NLP - EXP - 7 (Applications of BERT Model)

Atharva Prashant Pawar (9427) - [Batch - D]

Dataset

```
'''
### sentiment_train.csv ###

sentence,label
Ok brokeback mountain is such a horrible movie.,0
Brokeback Mountain was so awesome.,1
friday hung out with kelsie and we went and saw The Da Vinci Code SUCKED!!!!!,0
I am going to start reading the Harry Potter series again because that is one awesome story.,1
"Is it just me, or does Harry Potter suck?...",0
The Da Vinci Code sucked big time.,0
I am going to start reading the Harry Potter series again because that is one awesome story.,1
"For those who are Harry Potter ignorant, the true villains of this movie are awful creatures called dementors.",0
"Harry Potter dragged Draco Malfoy ’ s trousers down past his hips and sucked him into his throat with vigor, making whimpering noises ar
"So as felicia's mom is cleaning the table, felicia grabs my keys and we dash out like freakin mission impossible.",1
I love The Da Vinci Code...,1
'''
```

Fine Tuning Bert Model Code

```
!pip install transformers
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-packages (4.33.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transformers) (3.12.2)
Requirement already satisfied: huggingface-hub<1.0,>=0.15.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.17.1)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (1.23.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from transformers) (23.1)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (2023.6.3)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transformers) (2.31.0)
Requirement already satisfied: tokenizers!=0.11.3,<0.14,>=0.11.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.
Requirement already satisfied: safetensors>=0.3.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.3.3)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from transformers) (4.66.1)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.15.1->transformers)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2023.7.
```

```
import torch
from torch.utils.data import DataLoader, TensorDataset
from transformers import BertTokenizer, BertForSequenceClassification, AdamW, get_linear_schedule_with_warmup
import pandas as pd
import re
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
```

Dataset cleaning and diving : train - validation set

```
# Load the dataset
data = pd.read_csv('sentiment_train.csv')

# Clean and preprocess the sentences (remove special characters, lowercasing, etc.)
def clean_text(text):
    text = re.sub(r'[^A-Za-z0-9]+', ' ', text)
    return text.lower().strip()

data['sentence'] = data['sentence'].apply(clean_text)

# Split the dataset into train and validation sets
train_data, val_data = train_test_split(data, test_size=0.2, random_state=42)
```

Download 'bert-base-uncased' Tokenizer

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

def tokenize_data(data, max_length):
    input_ids = []
    attention_masks = []

    for sentence in data['sentence']:
        encoded = tokenizer.encode_plus(
            sentence,
            add_special_tokens=True,
            max_length=max_length,
            padding='max_length',
            truncation=True,
            return_tensors='pt',
            return_attention_mask=True
        )
        input_ids.append(encoded['input_ids'])
        attention_masks.append(encoded['attention_mask'])

    input_ids = torch.cat(input_ids, dim=0)
    attention_masks = torch.cat(attention_masks, dim=0)
    labels = torch.tensor(data['label'].tolist())

    return TensorDataset(input_ids, attention_masks, labels)

max_length = 128  # Adjust this value as needed
train_dataset = tokenize_data(train_data, max_length)
val_dataset = tokenize_data(val_data, max_length)
```

Download 'bert-base-uncased' Model

```
model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)
```

```
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly ini
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
batch_size = 32  # Adjust this value as needed
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size)
num_epochs = 5 # Adjust this value as needed
```

Training Loop

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

optimizer = AdamW(model.parameters(), lr=2e-5)
scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps=0, num_training_steps=len(train_loader) * num_epochs)

for epoch in range(num_epochs):
    model.train()
    total_loss = 0

    # Train Loop
    for batch in train_loader:
        input_ids, attention_mask, labels = batch
        input_ids, attention_mask, labels = input_ids.to(device), attention_mask.to(device), labels.to(device)

        optimizer.zero_grad()

        outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
        loss = outputs.loss
        total_loss += loss.item()

        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        optimizer.step()
        scheduler.step()
```

```
    average_loss = total_loss / len(train_loader)
    print(f'Epoch {epoch + 1}/{num_epochs}, Train Loss: {average_loss:.4f}')

    # Validation
    model.eval()
    val_loss = 0
    predictions, true_labels = [], []

    for batch in val_loader:
        input_ids, attention_mask, labels = batch
        input_ids, attention_mask, labels = input_ids.to(device), attention_mask.to(device), labels.to(device)

        with torch.no_grad():
            outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
            loss = outputs.loss  # Get the loss value from the outputs

        val_loss += loss.item()
        logits = outputs.logits
        predictions.extend(torch.argmax(logits, dim=1).tolist())
        true_labels.extend(labels.tolist())

    average_val_loss = val_loss / len(val_loader)
    print(f'Epoch {epoch + 1}/{num_epochs}, Validation Loss: {average_val_loss:.4f}')
    print(classification_report(true_labels, predictions))
```

```
Epoch 1/4, Train Loss: 0.0135
Epoch 1/4, Validation Loss: 0.0306
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       492
           1       1.00      1.00      1.00       642

    accuracy                           1.00      1134
   macro avg       1.00      1.00      1.00      1134
weighted avg       1.00      1.00      1.00      1134

Epoch 2/4, Train Loss: 0.0014
Epoch 2/4, Validation Loss: 0.0325
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       492
           1       1.00      1.00      1.00       642

    accuracy                           1.00      1134
   macro avg       1.00      1.00      1.00      1134
weighted avg       1.00      1.00      1.00      1134

Epoch 3/4, Train Loss: 0.0001
Epoch 3/4, Validation Loss: 0.0336
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       492
           1       1.00      1.00      1.00       642

    accuracy                           1.00      1134
   macro avg       1.00      1.00      1.00      1134
weighted avg       1.00      1.00      1.00      1134

Epoch 4/4, Train Loss: 0.0010
Epoch 4/4, Validation Loss: 0.0312
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       492
           1       1.00      1.00      1.00       642

    accuracy                           0.99      1134
   macro avg       0.99      0.99      0.99      1134
weighted avg       0.99      0.99      0.99      1134
```

