AIFFEL DL-thon DKTC Task Non-Submission

Anonymous authors
Paper under double-blind review

Abstract

The DKTC (Dataset of Korean Threatening Conversations) task of AIFFEL-thon involved predicting one of the four sub-categories of threats based on arbitrary conversational audio files. Participants were required to build a model for predicting and classifying the test data in any way they wished, using the given train and test data. The accuracy of the results could be verified by comparing them with a separate answer table. Our team chose to use an ensemble approach, combining models that showed high accuracy using pre-trained models like KLEU BERT, Ko-ELECTRA, and Funnel Transformer.(1)

1 Introduction

The DKTC dataset was created by TUNiB for participation in the 2021 AI Grand Challenge 4th Competition, Speech Recognition Track. The challenge involved predicting one of the four sub-categories of threats or general conversations based on arbitrary conversational audio files. Since the organizers did not provide any training data apart from the samples, participating teams had to create their own data. Therefore, TUNiB produced this dataset through crowd sourcing and released it for non-commercial uses.(1)

The DKTC dataset was divided into training and test data. The training data consisted of approximately 1,000 conversations in each of the four threat sub-categories: 'Threats', 'Extortion', 'Workplace Harassment', and 'Other Harassment'. The test data consisted of 100 conversations in each of the five categories: 'Threats', 'Extortion', 'Workplace Harassment', 'Other Harassment', and 'General Conversation'.

The task of this AIFFEL-thon, conducted by AIFFEL, was to classify test data from the DKTC dataset into one of the four threat sub-categories, excluding the 'General Conversation' class. Participants were given train and test data to devise solutions for the task, and they could check the accuracy of their solutions on a separate accuracy verification website.

To solve the task, our team first pre-processed the given dataset to make it usable in the task. We then tested the preprocessed dataset with several pre-trained models known for showing good performance in determining their respective accuracies. Based on this, we selected the model that best fit the dataset, identified the pre-processing elements that yielded the highest scores, and stored those conditions. We then used ensemble methods to combine the conditions from each model to achieve even higher accuracy.

2 Background

2.1 Ko-BERT (Korean Bidirectional Encoder Representations from Transformers)

Ko-BERT was released by SKT (SK Telecom) and trained with 50 million sentences collected from sources such as Wikipedia and news articles. To account for the irregular language variations present in Korean, a data-driven tokenization technique, the SentencePiece tokenizer, was applied. The vocabulary size is 8,002, and the model has 92 million parameters.(2)

2.2 Ko-ELECTRA (Korean Efficiently Learning an Encoder that Classifies Token Replacements Accurately)

Ko-ELECTRA is based on ELECTRA which utilizes Replaced Token Detection for training, where the discriminator determines whether a token coming from the generator is a "real" or "fake" token. This method has the advantage of being able to train on all input tokens and has shown better performance compared to models like BERT. KoELECTRA was trained with 34GB of Korean text.(3)

2.3 KLEU (Korean Language Understanding Evaluation) BERT

KLUE-BERT was a model used as a baseline in the KLUE benchmark dataset. It was trained on 63GB of data extracted from various sources such as Modu Corpus, CC-100-Kor, Namu Wiki, news, and petitions. A Morpheme-based Subword Tokenizer was used for tokenization. The vocabulary size is 32,000, and the model has 111 million parameters.(4)

2.4 Funnel Transformer

Funnel-transformer was designed to improve the efficiency of language models, by addressing redundancy in maintaining full-length token-level representations, particularly for tasks that only need a single-vector summary of the input sequence. The architecture gradually compresses the sequence of hidden states to a shorter one, reducing computational cost.(5)

2.5 Ensemble

Ensemble method combines multiple models to improve overall performance. Instead of relying on a single model to make predictions or decisions, an ensemble uses multiple models and aggregates their outputs. The most common techniques for ensemble methods include Bagging, Boosting, and Stacking. A simple Bagging by averaging the predictions of different models was used in our project.(6)

3 Method

3.1 Pre-processing

3.1.1 Cleaning Dataset

Special characters and numbers were removed from dataset, and stopwords recommended by ChatGPT and selected from datasets were removed.

```
def clean_text(text):
    # Convert to lowercase
    text = text.lower()

    # Remove special characters and numbers
    text = re.sub(r'[^   -  a  -z\s]', ' ', text)

    # Remove extra spaces
    text = ' '.join(text.split())

    # korean stopwords list
    stopwords = [
        '   ', '   ', '   ', '   ', '   ', '   ', '   ', '   ', '   ', '   ', '   ', '   ',
    '   ', '   ', '   ', '   ',
        '   ', '   ', '   ', '   ', '   ', '   ', '   ', '   ', '   ', '   ', '   ', '
      ', '   ', '   ',
        '   ', '   ', '   ', '   ', '   ', '   ', '   ', '   ', '   ', '   ', '
          ', '   ', '   ', '   ', '   ',
        '   ', '   ', '   ', '   ', '   ', '   ', '   ', '   ', '   ', '
          ', '   ', '   ', '   ',
        '   ', '   ', '   ', '   ', '   ', '   ', '   ', '   ', '   ', '   ', '   ', '
      ', '   ', '   ', '   ', '   ',
        '   ', '      ', '      ', '      ', '      ', '      ', '   ', '      ', '   ', '
       ', '      ', '      ', '      ', '   ',
        '   ', '   ', '      ', '      ', '   ', '   ', '   ', '      ', '   ', '   ', '
    ', '      ', '      ', '   '
    ]

    # removing stopwords
    text = ' '.join(word for word in text.split() if word not in stopwords)

    return text
```

