Sentiment Analysis in Korean Song Lyrics Data

Unstructured Data Analysis
Term Project
2021062742 Se Eun Lee

1. Objective

The main objective of this project is to analyze song lyrics data and extract emotions such as positive, negative, neutral, or more nuanced emotional categories. The project aims to apply various NLP techniques to identify common words, emotional patterns, and trends within song lyrics. Additionally, it seeks to compare and visualize emotional differences based on specific artists, genres, or time periods.

2. Methodology

The project methodology can be divided into several stages: data collection, preprocessing, model training, and evaluation.

2.1. Data Collection

The Korean song lyrics dataset was collected from multiple sources to ensure a diverse representation of emotions and styles:

- **KOTE Dataset**: The Korean Online That-gul Emotion (KOTE) dataset consists of 50,000 replies, each labeled with one of 44 emotion categories. This dataset provided a broad and nuanced set of emotional labels, useful for examining a wide range of possible sentiments in lyrics.
- **AI Hub Korean Corpus Data**: This dataset includes 51,574 sentences labeled with six distinct emotions (e.g., anxiety, anger, sadness, joy). The relatively smaller number of categories made this dataset more suitable for a focused analysis.
- **Crawled Melon Top-100 Lyrics**: The Top-100 chart songs were gathered from Melon, Korea's leading music streaming service. This dataset includes lyrics from current popular songs as of November 15th, 2024. The goal was to see how well models trained on conversational data could generalize to artistic text like lyrics.

Each dataset offered unique challenges and opportunities for model training and testing. The KOTE dataset's many categories made fine-tuning difficult, while the AI Hub data was simpler but lacked the depth needed to capture the complex emotions often found in music.

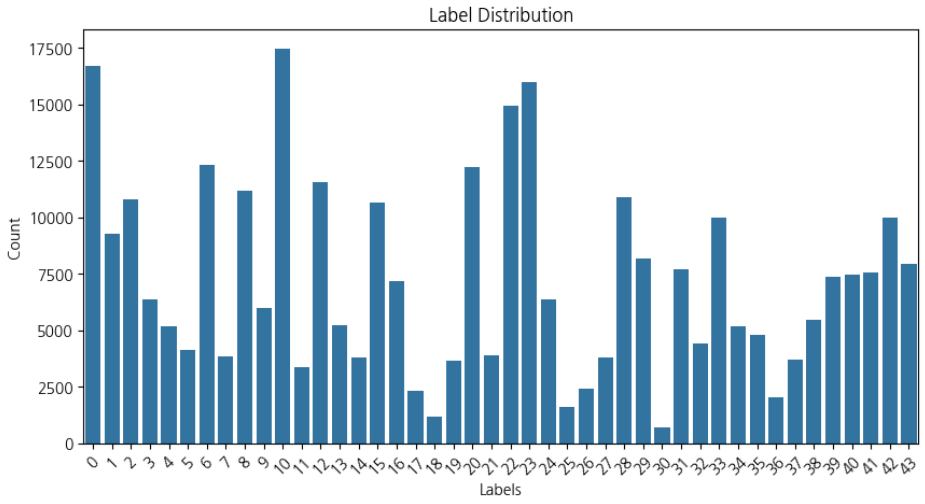
2.2. Data Preprocessing

- **Cleaning and Normalization**: Cleaning lyrics to remove unwanted characters such as punctuation marks and special symbols using regular expressions. This step helped reduce noise in the text.
- **Tokenization**: Lyrics were tokenized into smaller, more manageable units. For the KoBERT model, the `KoBERTTokenizer` was used to ensure compatibility with the pre-trained model's input requirements.
- **Sentence Splitting**: Long song lyrics were split into sentences to ensure that the model could effectively process each unit, as BERT-based models perform better with shorter, context-rich segments.

Handling artistic expressions and metaphors was particularly challenging, as the meaning is often context-dependent.

