Term Project: DADS7203 Semester 1 2567

Team Member:

1. Siriwan Sreebutkot 6520422019

2. Chalita Iamleelaporn 6610412002

3. Ranakorn Boonsuankergchai 6610412003

Topic: ThaiSMS-ScamGuard: Deep Learning-based Thai SMS Scam Detection using OpenThaiGPT

Objective: This project presents ThaiSMS-ScamGuard, a deep learning-based system for detecting SMS scams in the Thai language using OpenThaiGPT, a variant of GPT tailored for Thai text. The objective of this work is to classify SMS messages into two categories: "Scam" and "Non-Scam". To achieve this, we employ a transfer learning approach by fine-tuning a pre-trained OpenThaiGPT model. The model is enhanced with a classification head and trained using a dataset containing Thai SMS messages labeled as either 'Scam' or 'Non-scam'. The system utilizes a custom-built neural network architecture that incorporates tokenization, attention mechanisms, and a classification layer to predict SMS authenticity.

Dataset:

The dataset used in this project was compiled from two primary sources:

1. **Actual SMS messages** that team members have received.
2. **Publicly available datasets** identified through Google searches.

The SMS messages were labeled into two categories: *"Scam"* and *"Non-Scam"*. The distribution of data is summarized as follows:

Class	**Count**
1 : Scam	306
0: Non-Scam	309
Total	615

Proposed Method: (Solution, Architecture, Structure of code)

1. **Data Splitting**
   In this project, the data splitting is performed using stratified train-test-validation split, which is crucial for maintaining the class distribution across different datasets. The data was split into: Training Set 70%, Validation Set 10%, and Test Set 20%. Stratification strategy was applied (*stratify=df['label']) to* ensure that the class distribution is maintained across all splits, particularly important for binary classification of "Scam" vs "Non-Scam" SMS.

2. Model architecture

This research employed pre-trained OpenThaiGPT 1.5 7b as a feature extractor because outperforming Thai Large Language Model.OpenThaiGPT 1.5 is a sophisticated Thai language conversational model predicated on Qwen v2.5, finetuned on over 2,000,000 Thai instructional pairs. Benchmark results substantiate OpenThaiGPT 1.5's cutting-edge performance across various Thai language tasks, surpassing other open-source Thai language models. A transfer learning approach was utilized to develop our models. We adapt it for a specific downstream task, SMS scam detection, while leveraging the knowledge it gained during pre-training. For training, an entire pre-trained OpenThaiGPT 1.5 7b was used as a feature extractor, and a custom classifier head (fully connected layer) was added on top for the SMS classification task. To train the model, all weights in the feature extractor remained frozen (makes them non-trainable), and only the new classifier head was trained. The transfer learning technique is illustrated in Figure 1.

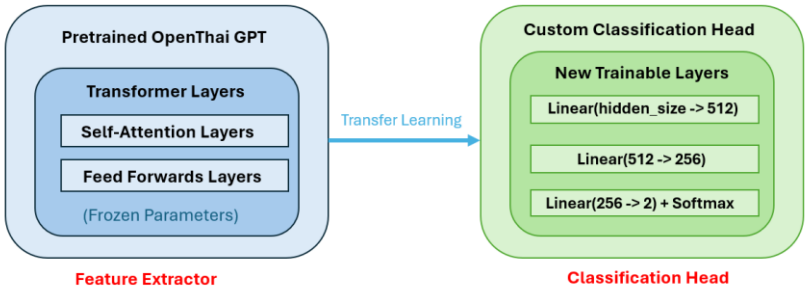

Figure 1: Transfer learning technique.

2.1 Feature Extractor

The key feature extraction happens in the forward method. The full OpenThaiGPT model extracts feature from the last hidden state layer because it contains most processed information and combines features from all previous layers. [CLS] token representation was got from the last hidden state as the final feature vector to aggregate information from the entire sequence**.** The detail of forward method in structure of code as the following step by step:

```
def forward(self, input_ids, attention_mask):
    # 1. Pass through GPT model
    outputs = self.gpt(input_ids=input_ids,
                       attention_mask=attention_mask,
                       output_hidden_states=True)

    # 2. Get last hidden layer output
    last_hidden_state = outputs.hidden_states[-1]

    # 3. Extract CLS token representation
    cls_hidden_state = last_hidden_state[:, 0, :]

    # 4. Classification
    logits = self.classifier(cls_hidden_state)
    return logits # Return class scores
```

Figure 2: Forward pass code

1. Forward pass through GPT model to get hidden states from all layers. *output_hidden_states=True* to return hidden states from all layers.
2. Extracting last hidden state.
3. Getting [CLS] token representation.
4. Pass through the classification layers

2.2 Classification Head

The custom classifier head included fully connected layers containing 512 and 256 nodes, respectively. A single final layer used a Softmax function with 2 nodes to predict two classes: “Scam” and “Non-Scam”. The activation function: ReLu was applied in each fully connected layer to introduce non-linearity. The ReLu was selected due to its computational efficiency.

The classification section takes the [CLS] token representation, then, transforms it into class probabilities which matches number of 2 classes. The classification is illustrated in Figure 3.