3.1.2 Normalizing Dataset

Both train and test dataset were normalized prior to training.

```
# load dataset
train = pd.read_csv('./data/train.csv')
test = pd.read_json('./data/test.json').transpose()

# normalize rows of texts for train and test dataset
train['conversation'] = train['conversation'].apply(clean_text)
test['text'] = test['text'].apply(clean_text)

# encode set classes as numbers
label_dict = {
    '             ': 0,
    '             ': 1,
    '                           ': 2,
    '                       ': 3
}
train['label_encoded'] = train['class'].map(label_dict)
```

3.2 Model training

3.2.1 Defining Dataset

Tokenizer and dataset to be used for training were configured for each model.

```
# Tokenizer configuration
from transformers import BertTokenizer
tokenizer = BertTokenizer.from_pretrained('monologg/kobert')

# defining dataset
class BERTDataset(Dataset):
    def __init__(self, dataframe, tokenizer, max_len):
        self.tokenizer = tokenizer
        self.data = dataframe
        self.text = dataframe['conversation'].tolist()
        self.labels = dataframe['label_encoded'].tolist()
        self.max_len = max_len

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

    def __getitem__(self, index):
        text = str(self.text[index])
        text = " ".join(text.split())
        inputs = self.tokenizer.encode_plus(
            text,
            None,
            add_special_tokens=True,
            max_length=self.max_len,
            pad_to_max_length=True,
            return_token_type_ids=True
        )
        ids = inputs['input_ids']
        mask = inputs['attention_mask']

        return {
            'ids': torch.tensor(ids, dtype=torch.long),
            'mask': torch.tensor(mask, dtype=torch.long),
            'targets': torch.tensor(self.labels[index], dtype=torch.long)
        }
```

3.2.2 Configuring DataLoader

Length of token and batch size was configured. For Length of token, through experimentation, optimal length at which the accuracy was highest was found-approximately 300-.

```
MAX_LEN = 512
TRAIN_BATCH_SIZE = 8
VALID_BATCH_SIZE = 8
train_dataset = BERTDataset(train, tokenizer, MAX_LEN)
train_data_loader = DataLoader(train_dataset, batch_size=TRAIN_BATCH_SIZE)
```

3.2.3 Configuring Training Steps for Train Data

The pre-trained models to be used, their loss function, optimizer, and training function were defined.

```
# defining model
class BERTClass(torch.nn.Module):
    def __init__(self):
        super(BERTClass, self).__init__()
        self.l1 = BertForSequenceClassification.from_pretrained('monologg/kobert',
    num_labels=4)

    def forward(self, ids, mask):
        output = self.l1(ids, attention_mask=mask)
        return output.logits

model = BERTClass()
model.to(torch.device("cuda"))

# configuring loss function and optimizer
loss_function = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(params=model.parameters(), lr=1e-5)

# defining training function
def train(epoch):
    model.train()
    for _, data in tqdm(enumerate(train_data_loader), total=len(train_data_loader)):
        ids = data['ids'].to(torch.device("cuda"), dtype=torch.long)
        mask = data['mask'].to(torch.device("cuda"), dtype=torch.long)
        targets = data['targets'].to(torch.device("cuda"), dtype=torch.long)

        outputs = model(ids, mask)
        optimizer.zero_grad()
        loss = loss_function(outputs, targets)
        loss.backward()
        optimizer.step()

# training
EPOCHS = 3
for epoch in range(EPOCHS):
    train(epoch)
```

