Hate Speech Detection using BERT

1. Introduction

This document provides an overview of the hate speech classification project using BERT. The project involves training a deep learning model to classify tweets as either hate speech or non-hate speech.

---

2. Steps Overview

The project consists of the following steps:

1. **Load Dataset** - Importing training and test data.
2. **Preprocessing** - Cleaning text data to remove noise.
3. **Tokenization** - Converting text into numerical format using BERT tokenizer.
4. **Model Definition** - Creating a deep learning model based on BERT.
5. **Training the Model** - Training the model using labeled data.
6. **Evaluation** - Assessing model performance on validation data.
7. **Inference on Test Data** - Classifying tweets in the test dataset and saving results.

---

3. Dataset

- `train.csv` contains labeled tweets (`tweet` and `label` columns).
- `test.csv` contains unlabeled tweets to be classified.

---

4. Implementation Details

4.1 Load Dataset

```
import pandas as pd

train_df = pd.read_csv("train.csv")
test_df = pd.read_csv("test.csv")
```

4.2 Preprocessing

```
import re

def clean_text(text):
    text = text.lower()  # Convert to lowercase
    text = re.sub(r'@\w+', '', text)  # Remove mentions
    text = re.sub(r'http\S+', '', text)  # Remove URLs
    text = re.sub(r'[^a-zA-Z0-9 ]', '', text)  # Remove special characters
    return text

train_df['tweet'] = train_df['tweet'].apply(clean_text)
test_df['tweet'] = test_df['tweet'].apply(clean_text)
```

4.3 Tokenization using BERT

```
from transformers import BertTokenizer

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

def tokenize_text(text, max_length=64):
    tokens = tokenizer(text, padding='max_length', truncation=True, max_length=max_length,
return_tensors="pt")
    return tokens.input_ids.squeeze(), tokens.attention_mask.squeeze()
```

4.4 Creating Dataset Class

```
import torch
from torch.utils.data import Dataset, DataLoader

class TweetDataset(Dataset):
    def __init__(self, texts, labels):
        self.texts = texts
        self.labels = labels

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

    def __getitem__(self, idx):
        input_ids, attention_mask = tokenize_text(self.texts[idx])
        return {
            'input_ids': input_ids,
            'attention_mask': attention_mask,
            'label': torch.tensor(self.labels[idx], dtype=torch.long)
```

```
    }
```

4.5 Train-Test Split

```
from sklearn.model_selection import train_test_split

X_train, X_val, y_train, y_val = train_test_split(train_df['tweet'].values, train_df['label'].values,
test_size=0.2)
train_dataset = TweetDataset(X_train, y_train)
val_dataset = TweetDataset(X_val, y_val)

train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=16, shuffle=False)
```

4.6 Define Model

```
import torch.nn as nn
from transformers import BertModel

class HateTweetClassifier(nn.Module):
  def __init__(self):
    super(HateTweetClassifier, self).__init__()
    self.bert = BertModel.from_pretrained('bert-base-uncased')
    self.dropout = nn.Dropout(0.3)
    self.fc = nn.Linear(768, 2)  # 2 classes (hate/non-hate)

  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)
    return self.fc(x)
```

4.7 Train the Model

```
import torch.optim as optim

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

for epoch in range(3):
  model.train()
```

```
    total_loss = 0
    for batch in train_loader:
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['label'].to(device)

        optimizer.zero_grad()
        outputs = model(input_ids, attention_mask)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    print(f"Epoch {epoch+1}, Loss: {total_loss/len(train_loader)}")
```

4.8 Save and Load Model

```
torch.save(model.state_dict(), "hate_tweet_model.pth")

model.load_state_dict(torch.load("hate_tweet_model.pth"))
model.to(device)
model.eval()
```

4.9 Evaluate Model

```
from sklearn.metrics import classification_report

model.eval()
y_true, y_pred = [], []

with torch.no_grad():
    for batch in val_loader:
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['label'].cpu().numpy()

        outputs = model(input_ids, attention_mask)
        predictions = torch.argmax(outputs, dim=1).cpu().numpy()

        y_true.extend(labels)
        y_pred.extend(predictions)

print("Classification Report:")
```

```
print(classification_report(y_true, y_pred))
```

---

5. Training Results

Classification Report:

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.98      5955
           1       0.82      0.76      0.79       438

    accuracy                           0.97      6393
   macro avg       0.90      0.87      0.89      6393
weighted avg       0.97      0.97      0.97      6393
```

Training Loss per Epoch:

Epoch 1, Loss: 0.1304609259361025
Epoch 2, Loss: 0.05634273243460286
Epoch 3, Loss: 0.01909496094166075

---

6. Classifying Test Data & Saving Results

```
def classify_tweets(df):
    df['prediction'] = df['tweet'].apply(lambda x: predict_tweet(x))
    df.to_csv("classified_test_results.csv", index=False)
    print("Results saved successfully!")
```

This will create a CSV file `classified_test_results.csv` containing the original tweets with their predicted labels.

---

7. Conclusion

This project successfully builds a hate speech classifier using BERT, achieving 97% accuracy. The trained model can be used to classify new tweets effectively.