Save the Model and Tokenizer

```
# Save the fine-tuned model and tokenizer
model.save_pretrained('/content/fine_tuned_model')
tokenizer.save_pretrained('/content/fine_tuned_model')
```

```
('/content/fine_tuned_model/tokenizer_config.json',
 '/content/fine_tuned_model/special_tokens_map.json',
 '/content/fine_tuned_model/vocab.txt',
 '/content/fine_tuned_model/added_tokens.json')
```

Inference (Testing)

```
import torch
from transformers import BertTokenizer, BertForSequenceClassification

# Load the fine-tuned model and tokenizer
model_path = '/content/fine_tuned_model'  # Update with your model path
model = BertForSequenceClassification.from_pretrained(model_path)
tokenizer = BertTokenizer.from_pretrained(model_path)

# Define a function for inference
def predict_sentiment(sentence):
    # Tokenize the input sentence
    inputs = tokenizer(sentence, return_tensors="pt", padding=True, truncation=True, max_length=128)

    # Perform inference
    with torch.no_grad():
        outputs = model(**inputs)

    # Get the predicted label (0 or 1)
    logits = outputs.logits
    predicted_label = torch.argmax(logits, dim=1).item()

    # Map the label to its meaning
    sentiment = "Positive" if predicted_label == 1 else "Negative"

    return sentiment
```

```
Sentiment: Negative
```

Positive Test

```
# Example usage:
test_sentence = "I really enjoyed that movie" # Positive
# test_sentence = "It was a bad movie" # Negative
result = predict_sentiment(test_sentence)
print(f"Sentence: {test_sentence}")
print(f"Sentiment: {result}")
```

```
Sentence: I really enjoyed that movie
Sentiment: Positive
```

Negative Test

```
# Example usage:
# test_sentence = "I really enjoyed that movie" # Positive
test_sentence = "It was a bad movie" # Negative
result = predict_sentiment(test_sentence)
print(f"Sentence: {test_sentence}")
print(f"Sentiment: {result}")
```

```
Sentence: It was a bad movie
Sentiment: Negative
```

Uploading Model on Hugging Face Server

```
!huggingface-cli login
```

```
model.push_to_hub("atharvapawar/Bert-Sentiment-Classification-pos-or-neg", check_pr=True)
```

```
tokenizer.push_to_hub("atharvapawar/Bert-Sentiment-Classification-pos-or-neg",check_pr=True)
```

```
    _|    _|  _|    _|    _|_|_|    _|_|_|  _|_|_|  _|      _|    _|_|_|      _|_|_|_
    _|    _|  _|    _|  _|        _|          _|    _|_|    _|  _|            _|
    _|_|_|_|  _|    _|  _|  _|_|  _|  _|_|    _|    _|  _|  _|  _|  _|_|      _|_|_|
    _|    _|  _|    _|  _|    _|  _|    _|    _|    _|    _|_|  _|    _|      _|
    _|    _|    _|_|      _|_|_|    _|_|_|  _|_|_|  _|      _|    _|_|_|      _|

    To login, `huggingface_hub` requires a token generated from https://huggingface.c
Token:
Add token as git credential? (Y/n) y
Token is valid (permission: write).
Cannot authenticate through git-credential as no helper is defined on your machine.
You might have to re-authenticate when pushing to the Hugging Face Hub.
Run the following command in your terminal in case you want to set the 'store' creden

git config --global credential.helper store
```

Api Testing

```
import requests

API_URL = "https://api-inference.huggingface.co/models/atharvapawar/Bert-Sentiment-Classification-pos-or-neg"
headers = {"Authorization": "Bearer xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"}

# "Bearer xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"} = replace it with your read hugging face token

def query(payload):
    response = requests.post(API_URL, headers=headers, json=payload)
    return response.json()


def inferenceMain(sentence):

  output = query({ "inputs": sentence, })

  for sample in output:
    highest_score_label = max(sample, key=lambda x: x['score'])
    # print(f'Highest Score Label: {highest_score_label["label"]}, Score: {highest_score_label["score"]}')
    if highest_score_label["label"] == "LABEL_1":
      output = f'\n\n Sentiment: Positive \n Sentence : {sentence} \n Score : {round(highest_score_label["score"] * 100, 2)}'
    else:
      output = f'\n\n Sentiment: Negative \n Sentence : {sentence} \n Score : {round(highest_score_label["score"] * 100, 2)}'
    return output


sentenceList = [ "It was a bad movie", "I really enjoyed that movie" ]
for item in sentenceList:
  print(inferenceMain(item))
```

```


 Sentiment: Negative
 Sentence : It was a bad movie
 Score : 99.99


 Sentiment: Positive
 Sentence : I really enjoyed that movie
 Score : 100.0
```