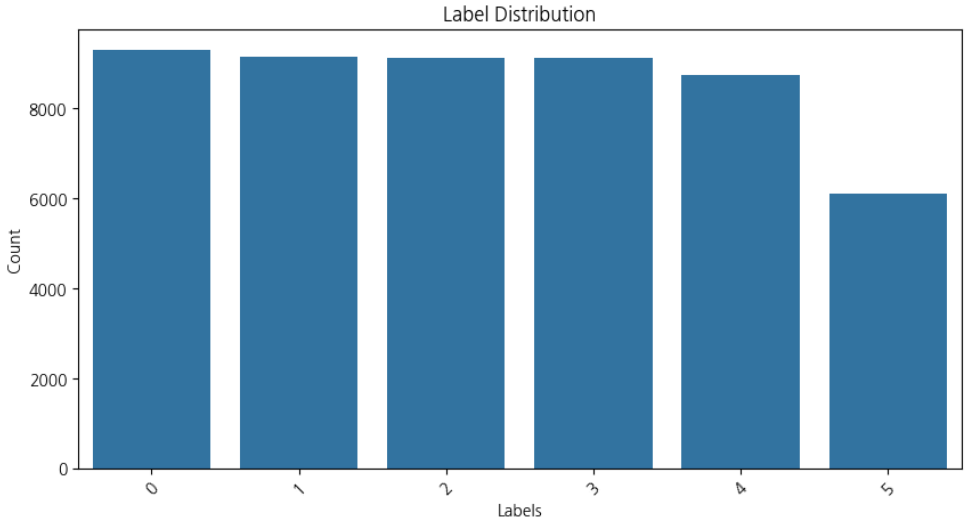
2.3. EDA

- **KOTE Dataset**: The dataset was split into 40,000 training examples, 5,000 validation examples, and 5,000 test examples. It was observed that the emotion categories were highly imbalanced, with some emotions having significantly fewer examples.



=> 0: "Complaint/Dissatisfaction", 1: "Welcome/Favor", 2: "Impression/Amazement", 3: "Sick of/Tired of", 4: "Gratitude", 5: "Sadness", 6: "Anger/Fury", 7: "Respect", 8: "Expectation", 9: "Arrogance/Contempt", 10: "Pity/Disappointment", 11: "Determination/Solemnity", 12: "Suspicion/Distrust", 13: "Pride", 14: "Comfort/Pleasantness", 15: "Wonder/Interest", 16: "Caring", 17: "Embarrassment", 18: "Fear/Terror", 19: "Despair", 20: "Patheticness", 21: "Disgust/Repulsion", 22: "Annoyance", 23: "Absurdity", 24: "None", 25: "Defeat/Self-Loathing", 26: "Annoyed/Bothered", 27: "Exhaustion/Weary", 28: "Joy/Excitement", 29: "Realization", 30: "Guilt", 31: "Hatred/Loathing", 32: "Fondness (Cuteness/Beauty)", 33: "Embarrassment/Awkwardness", 34: "Shock", 35: "Burden/Reluctance", 36: "Sadness/Sorrow", 37: "Boredom", 38: "Pity/Compassion", 39: "Surprise", 40: "Happiness", 41: "Anxiety/Worry", 42: "Delight", 43: "Relief/Trust"

- **AI Hub Dataset**: The AI Hub dataset was relatively balanced among the six emotion categories. However, the nature of the data, being more conversational, presented differences in language style compared to song lyrics.



=> 'Anxiety': 0 / 'Anger': 1 / 'Hurt': 2 / 'Sadness': 3, / 'Embarrassment': 4 / 'Joy': 5

3. Model Architecture

The model used for this project was based on the BERT architecture, with adaptations to work specifically with Korean text through KoBERT:

- **Model Type**: Transformer-based architecture (BERT)
- **Pre-trained Model**: `monologg/kobert`
- **Label Configuration**: The model was fine-tuned for both 44-label (KOTE) and 6-label (AI Hub) configurations.
- **Model Structure**: The transformer encoder layers were used to extract contextual information from the lyrics. A fully connected output layer was added for classification into sentiment categories.

