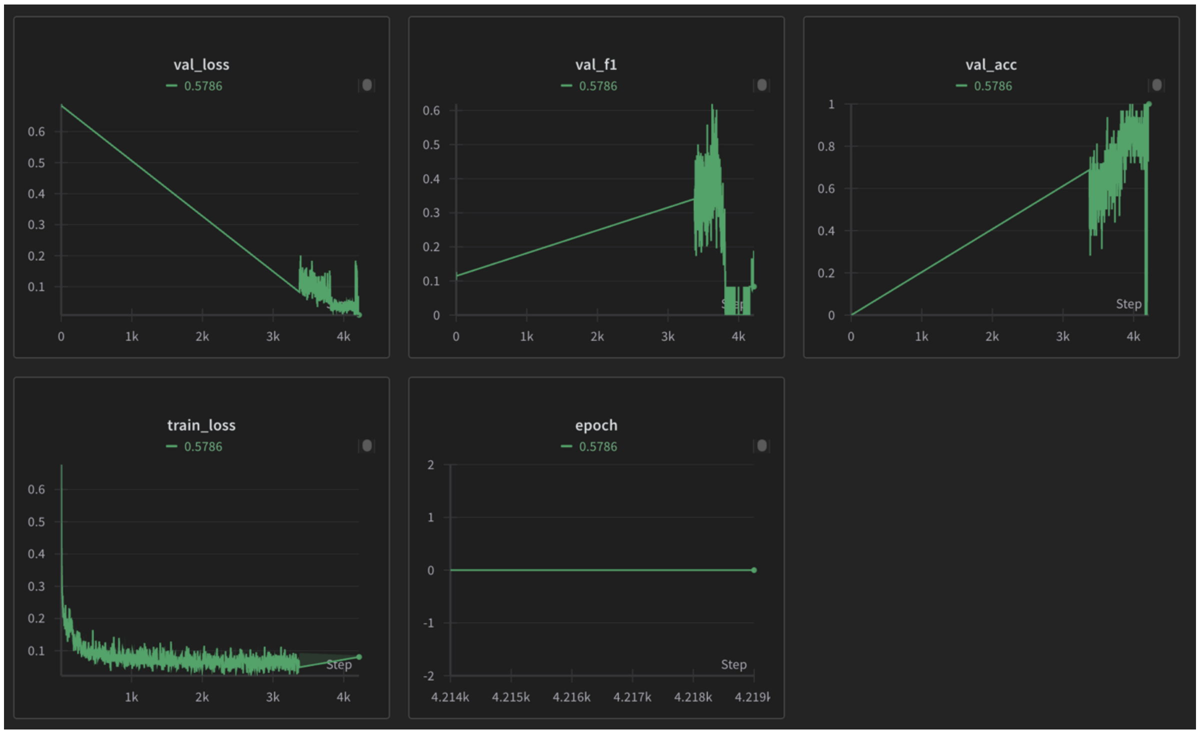


Fig. 12 : WanDB training graphs

The graph in Figure 12 indicates a decrease in validation loss as training progresses suggesting that the model is reducing its error on the validation dataset. The F1-score increases gradually over time, reaching its highest point near step 3,500–4,000 before fluctuating. The validation accuracy improves significantly throughout training, rising from near 0 at the start to a high value close to 1 at the final steps. The training loss graph shows a sharp decline in the early stages and then stabilizes at a low value

The graphs indicate a successful training process where training and validation loss decrease, while validation accuracy and F1-score increase. The sharp improvement in accuracy and F1-score suggests that the model effectively learned from the data. However, some fluctuations in validation F1-score near the end could suggest potential overfitting or variations in label distribution that may require further tuning.

After making predictions using a 0.4 prediction threshold on the test dataset, the distribution of predictions is visualized in Figures 13 and 14 below. The model predicts two additional label counts as expectations. However, the predictions for classes 3, 5, 8, and 11 remain relatively low. This model achieved a private F1 score of 0.4878 and a public F1 score of 0.5786.

