HUST

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

TRƯỜNG ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Natural Language Processing
Middle Project: Song Classification using BERT

Instructor: Dr. Le Thanh Huong
Student: Le Hoang Long - 20232099M - 23B-IT-KHDL-E

ONE LOVE. ONE FUTURE.

CONTENTS

- Song genre classification is an interesting and challenging task in the field of natural language processing (NLP).
    - It involves automatically assigning a genre label to a given song based on its audio features or lyrics.
    - Traditionally, experts manually categorized songs, but with the advent of machine learning and data science, we can now automate this process using algorithms
- BERT (Bidirectional Encoder Representations from Transformers) has become a powerful tool for text classification tasks in natural language processing (NLP).

- Song genre classification is an interesting and challenging task in the field of natural language processing (NLP).
  - It involves automatically assigning a genre label to a given song based on its audio features or lyrics.
  - Traditionally, experts manually categorized songs, but with the advent of machine learning and data science, we can now automate this process using algorithms
- BERT (Bidirectional Encoder Representations from Transformers) has become a powerful tool for text classification tasks in natural language processing (NLP).

- Project overview:
  - The goal is to categorize lyrics into specific genres (e.g., disco, hip-hop, rock) based on their content.
  - Applying transfer learning method to leverage knowledge learned from one task to improve performance on another related task.
- BERT overview
  - It captures bidirectional context by considering both left and right context for each token in a sentence.
  - It captures contextual information effectively.
  - Fine-tuning BERT requires less labeled data compared to training from scratch. It provides better performance and faster convergence.

- Dataset Selection
  - The data is composed of four sources. The initial data was forwarded from Sparktech's 2018 Textract Hackathon. This was enhanced with data from other three kaggle datasets: 150K Lyrics Labeled with Spotify Valence, dataset lyrics musics and AZLyrics song lyrics.
  - Dataset link: https://www.kaggle.com/datasets/mateibejan/multilingual-lyrics-for-genre-classification
- Approaches
  - Applying a pre-trained BERT model on the dataset.
  - Adding a classification layer on top of BERT.
  - Training the entire model on labeled data.

- Data Preprocessing
  - Challenges
    - Imbalanced data: the representation of different genres is not balanced
    - Lyric length: BERT has a maximum token limit
  - Solutions
    - Removing all non english song
    - Handling imbalanced class distributions by choosing an equal number of songs per class
    - Chunking song lyrics

- Solution
  - Link: https://www.kaggle.com/code/lehoanglonglong/hust-song-classification-using-bert
  - Source code:

```
import os
import torch
from torch import nn
from torch.utils.data import DataLoader, Dataset
from transformers import BertTokenizer, BertModel, AdamW,
get_linear_schedule_with_warmup
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
import pandas as pd
```

```
def load_song_data(data_file, bert_model_name, max_length):
  df = pd.read_csv(data_file)
  tokenizer = BertTokenizer.from_pretrained(bert_model_name)
  df['encoding'] = df.apply(lambda x: tokenizer(x['S_Lyric'],
return_tensors='pt', max_length=max_length, padding='max_length',
truncation=True) ,  axis=1)
  encodings = df['encoding'].tolist()
  #texts = df['S_Lyric'].tolist()
  labels = [int(v) for v in df['Genre_Index'].tolist()]
  ids_genres = df[['Genre_Index', 'Genre']].drop_duplicates()
  ids_genres = ids_genres.set_index('Genre_Index')
  return encodings, labels, ids_genres
  #return texts, labels, ids_genres
```

```
class TextClassificationDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_length):
        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_length = max_length
        pass
    def __len__(self):
        return len(self.texts)
    def __getitem__(self, idx):
        encoding = self.texts[idx]
        label = self.labels[idx]
        return {'input_ids': encoding['input_ids'].flatten(), 'attention_mask':
encoding['attention_mask'].flatten(), 'label': torch.tensor(label)}
```

```python
class BERTClassifier(nn.Module):
    def __init__(self, bert_model_name, num_classes):
        super(BERTClassifier, self).__init__()
        self.bert = BertModel.from_pretrained(bert_model_name)
        self.dropout = nn.Dropout(0.1)
        self.fc = nn.Linear(self.bert.config.hidden_size, num_classes)

    def forward(self, input_ids, attention_mask):
        outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask)
        pooled_output = outputs.pooler_output
        x = self.dropout(pooled_output)
        logits = self.fc(x)
        return logits
```

```python
def train(model, data_loader, optimizer, scheduler, device):
    model.train()
    for batch in data_loader:
        optimizer.zero_grad()
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['label'].to(device)
        outputs = model(input_ids=input_ids, attention_mask=attention_mask)
        loss = nn.CrossEntropyLoss()(outputs, labels)
        loss.backward()
        optimizer.step()
        scheduler.step()
```