The use of KoBERT allowed for deeper contextual understanding of the Korean language, although the large number of classes in the KOTE dataset presented significant challenges.

4. Model Works

4.1. Training(Fine-tuning)

The core of this project was training a KoBERT-based transformer model to classify emotions in song lyrics. KoBERT is a variant of BERT, pre-trained specifically on a large Korean corpus, making it particularly suited for this task.

- **Model Selection**: KoBERT (`monologg/kobert`) was selected for its pre-training on Korean text, which helped in understanding the nuances of the language.
- **Training Setup**: The dataset was divided into training, validation, and test sets to ensure robust evaluation of model performance. A learning rate with warm-up steps and weight decay was used to improve the optimization process. The training was performed on Google Colab, using a GPU if available.
- **Training Challenges**: Training with the KOTE dataset was particularly challenging due to the large number of labels (44), which resulted in longer training times and a model that struggled to generalize well on nuanced emotions. The training was eventually stopped after 2.6 epochs due to time constraints.
- **Hyperparameters**:
    - Batch size: 16
    - Number of epochs: 3
    - Learning rate: Adjusted with warm-up steps for stability

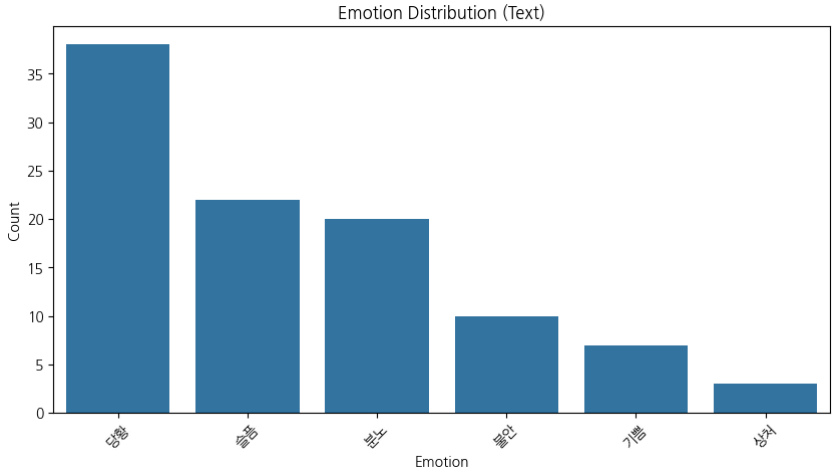
4.2. Evaluation

- **Accuracy**: The overall accuracy of the model, which provided a general sense of its ability to correctly classify emotions.

The test accuracy with the AI Hub dataset was approximately 0.5618, which was satisfactory given the complexity of the task and the imbalanced nature of the data.

5. Insights

5.1. Sentiment Distribution



=> Embarrassment 38 (includes NaN values) / Sadness 22 / Anger 20 /

Anxiety 10 / Joy 7 / Hurt 3

- **General Trends**: Most lyrics were classified as embarrassment or sadness, with fewer instances of strongly positive sentiment. But, the model classified nan values as embarrassment as well, this is the blindspot of this project.
- **Imbalanced Sentiment Categories**: The KOTE dataset's many categories made it difficult for the model to generalize well, particularly for underrepresented emotions. The AI Hub dataset provided a more balanced view, although it lacked the complexity needed for artistic expressions.

5.2. Model Performance

- **Effect of Data Imbalance**: Underrepresented classes were often misclassified, highlighting the need for a more balanced dataset or additional data augmentation techniques.

5.3. Lyrics Characteristics

- **Themes in Lyrics**: The frequent usage of words related to love, separation, and hope was identified. These recurring themes are indicative of Korean pop music's emotional landscape.
- **Challenges with Artistic Language**: The model faced difficulties with metaphorical or poetic expressions common in song lyrics, where the emotion conveyed is implicit rather than explicit.