3.2.4 Configuring Training Steps for Test Data

The fine-tuned model was used to return predictions using test data.

```
# DataLoader configuration for test data
```

```
class BERTTestDataset(Dataset):
    def __init__(self, dataframe, tokenizer, max_len):
        self.tokenizer = tokenizer
        self.data = dataframe
        self.text = dataframe['text'].tolist()
        self.max_len = max_len

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

    def __getitem__(self, index):
        text = str(self.text[index])
        text = " ".join(text.split())
        inputs = self.tokenizer.encode_plus(
            text,
            None,
            add_special_tokens=True,
            max_length=self.max_len,
            pad_to_max_length=True,
            return_token_type_ids=True
        )
        ids = inputs['input_ids']
        mask = inputs['attention_mask']

        return {
            'ids': torch.tensor(ids, dtype=torch.long),
            'mask': torch.tensor(mask, dtype=torch.long)
        }

test_dataset = BERTTestDataset(test, tokenizer, MAX_LEN)
test_data_loader = DataLoader(test_dataset, batch_size=VALID_BATCH_SIZE)

# defining prediction function
def predict():
    model.eval()
    predictions = []
    with torch.no_grad():
        for _, data in tqdm(enumerate(test_data_loader), total=len(test_data_loader)):
            ids = data['ids'].to(torch.device("cuda"), dtype=torch.long)
            mask = data['mask'].to(torch.device("cuda"), dtype=torch.long)

            outputs = model(ids, mask)
            _, predicted = torch.max(outputs, 1)

            predictions.extend(predicted.cpu().numpy().tolist())
    return predictions

# prediction
predictions = predict()

submission = pd.DataFrame({'file_name': test.index, 'class': predictions})
submission.to_csv('./final_submission.csv', index=False)
```

3.2.5 Saving and Loading Model

Fine-tuned model which returned acceptable results were saved for use in ensemble procedure.

3.3 Ensemble

Predictions of different fine-tuned models were averaged. The average of predictions were used to deduce the conversation class of the test dataset.

```
# prediction made with each model
kobert_probs = predict_proba(kobert_model, kobert_test_data_loader)
koelectra_probs = predict_proba(koelectra_model, koelectra_test_data_loader)

```

```
# average the predictions and deduce conversation class for final result
final_predictions = []
for k_bert_prob, k_electra_prob in zip(kobert_probs, koelectra_probs):
    avg_prob = [(a+b)/2 for a, b in zip(k_bert_prob, k_electra_prob)]
    final_predictions.append(np.argmax(avg_prob))

submission = pd.DataFrame({'file_name': test.index, 'class': final_predictions})
submission.to_csv('./ensemble_submission.csv', index=False)
```

4 Result

[HTML]FFFFFF**DataSet**	[HTML]FFFFFF**Batch**	[HTML]FFFFFF**Epoch**	[HTML]FFFFFF**Score**	[HTML]FFFF
kokoensemble	256	4	2	0.875
kokoensemble	256	4	3	0.8925
kokoensemble	256	4	4	0.88
kokoensemble	512	4	3	0.8675
kokoensemble	512	4	4	0.885
kokoensemble	512	4	5	0.8825
kokoensemble	256	16	3	0.8775
kokoensemble	256	4	3	0.575
kokoensemble	256	4	3	0.8575
kokoensemble	288	4	1	0.8425
kokoensemble	288	4	2	0.885
kokoensemble	288	4	3	0.87
kokoensemble	288	4	4	0.895
kokoensemble	288	4	5	0.89
kokoensemble	288	4	6	0.8975
kokoensemble	320	4	6	0.875
kokoensemble	350	4	1	0.8375
kokoensemble	350	4	2	0.88
kokoensemble	350	4	3	0.8925
kokoensemble	350	4	4	0.9025
kokoensemble	350	4	5	0.9
kokoensemble	256	8	3	0.88
kokoensemble	350	4	2	0.8725
kokoensemble	350	4	2	0.8675
kokoensemble	350	4	5	0.8675
kokoensemble	350	4	1	0.8725
kokoensemble	350	4	2	0.9025
kokoensemble	350	4	3	0.8925
kokoensemble	350	4	4	0.8825
Ko-ELECTRA	350	4	3	0.905
Ko-ELECTRA	350	4	4	0.91
Ko-ELECTRA	350	4	5	0.895
Ko-ELECTRA	350	4	2	0.865
Ko-ELECTRA	350	4	5	0.895
Ko-ELECTRA	350	4	1	0.835
Ko-ELECTRA	350	4	2	0.8825
Ko-ELECTRA	350	4	3	0.87
Ko-ELECTRA	350	4	4	0.8725
Ko-ELECTRA	350	4	4	0.89
Ko-ELECTRA	350	4	4	0.8825
Ko-ELECTRA	350	4	4	0.8925
Ko-ELECTRA	350	4	4	0.9025