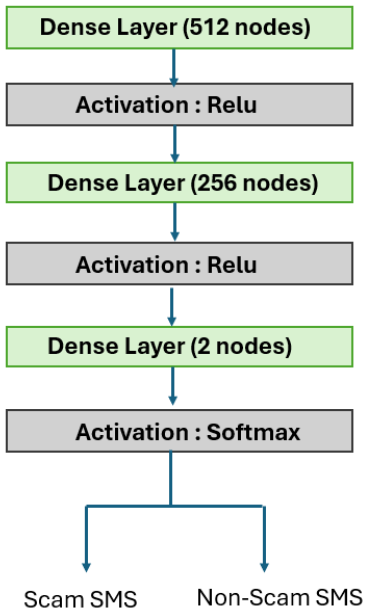

Figure 3: Custom classification Head

3. Tokenization

The tokenizer is created using the OpenThaiGPT model's pre-trained tokenizer in

tokenizer = AutoTokenizer.from_pretrained("openthaigpt/openthaigpt1.5-7b-instruct")

The tokenizer converts Thai text to token IDs, handling padding, truncation, and creating attention masks automatically. Tokenization Parameters as the following:

- *truncation=True*: Truncates texts longer than *max_length* to limit the length of a sequence for neural network processing.
- *padding='max_length'*: Pads shorter texts to *max_length*.For sequences shorter than 512 tokens, special padding tokens are added to the end of the sequence as typically set to zero or a special 'pad' token. This process ensures all sequences reach the maximum length (512 tokens).

4. Model Training

For training each model, binary cross-entropy loss was used to calculate the errors between the actual and predicted labels due to the binary classification task. Adaptive Moment Estimation (Adam) was chosen as the optimizer. A model was defined to train with 10 epochs, batch size = 16, and learning rate 0.001.

4.1 Early Stop Strategy

The Early Stopping strategy was applied to prevent overfitting during model training. This helps ensure the model generalizes well to unseen data without unnecessarily continuing training when performance plateaus. The strategy monitors the validation loss during training. If the validation loss does not improve (within a small delta) for a specified number of epochs (patience), training stops. Patience is set to 3 epochs. The minimum delta is 0 (any improvement counts). The model was trained only 5 epoch due to early stop implementation.

5. Evaluation

The model's performance was assessed on the test dataset with various metrics: Accuracy, Precision, Recall, F1-score, and Confusion Matrix to breakdown the correct and incorrect predictions.

6. Result

The classification report shows the result as Table 1.

Table1 : Evaluation Result

Metrics	Result
Accuracy	0.72
Precision	0.72
Recall	0.72
F1-Score	0.72

The overall accuracy of the model on the test set is 72%, which is a decent result but indicates there is still room for improvement. The precision and recall are also around 72%, suggesting the model is able to identify both “scam” and “non-scam” messages with similar levels of accuracy.

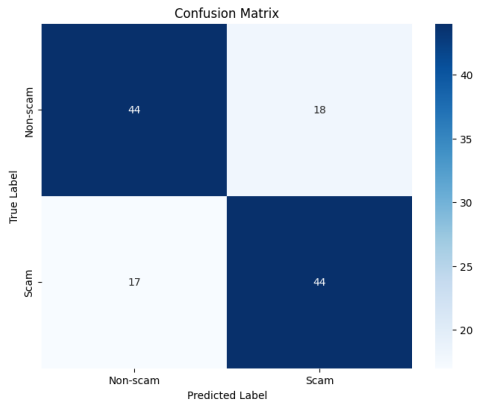

Figure 4: Confusion Matrix

The confusion matrix indicates that the model has a tendency to be more False Positive in its predictions, tending to classify messages as "Scam" more often than they actually are. This could be due to the model being overly sensitive to certain linguistic patterns or features that are indicative of scam messages.

7. Inference:

The model inference provides outputs probabilities for Scam (1) and Non-Scam (0) classes by prediction process as the following:

- Tokenizes input text
- Passes through the pre-trained GPT model
- Uses the classification head to predict the class
- Returns class and prediction probability

To implement the model for classifying unseen Thai SMS, the model and tokenizer were saved to a directory, enabling easy model reloading and deployment. The following example illustrates the model's accurate prediction of an unseen scam SMS.

```
Single Text Prediction:
Text: ยินดีด้วยคุณได้รับรางวัลจากธนาคาร กรุณาโอนค่าธรรมเนียม 1,000 บาท
Prediction: Scam
Confidence: 94.22%
Class Probabilities: {'Non-scam': 0.057820286601781845, 'Scam': 0.9421797394752502}
```

Figure 5: Example of model inference

8. Structure of code

The structure of code was divided as the following:

1. Read CSV files : The function defines a list of common Thai text encodings:
    - tis-620: Traditional Thai encoding
    - cp874: Windows Thai encoding
    - utf-8-sig: UTF-8 with signature
    - utf-16: Unicode 16-bit encoding
2. Construct the model
3. Tokenization process: This section is crucial for preparing the Thai SMS dataset for the deep learning model, ensuring consistent and properly formatted input for the OpenThaiGPT-based classifier.
    - Raw SMS texts are passed to the class
    - Tokenizer converts texts to numerical representations
    - Texts are padded/truncated to a fixed length
    - Each text gets an attention mask
    - Labels are converted to tensors
    - Can be easily used with DataLoader for batch processing during training
4. Early Stopping
5. Plot Learning Curve: This function is designed to visualize the model's learning progress by plotting training and validation losses and accuracies across epochs, which is crucial for understanding the model's performance and training dynamics.
6. Train Model
7. Evaluation
8. Save Model
9. Inference : This part includes the functions to load saved mode and tokenizer and predict on unseen SMS.

Data Source:

The data used for training the model was collected from students at Varee Chiangmai School who worked on a project developing a mobile application to filter Scam messages.

สแปมจ๋า ลาก่อนจ้าเมื่อมาเจอ Chatbot สุดล้ำ EP2 : การเทรนนิ่งโมเดล NLP จำแนกข้อความ spam | by Sorapong Somsorn | Medium