6. Limitation

1) Complexity of Emotions

- The complex nature of song lyrics, which often convey mixed emotions, posed a challenge for discrete classification. This complexity led to frequent misclassifications, particularly when dealing with emotions that are not clearly separated.

2) Training Time

- The training process was computationally intensive. The large size of the dataset and the need for fine-tuning on multiple labels resulted in long training times. The training had to be stopped at 2.6 epochs because I don't have enough time to do all epochs(it might take 17-24 hours to be done with KOTE dataset). So I tried only 3 epochs at the second try with AI Hub Corpus Dataset.

7. Conclusion

This project demonstrated the feasibility of using KoBERT, a transformer-based model, for sentiment analysis of Korean song lyrics. Despite the challenges related to data imbalance and the complexity of emotions, the model showed promising results, particularly when using a smaller number of well-defined emotion labels.

- **Future Work**: Future work will involve experimenting with ensemble approaches, leveraging larger datasets, and exploring methods to better understand and classify complex, nuanced emotional expressions in song lyrics.

8. Attachment(Result and Code Implementation)

result for my data: mydata_result.xlsx

koBERT + KOTE dataset	koBERT + AI Hub Corpus dataset
koBERT_with_KOTE.ipynb	koBERT_Sentiment_Anaylsis.ipynb

```
class KoteDataset(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 = self.texts[idx]
        label = int(self.labels[idx][0]) if isinstance(self.labels[idx], list) else int(self.labels[idx])

        inputs = self.tokenizer.encode_plus(
            text,
            add_special_tokens=True,
            max_length=self.max_len,
            padding='max_length',
            truncation=True,
            return_attention_mask=True,
            return_tensors='pt'
        )

        return {
            'input_ids': inputs['input_ids'].flatten(),
            'attention_mask': inputs['attention_mask'].flatten(),
            'labels': torch.tensor(label, dtype=torch.long)
        }
```

```
tokenizer = get_tokenizer()
model = BertForSequenceClassification.from_pretrained("monologg/kobert", num_labels=44)
```

```
def tokenize_fn(text, max_length=128):
    return tokenizer(
        text,
        padding='max_length',
        truncation=True,
        max_length=max_length,
        return_tensors='pt'
    )

def custom_dataset(df):
    input_ids, attention_masks = [], []
    for _, row in df.iterrows():
        tokenized_data = tokenize_fn(row['lyrics'])
        input_ids.append(tokenized_data['input_ids'].squeeze())
        attention_masks.append(tokenized_data['attention_mask'].squeeze())

    return {
        'input_ids': torch.stack(input_ids),
        'attention_mask': torch.stack(attention_masks)
    }

custom_data = custom_dataset(df)

print(f"Custom Data Input IDs Shape: {custom_data['input_ids'].shape}")
```

```
Custom Data Input IDs Shape: torch.Size([100, 128])
```

```
from transformers import Trainer, TrainingArguments, DataCollatorWithPadding

data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=5,
    per_device_train_batch_size=256,
    per_device_eval_batch_size=32,
    warmup_steps=500,$
    weight_decay=0.001,
    logging_dir='./logs',
    logging_steps=10,
    evaluation_strategy="epoch"
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    data_collator=data_collator
)

trainer.train()
```

```
class BERTDataset(Dataset):
    def __init__(self, sentences, labels, tokenizer, max_len, pad=True, pair=False):
        self.sentences = sentences
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_len = max_len