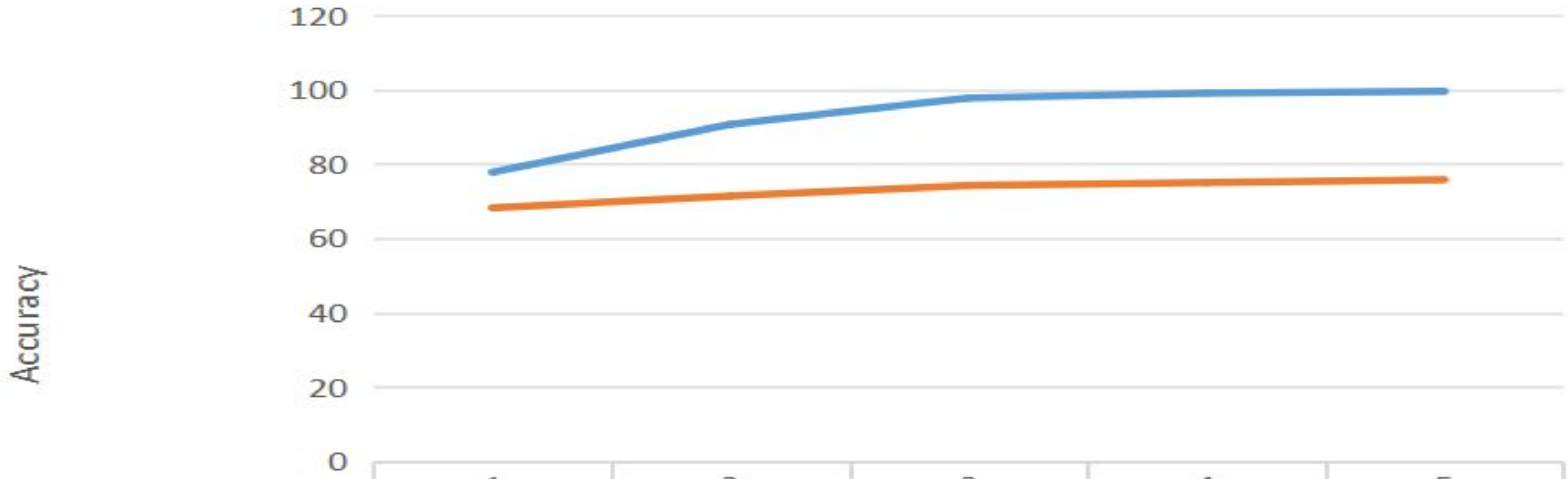
```
def evaluate(model, data_loader, device):
    model.eval()
    predictions = []
    actual_labels = []
    with torch.no_grad():
        for batch in data_loader:
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['label'].to(device)
            outputs = model(input_ids=input_ids, attention_mask=attention_mask)
            _, preds = torch.max(outputs, dim=1)
            predictions.extend(preds.cpu().tolist())
            actual_labels.extend(labels.cpu().tolist())
    return accuracy_score(actual_labels, predictions),
classification_report(actual_labels, predictions)
```

```python
# Set up parameters
bert_model_name = 'bert-base-uncased'
max_length = 128
batch_size = 16
num_epochs = 10
learning_rate = 2e-5
data_file = "/kaggle/input/smallsongs2/l_df.csv"
texts, labels, ids_genres = load_song_data(data_file, bert_model_name,
max_length)
train_texts, val_texts, train_labels, val_labels = train_test_split(texts,
labels, test_size=0.2)
```

```
tokenizer = BertTokenizer.from_pretrained(bert_model_name)
train_dataset = TextClassificationDataset(train_texts, train_labels, tokenizer,
max_length)
val_dataset = TextClassificationDataset(val_texts, val_labels, tokenizer, max_length)
train_dataloader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_dataloader = DataLoader(val_dataset, batch_size=batch_size)
num_classes = ids_genres.shape[0]
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = BERTClassifier(bert_model_name, num_classes) #.to(device)
model = torch.nn.DataParallel(model).to(device) #.to(device).to(device)
optimizer = AdamW(model.parameters(), lr=learning_rate)
total_steps = len(train_dataloader) * num_epochs
scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps=0,
num_training_steps=total_steps)
```

```
for epoch in range(num_epochs):
  print(f"Epoch {epoch + 1}/{num_epochs}")
  train(model, train_dataloader, optimizer, scheduler, device)
  accuracy, report = evaluate(model, train_dataloader, device)
  print(f"Train Accuracy: {accuracy:.4f}")
  accuracy, report = evaluate(model, val_dataloader, device)
  print(f"Validation Accuracy: {accuracy:.4f}")
  torch.save(model.state_dict(), f"version-{epoch}-acc-{accuracy:.4f}.pth")
  print(report)
```

EVALUATION



	1	2	3	4	5
Training	77.8	90.7	97.8	99.1	99.6
Validation	68.2	71.4	74.2	75	75.8

Epoch

- Text classification is a fundamental task in natural language processing (NLP) that involves automatically determining the class or category to which a piece of text belongs.
- We applied transformer-based model in the project (BERT) in order to capture rich contextual information, making it effective for downstream tasks.
- We could gain a better result with BigBird model but the new model asks for a greater GPU resource which is beyond the current investment.
- According to industry standard, a good accuracy is above 70%.

THANK YOU !