Ko-ELECTRA	350	4	4	0.8675
Ko-ELECTRA	350	4	4	0.9125
Ko-ELECTRA	350	4(2)	4	0.9
Ko-ELECTRA	350	4	4	0.8875
Ko-ELECTRA	350	2	4	0.8925
Ko-ELECTRA	350	4	4	0.9
Ko-ELECTRA	350	4	4	0.86
Ko-ELECTRA	350	4	4	0.9125
Ko-ELECTRA	350	4	4	0.9025
Ko-ELECTRA	350	4	4	0.8975
Ko-ELECTRA	350	4	4	0.8975
Ko-ELECTRA	350	4	4	0.9
Ko-ELECTRA	350	4	4	0.9025
Ko-ELECTRA	350	4	4	0.905
Ko-ELECTRA	350	4	4	0.905
Ko-ELECTRA	350	4	4	0.9
Ko-ELECTRA	350	4	4	0.9
Ko-ELECTRA	350	4	4	0.9
Ko-ELECTRA	350	4	4	0.8975
Ko-ELECTRA	350	4	4	0.8975
Ko-ELECTRA	350	4	4	0.905
Ko-ELECTRA	350	4	4	0.8925
Ko-ELECTRA	350	4	4	0.9025
Ko-ELECTRA	350	4	4	0.8975
Ko-ELECTRA	350	4	4	0.9
Ko-ELECTRA	350	4	4	0.9025
Ko-ELECTRA	350	4	4	0.9
Ko-ELECTRA	350	4	4	0.905
Ko-ELECTRA	350	4	4	0.9
Ko-ELECTRA	512	8	3	0.8875
Ko-ELECTRA	512	8	4	0.885
Ko-ELECTRA	512	8	2	0.855
Ko-ELECTRA	288	4	1	0.84
Ko-ELECTRA	288	4	2	0.87
Ko-ELECTRA	288	4	3	0.87
Ko-ELECTRA	288	4	4	0.8925
Ko-ELECTRA	288	4	5	0.8975
Ko-ELECTRA	288	4	6	0.8925
Ko-ELECTRA	320	4	6	0.875
Ko-BERT	512	16	3	0.6225
Ko-BERT	256	16	3	0.585
Ko-BERT	512	8	2	0.5625
Ko-BERT	512	8	4	0.5975
Ko-BERT	512	8	5	0.6175
Ko-BERT	128	4	3	0.6325
Ko-BERT	64	4	3	0.5825
Ko-BERT	350	4	3	0.63
Ko-BERT	350	4	4	0.625
Ko-BERT	350	4	5	0.6175
Ko-BERT	350	4	2	0.6175
Ko-BERT	350	4	5	0.605
Ko-BERT	512	8	1	0.3975
Ko-BERT	512	8	2	0.5375

Ko-BERT	512	8	3	0.6125
Ko-BERT	512	8	6	0.6625
Ko-BERT	512	8	10	0.5975
Ko-BERT	512	8	3	0.5775
Ko-BERT	512	8	6	0.625
Ko-BERT	128	16	3	0.555
Ko-BERT	256	16	3	0.59
Ko-BERT	288	4	1	0.535
Ko-BERT	288	4	2	0.5425
Ko-BERT	288	4	3	0.5525
Ko-BERT	288	4	4	0.615
Ko-BERT	288	4	5	0.615
Ko-BERT	288	4	6	0.625
Ko-BERT	320	4	6	0.555
klue	350	4	1	0.8825
klue	350	4	2	0.9
klue	350	4	3	0.8975
klue	350	4	4	0.8875
funnel-transformer-kor	350	4	1	0.875
funnel-transformer-kor	350	4	2	0.8875
funnel-transformer-kor	350	4	3	0.8825
funnel-transformer-kor	350	4	4	0.8725
funnel-transformer-kor	350	4	2	0.8925
funnel-transformer-kor	350	4	2	0.9075
Ensemble is all you need				0.92

Ensemble of Ko-ELECTRA, KLEU BERT, and Funnel-Transformer model ensembled with soft-voting resulted in accuracy of 0.92.

5 Discussion

Improving pre-processing procedures may yield better results. For this, augmenting data to increase the small size of dataset (4000 conversations) would be a possible option. Using different stopwords would be another possible approach to improving quality of train dataset to be used for fine-tuning. Making adjustments to optimizer could also affect the results, but with pre-trained models, this may not be optimal approach to take.

6 Conclusion

Our project which adopted the ensemble method with multiple high-performance pre-trained models, resulted in acceptable accuracy for classification task, with minimal pre-processing. Further pre-processing through data augmentation, stopwords modification, and optimizer adjustments, may result in improved outcome.

7 References

References

[1] `https://github.com/tunib-ai/DKTC/tree/main`. [Online; accessed 21-November-2023].

[2] `https://www.letr.ai/blog/tech-20221124`. [Online; accessed 21-November-2023].

[3] `https://github.com/monologg/KoELECTRA`. [Online; accessed 21-November-2023].

[4] `https://www.letr.ai/blog/tech-20221124`. [Online; accessed 21-November-2023].

[5] Zihang Dai, Guokun Lai, Yiming Yang, and Quoc Le. Funnel-transformer: Filtering out sequential redundancy for efficient language processing. *Advances in neural information processing systems*, 33:4271–4282, 2020.

[6] Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.