        self.pad = pad
        self.pair = pair

    def __getitem__(self, idx):
        sentence = str(self.sentences[idx])
        label = self.labels[idx]

        encoding = self.tokenizer(
            sentence,
            max_length=self.max_len,
            padding='max_length' if self.pad else False,
            truncation=True,
            return_tensors='pt'
        )

        input_ids = encoding['input_ids'].squeeze(0)
        token_type_ids = encoding['token_type_ids'].squeeze(0)
        attention_mask = encoding['attention_mask'].squeeze(0)

        return input_ids, attention_mask, token_type_ids, torch.tensor(label, dtype=torch.long)

    def __len__(self):
        return len(self.labels)
```

```
tokenizer = KoBERTTokenizer.from_pretrained('skt/kobert-base-v1')
bertmodel = BertModel.from_pretrained('skt/kobert-base-v1', return_dict=False)
summary = SummaryWriter()
```

```
max_len = 128
batch_size = 64
warmup_ratio = 0.1
max_grad_norm = 1
log_interval = 200
learning_rate = 1e-5
```

```
x_train, x_test, y_train, y_test = train_test_split(features, labels, test_size=0.2, stratify=labels, random_state=41)
print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)

train_data = BERTDataset(x_train, y_train, tokenizer, max_len, pad=True, pair=False)
test_data = BERTDataset(x_test, y_test, tokenizer, max_len, pad=True, pair=False)

train_size = int(0.9 * len(train_data))
valid_size = len(train_data) - train_size
train_, valid_ = random_split(train_data, [train_size, valid_size])

train_dataloader = DataLoader(train_, batch_size=batch_size, shuffle=True, num_workers=0)
val_dataloader = DataLoader(valid_, batch_size=batch_size, shuffle=False, num_workers=0)
test_dataloader = DataLoader(test_data, batch_size=batch_size, shuffle=False, num_workers=0)
```

```
(41259,) (10315,) (41259,) (10315,)
```

```
class BERTClassifier(nn.Module):
    def __init__(self, bert, hidden_size=768, num_classes=6, dr_rate=None):
        super(BERTClassifier, self).__init__()
        self.bert = bert
        self.dr_rate = dr_rate
        self.classifier = nn.Linear(hidden_size, num_classes)
        if dr_rate:
            self.dropout = nn.Dropout(p=dr_rate)

    def forward(self, token_ids, attention_mask, segment_ids):
        _, pooler_output = self.bert(input_ids=token_ids,
                                     token_type_ids=segment_ids,
                                     attention_mask=attention_mask,
                                     return_dict=False)

        if self.dr_rate:
            pooler_output = self.dropout(pooler_output)
        return self.classifier(pooler_output)

def calc_accuracy(X, Y):
    max_vals, max_indices = torch.max(X, 1)
    train_acc = (max_indices == Y).sum().data.cpu().numpy() / max_indices.size()[0]
    return train_acc
```

```
model = BERTClassifier(bertmodel, dr_rate=0.5).to(device)
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
loss_fn = nn.CrossEntropyLoss()
```

```
num_epochs = 3

no_decay = ['bias', 'LayerNorm.weight']
optimizer_grouped_parameters = [
    {'params': [p for n, p in model.named_parameters() if not any(nd in n for nd in no_decay)], 'weight_decay': 0.01},
    {'params': [p for n, p in model.named_parameters() if any(nd in n for nd in no_decay)], 'weight_decay': 0.0}
]

optimizer = optim.AdamW(optimizer_grouped_parameters, lr=1e-5)

t_total = len(train_dataloader) * num_epochs
warmup_step = int(t_total * warmup_ratio)
scheduler = get_cosine_schedule_with_warmup(optimizer, num_warmup_steps=warmup_step, num_training_steps=t_total)
```

```
val_results = trainer.evaluate()
print("Validation Results:", val_results)
```

[409/785 9:44:14 < 8:59:44, 0.01 it/s, Epoch 2.60/5]