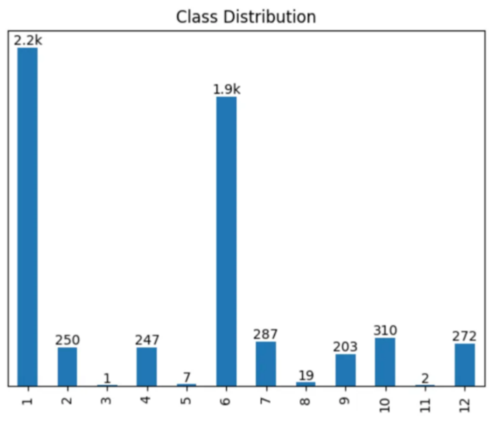


Fig. 13 : Predicted class distribution graph

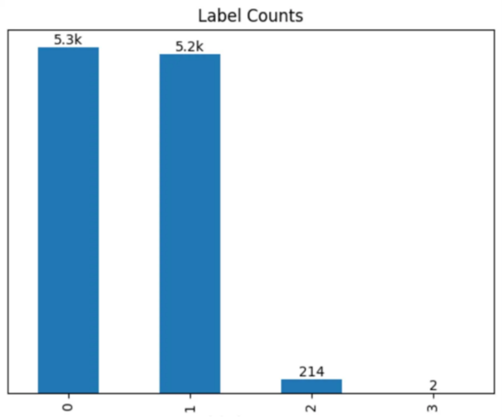


Fig. 14 : Predicted label counts graph

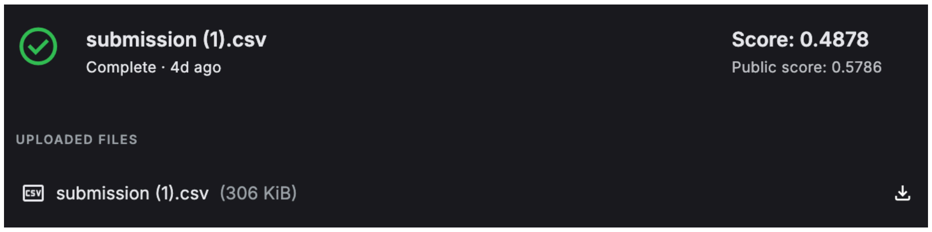

submission (1).csv
Complete · 4d ago
Score: 0.4878
Public score: 0.5786
UPLOADED FILES
submission (1).csv (306 KiB)

Fig. 15 : Kaggle submission scores

Discussion

Error analysis

After evaluating the model's predictions, several key observations highlight areas where the model underperforms and potential reasons behind these errors.

- Imbalanced label predictions

The model tends to predict two additional label counts more than expected. This suggests that the model may have a bias towards assigning multiple labels to comments, potentially due to the dataset's label distribution. Additionally, classes 3, 5, 8, and 11 have significantly fewer predictions, indicating that the model struggles to learn patterns associated with these specific classes. This could be due to an imbalance in training data, where these classes still have fewer samples compared to others.

- Low performance in certain classes

From the F1-score evaluation, it is evident that some classes perform worse than others. The lower predictions in classes 3, 5, 8, and 11 suggest that the model might not have enough representative examples to generalize well. This leads to poor recall, where the model fails to detect instances belonging to these categories.

- Overconfidence in predictions

The model sometimes over-predicts certain labels, meaning that it assigns labels too frequently even when they are incorrect. This overconfidence can stem from an issue with threshold selection (e.g., the 0.5 threshold used for classification might not be optimal for all labels). Tuning this threshold based on precision-recall trade-offs for each label may help improve accuracy.

- Potential model overfitting

While the validation accuracy and F1-score improved, fluctuations in the validation loss and F1-score graphs suggest potential overfitting. The model might be memorizing patterns from the training data instead of generalizing effectively. Regularization techniques such as dropout, weight decay, or additional data augmentation could improve robustness.

Further improvements

- Handling Imbalanced Classes

  Using class weighting, oversampling underrepresented classes, or synthetic data generation to improve performance for rare labels.

- Data Augmentation

  Expanding the dataset by including more labeled data or generating synthetic text samples for underrepresented labels.

- Hyperparameter Tuning

  Adjusting learning rates, batch sizes, and optimizer parameters to find optimal settings for better generalization.

- Text Preprocessing Enhancements

  Handling misspellings and typos using text normalization techniques.

Conclusion

This study focused on fine-tuning PhayaThaiBERT, a pre-trained Thai language model, for a multi-label text classification task. The model achieved a private F1 score of 0.4878 and a public F1 score of 0.5786 in test data prediction, demonstrating moderate classification performance. However, error analysis revealed class imbalance issues, inconsistent label predictions, and potential overfitting. To further improve model performance, several strategies can be implemented, including handling data imbalance, applying data augmentation, optimizing hyperparameters, and refining text preprocessing techniques.

To be concluded, while the model demonstrates promising results, there is still opportunity for improvement. By addressing the identified challenges and refining the training approach, the model can achieve higher accuracy, better generalization, and improved classification performance in real-world applications.