test

April 23, 2025

[1]:
```
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from transformers import AutoTokenizer, AutoModelForSequenceClassification
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
from tqdm import tqdm
import pandas as pd
import numpy as np
import os
```

[2]:
```
# Load the dataset containing generated sentences with gender and model metadata
df = pd.read_csv("../Phase_02/output/sentences_cleaned.csv")

# Add a 'label' column based on the combination of noun and adjective gender:
# - "MM": male noun + male adjective
# - "FF": female noun + female adjective
# - "MF": male noun + female adjective
# - "FM": female noun + male adjective
df["label"] = df.apply(
    lambda row: "MM" if row["noun_gender"] == "male" and␣
 ↪row["adjective_gender"] == "male"
    else "FF" if row["noun_gender"] == "female" and row["adjective_gender"] ==␣
 ↪"female"
    else "MF" if row["noun_gender"] == "male" and row["adjective_gender"] ==␣
 ↪"female"
    else "FM", axis=1
)

# Create a binary 'stereotype' column:
# Assign 1 to stereotypical combinations (MM or FF), and 0 otherwise (MF or FM)
df["stereotype"] = df["label"].apply(lambda x: 1 if x in ["MM", "FF"] else 0)
df["stratify_group"] = df["stereotype"].astype(str) + "_" + df["model"].
 ↪astype(str) + "_" + df["temperature"].astype(str)

sentences = df["sentence"].tolist()
labels = df["stereotype"].astype(int).tolist()
```

```
stratify_labels = df["stratify_group"].tolist()
```

[3]:
```
class BiasDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_len=128):
        self.encodings = tokenizer(texts, padding=True, truncation=True,␣
 ↪max_length=max_len, return_tensors="pt")
        self.labels = torch.tensor(labels)

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

    def __getitem__(self, idx):
        item = {key: val[idx] for key, val in self.encodings.items()}
        item["labels"] = self.labels[idx]
        return item
```

[4]:
```
def train(model, dataloader, optimizer, device):
    model.train()
    total_loss = 0
    for batch in tqdm(dataloader):
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = model(**batch)
        loss = outputs.loss
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
        total_loss += loss.item()
    return total_loss / len(dataloader)

def evaluate(model, dataloader, device):
    model.eval()
    preds, labels = [], []
    with torch.no_grad():
        for batch in dataloader:
            batch = {k: v.to(device) for k, v in batch.items()}
            outputs = model(**batch)
            logits = outputs.logits
            preds.extend(torch.argmax(logits, axis=1).cpu().numpy())
            labels.extend(batch["labels"].cpu().numpy())
    acc = accuracy_score(labels, preds)
    prec, rec, f1, _ = precision_recall_fscore_support(labels, preds,␣
 ↪average="binary")
    return acc, prec, rec, f1
```

[5]:
```
def run_training(model_name, tokenizer_name, train_texts, train_labels,␣
 ↪test_texts, test_labels, fold):
    tokenizer = AutoTokenizer.from_pretrained(tokenizer_name)
```

```
    model = AutoModelForSequenceClassification.from_pretrained(model_name,␣
 ↪num_labels=2)

    train_dataset = BiasDataset(train_texts, train_labels, tokenizer)
    test_dataset = BiasDataset(test_texts, test_labels, tokenizer)
    train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True)
    test_loader = DataLoader(test_dataset, batch_size=8)

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model.to(device)
    optimizer = torch.optim.AdamW(model.parameters(), lr=2e-5)

    for epoch in range(3):
        print(f"Epoch {epoch + 1}")
        train(model, train_loader, optimizer, device)

    acc, prec, rec, f1 = evaluate(model, test_loader, device)

    # Opslaan model
    save_dir = f"trained_models/{model_name.replace('/', '_')}/fold_{fold}"
    os.makedirs(save_dir, exist_ok=True)
    model.save_pretrained(save_dir)
    tokenizer.save_pretrained(save_dir)

    return acc, prec, rec, f1
```

[ ]:
```
models = {
    "GroNLP/bert-base-dutch-cased": "GroNLP/bert-base-dutch-cased",
    "bert-base-multilingual-cased": "bert-base-multilingual-cased",
    "DTAI-KULeuven/robbert-2023-dutch-large": "DTAI-KULeuven/
 ↪robbert-2023-dutch-large"
}

results = []

skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

for fold, (train_idx, test_idx) in enumerate(skf.split(sentences, labels)):
    print(f"\n Fold {fold + 1}")
    train_texts = [sentences[i] for i in train_idx]
    test_texts = [sentences[i] for i in test_idx]
    train_labels = [labels[i] for i in train_idx]
    test_labels = [labels[i] for i in test_idx]

    for model_name, tokenizer_name in models.items():
        print(f"\n Model: {model_name}")
```

```
        acc, prec, rec, f1 = run_training(model_name, tokenizer_name,␣
↪train_texts, train_labels, test_texts, test_labels, fold)
        results.append({
            "model": model_name,
            "fold": fold + 1,
            "accuracy": acc,
            "precision": prec,
            "recall": rec,
            "f1_score": f1
        })
```

```
 Fold 1


 Model: GroNLP/bert-base-dutch-cased
```

```
Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at GroNLP/bert-base-dutch-cased and are newly initialized:
['bert.pooler.dense.bias', 'bert.pooler.dense.weight', 'classifier.bias',
'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

```
Epoch 1
```

```
  0%|          | 0/1301 [00:00<?, ?it/s]huggingface/tokenizers: The current
process just got forked, after parallelism has already been used. Disabling
parallelism to avoid deadlocks…
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true |
false)
  7%|          | 91/1301 [27:46<32:30,  1.61s/it]
```

[ ]:
```
results_df = pd.DataFrame(results)
results_df.to_csv("bias_detection_results.csv", index=False)

# === 7. Gemiddelde prestaties per model ===
print("\n Gemiddelde prestaties per model:")
summary = results_df.groupby("model").agg({
    "accuracy": ["mean", "std"],
    "precision": ["mean", "std"],
    "recall": ["mean", "std"],
    "f1_score": ["mean", "std"]
})
print(summary)
```