Epoch	Training Loss	Validation Loss
1	2.095000	1.938385
2	1.473200	1.524684

```
Validation Results: {'eval_loss': 1.5246835947036743}
```

```
from transformers import Trainer, TrainingArguments, DataCollatorWithPadding

data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
pretrained_model = BertForSequenceClassification.from_pretrained('/content/drive/MyDrive/trainedbykote_kobert_model')

training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=5,
    per_device_train_batch_size=256,
    per_device_eval_batch_size=32,
    warmup_steps=500,
    weight_decay=0.001,
    logging_dir='./logs',
    logging_steps=10,
    evaluation_strategy="epoch"
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    data_collator=data_collator
)
```

```
from sklearn.metrics import accuracy_score, classification_report

predictions = trainer.predict(test_dataset)
preds = np.argmax(predictions.predictions, axis=1)

test_labels = [label[0] if isinstance(label, list) else label for label in testset['labels']]

accuracy = accuracy_score(test_labels, preds)
print(f"=> Test Accuracy: {accuracy:.4f}")
print("-------------------------------------")
print(classification_report(test_labels, preds))
```

```
=> Test Accuracy: 0.0026
-------------------------------------
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      2113
           1       0.27      0.00      0.01      1016
           2       0.00      0.00      0.00       531
           3       0.04      0.01      0.01       160
           4       0.00      0.00      0.00       102
           5       0.06      0.02      0.03       188
           6       0.00      0.00      0.00       118
           7       0.00      0.00      0.00        14
           8       0.04      0.03      0.03       220
           9       0.00      0.00      0.00       118
          10       0.00      0.00      0.00       136
          11       0.00      0.00      0.00        18
          12       0.00      0.00      0.00        56
          13       0.00      0.00      0.00        16
          14       0.00      0.00      0.00        14
          15       0.00      0.00      0.00        53
          16       0.00      0.00      0.00        24
          17       0.00      0.00      0.00         4
          18       0.00      0.00      0.00         9
          19       0.00      0.00      0.00         1
          20       0.00      0.00      0.00        12
          21       0.00      0.00      0.00         2
          22       0.00      0.00      0.00         1
          23       0.00      0.00      0.00        13
          24       0.00      0.00      0.00        18
          25       0.00      0.00      0.00         2
```

```
for e in range(num_epochs):
    train_acc = 0.0
    val_acc = 0.0
    model.train()
    lossF = 0

    for batch_id, (token_ids, valid_length, segment_ids, label) in enumerate(tqdm_notebook(train_dataloader)):
        optimizer.zero_grad()
        token_ids = token_ids.long().to(device)
        segment_ids = segment_ids.long().to(device)
        label = label.long().to(device)

        out = model(token_ids, valid_length, segment_ids)
        loss = loss_fn(out, label)
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), max_grad_norm)
        optimizer.step()
        scheduler.step()

        train_acc += calc_accuracy(out, label)
        lossF = loss.data.cpu().numpy()

        if batch_id % log_interval == 0:
            print("Epoch {} Batch ID {} Loss {:.4f} Train Acc {:.4f}".format(
                e+1, batch_id+1, lossF, train_acc / (batch_id+1)
            ))

    print("Epoch {} Train Acc {:.4f}".format(e+1, train_acc / len(train_dataloader)))

    model.eval()
    with torch.no_grad():
        for batch_id, (token_ids, valid_length, segment_ids, label) in enumerate(tqdm_notebook(val_dataloader)):
            token_ids = token_ids.long().to(device)
            segment_ids = segment_ids.long().to(device)
            label = label.long().to(device)

            out = model(token_ids, valid_length, segment_ids)
            val_acc += calc_accuracy(out, label)

    print("Epoch {} Validation Acc {:.4f}".format(e+1, val_acc / len(val_dataloader)))

    summary.add_scalar('Train Accuracy', train_acc / len(train_dataloader), e+1)
    summary.add_scalar('Validation Accuracy', val_acc / len(val_dataloader), e+1)
    summary.add_scalar('Loss', lossF, e+1)
```

```
100% 581/581 [3:32:37<00:00, 17.90s/it]
Epoch 1 Batch ID 1 Loss 1.8581 Train Acc 0.1406
Epoch 1 Batch ID 201 Loss 1.7396 Train Acc 0.1842
Epoch 1 Batch ID 401 Loss 1.3542 Train Acc 0.2960
Epoch 1 Train Acc 0.3625
100% 65/65 [07:55<00:00, 5.72s/it]
Epoch 1 Validation Acc 0.5365
100% 581/581 [3:31:34<00:00, 14.83s/it]
Epoch 2 Batch ID 1 Loss 1.3117 Train Acc 0.6094
Epoch 2 Batch ID 201 Loss 1.3182 Train Acc 0.5520
Epoch 2 Batch ID 401 Loss 1.1044 Train Acc 0.5531
Epoch 2 Train Acc 0.5573
100% 65/65 [06:53<00:00, 5.31s/it]
Epoch 2 Validation Acc 0.5594
100% 581/581 [3:19:37<00:00, 16.87s/it]
Epoch 3 Batch ID 1 Loss 1.2093 Train Acc 0.6250
Epoch 3 Batch ID 201 Loss 1.1904 Train Acc 0.5836
Epoch 3 Batch ID 401 Loss 1.0119 Train Acc 0.5857
Epoch 3 Train Acc 0.5837
100% 65/65 [06:55<00:00, 5.23s/it]
Epoch 3 Validation Acc 0.5603
```

```
model.eval()
test_acc = 0.0
with torch.no_grad():
    for batch_id, (token_ids, attention_mask, segment_ids, label) in enumerate(tqdm_notebook(test_dataloader)):
        token_ids = token_ids.to(device)
        segment_ids = segment_ids.to(device)
        attention_mask = attention_mask.to(device)
        label = label.to(device)

        out = model(token_ids, attention_mask, segment_ids)
        test_acc += calc_accuracy(out, label)

print("Test Accuracy {:.4f}".format(test_acc / len(test_dataloader)))
```

```
100% 162/162 [16:32<00:00, 4.57s/it]
Test Accuracy 0.5618
```

```
aydata['감정_label'] = predicted_labels

class2label = {0: '기쁨', 1: '분노', 2: '상처', 3: '슬픔', 4: '불안', 5: '당황'}
aydata['감정'] = aydata['감정_label'].map(class2label)

aydata
```

