DM2023_Lab2_Report

109062304 林宗翰

Data Preprocessing

```
# Use package json to load tweets in Dataframe
# Use pandas.read_csv to load dataset into dataframe
import json
import pandas as pd
json_filepath = './tweets_DM.json'
df_ident = pd.read_csv('./data_identification.csv')
tweets_list = []
with open(json_filepath, 'r') as file:
    for line in file:
        data = json.loads(line)
        tweets_list.append({
                'tweet_id': data['_source']['tweet']['tweet_id'],
                'text': data['_source']['tweet']['text']
            })

df_tweets = pd.DataFrame(tweets_list)
# Separate dataset between testing and training
df_test = df_ident[df_ident['identification'] == 'test']
df_train = df_ident[df_ident['identification'] == 'train']
df_emo = pd.read_csv('./emotion.csv')
# Merge training dataset with its text
tmp = pd.merge(df_tweets, df_train, on='tweet_id', how='left')
# Merge training dataset with its emotion
df_train_with_emotion = pd.merge(tmp, df_emo, on='tweet_id', how='left')
# Clean the training data
df_train_with_emotion = df_train_with_emotion.dropna()
```

Import packages I need

```
import torch
from torch.utils.data import DataLoader, TensorDataset
from transformers import BertTokenizer, BertForSequenceClassification,
get_linear_schedule_with_warmup
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from tqdm import tqdm
```

```
# Encode every emotion with a label(number)
label_encoder = LabelEncoder()
df_train_with_emotion['label'] =
label_encoder.fit_transform(df_train_with_emotion['emotion'])
```

Model Training Preparation

Due to lack of computation power, I choose to sample 30000 data for each emotion. (240000 data in total)

```
df_train_sample = create_sample(df_train_with_emotion, 30000)
num_classes = len(label_encoder.classes_)
# Separate training set and validation set
X_train, X_val, y_train, y_val = train_test_split(df_train_sample,
df_train_sample['label'], test_size=0.2, random_state=42)
# Select tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
# Encode training dataset
train_encodings = tokenizer.batch_encode_plus(X_train['text'].values,
add_special_tokens=True, padding='max_length', max_length=256,
return_tensors='pt')
val_encodings = tokenizer.batch_encode_plus(X_val['text'].values,
add_special_tokens=True, padding='max_length', max_length=256,
return_tensors='pt')
```

```
# Transform the type of dataset
train_input_ids = train_encodings['input_ids']
train_attention_mask = train_encodings['attention_mask']
train_labels = torch.tensor(X_train['label'].values)
val_input_ids = val_encodings['input_ids']
val_attention_mask = val_encodings['attention_mask']
val_labels = torch.tensor(X_val['label'].values)
train_dataset = TensorDataset(train_input_ids, train_attention_mask, train_labels)
val_dataset = TensorDataset(val_input_ids, val_attention_mask, val_labels)
```

The function I write for creating sample data

```
def create_sample(df, n):
  df0 = df[df['emotion']=='anger'].sample(n=n)
  df1 = df[df['emotion']=='anticipation'].sample(n=n)
  df2 = df[df['emotion']=='disgust'].sample(n=n)
  df3 = df[df['emotion']=='fear'].sample(n=n)
  df4 = df[df['emotion']=='joy'].sample(n=n)
  df5 = df[df['emotion']=='sadness'].sample(n=n)
  df6 = df[df['emotion']=='surprise'].sample(n=n)
  df7 = df[df['emotion']=='trust'].sample(n=n)
```

```
    df_train_sample = pd.concat([df0, df1, df2, df3, df4, df5, df6, df7])
    return df_train_sample
```

Parameters settings

The model I choose is BERT, due to lack of RAM, I set the batch size only to 16 (It will out of memory if I set to 32)

```
# Clean GPU's memory
torch.cuda.empty_cache()
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=16, shuffle=False)
epoch = 3
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(device)
model = BertForSequenceClassification.from_pretrained('bert-base-uncased',
num_labels=num_classes)
model.to(device)
optimizer = torch.optim.AdamW(model.parameters(), lr=1e-5)
scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps=0,
num_training_steps=len(train_loader)*epoch)
```

Model Training

```
for i in range(epoch):
    model.train()
    total_loss = 0
    with tqdm(total=len(train_loader), desc=f"Epoch {i + 1}/{epoch}") as pbar:
        for batch in train_loader:
            optimizer.zero_grad()
            input_ids = batch[0].to(device)
            attention_mask = batch[1].to(device)
            labels = torch.tensor(batch[2], dtype=torch.long).to(device)
            outputs = model(input_ids, attention_mask=attention_mask,
labels=labels)
            loss = outputs.loss
            total_loss += loss
            loss.backward()
            optimizer.step()
            pbar.update(1)
            pbar.set_postfix(loss=f'{loss.item():.4f}')
    scheduler.step()
# Clean GPU's memory
torch.cuda.empty_cache()
```

Evaluation

```
model.eval()
predictions = []
true_labels = []
with tqdm(total=len(val_loader)) as pbar:
    for batch in val_loader:
        optimizer.zero_grad()
        input_ids = batch[0].to(device)
        attention_mask = batch[1].to(device)
        labels = torch.tensor(batch[2], dtype=torch.long).to(device)
        outputs = model(input_ids, attention_mask=attention_mask)
        logits = outputs.logits
        predictions.extend(torch.argmax(logits, dim=1).cpu().numpy())
        true_labels.extend(labels.cpu().numpy())
        pbar.update(1)
print(accuracy_score(true_labels, predictions))
print(classification_report(true_labels, predictions))
```

Testing dataset

Just like training dataset

```
df_test_label = pd.merge(df_tweets, df_test, on='tweet_id', how='left')
df_test_label = df_test_label.dropna()
test_encodings = tokenizer(list(df_test_label['text']), truncation=True,
padding=True, max_length=128, return_tensors='pt')
test_input_ids = test_encodings['input_ids']
test_attention_mask = test_encodings['attention_mask']
test_dataset = TensorDataset(test_input_ids, test_attention_mask)
```

Prediction

```
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)
predicted_emotion = []
model.eval()
with torch.no_grad():
    with tqdm(total=len(test_loader)) as pbar:
        for batch in test_loader:
            input_ids = batch[0].to(device)
            attention_mask = batch[1].to(device)
            outputs = model(input_ids, attention_mask=attention_mask)
            logits = outputs.logits
            predicted_emotion.extend(torch.argmax(logits, dim=1).cpu().numpy())
            pbar.update(1)
```

Transfrom to csv file

```
# Use inverse_transform to transfrom the label back into the emotion
df_test_label['emotion'] = label_encoder.inverse_transform(predicted_emotion)
submission = pd.concat([df_test_label['tweet_id'], df_test_label['emotion']], axis
= 1)
submission.columns = ['id', 'emotion']
submission.to_csv('submission.csv', index=False)
```