	곡명	가사	감정_label	감정
0	APT.	아파트 아파트 아파트 아파트 아파트 아파트 아파트 아파트 아파트 아파...	5	당황
1	Whiplash	집중해 좀 더 이유 넌 이해 못 해 왜 이제야 흔들린 채 무리해도 어디서나 거침없어...	1	분노
2	POWER	역겨 짤 피다 살라살라하다가 난 자유로워 나는 나다워서 아름다워 애들이 나보고 개꿀...	1	분노
3	UP (KARINA Solo)	다들 뻔해 또 거짓말 번듯한 말 어디든 뛰어다녀 여기저기 다 가져갈게 한순간에 타올...	1	분노
4	HAPPY	그런 날이 있을까요 아냥 좋은 그런 날이요 내일 걱정 하나 없이 웃게 되는 그런 날...	5	당황
...	...	...	...	...
95	내가 s면 넌 나의 N이 되어줘	내가 면 넌 나의 이 되어줘어떤 순간에도 너를 찾을 수 있게반대가 끌리는 천만번째 ...	1	분노
96	Get A Guitar	너와 내 느낌대로 시작해 봐 발 박자를 맞추고 손은 모두 집중해 줘 너와 맞추는 눈...	5	당황
97	Super Shy	떨리는 지금도 떨리는 지금도 우리 둘이 나란히 보이지 봐 내 눈이 감자기 빛나지 누...	0	기쁨
98	Over The Moon	너는 나의 미래 내 사공의 질서 널 안고 입 맞출 때 내 겨울이 녹아내려 데려가 줘...	5	당황
99	I'll Be There	이리저리 바쁘게 산 사람들 힘든 세상 어떻게 막 살아 들 작은 것보단 큰 걸 터 주...	1	분노

100 rows × 4 columns