Emotion Detection Using Text

# Introduction

Content not provided in the notebook.

# Data Preprocessing

Content not provided in the notebook.

# Label Encoding

Content not provided in the notebook.

# Model Training

Content not provided in the notebook.

# Evaluation

Content not provided in the notebook.

# Conclusion

Content not provided in the notebook.

# Code Snippets

```python  
import torch  
from torch.utils.data import Dataset, DataLoader, TensorDataset, RandomSampler, SequentialSampler  
from transformers import BertTokenizer, BertForSequenceClassification, AdamW, get\_linear\_schedule\_with\_warmup  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import LabelEncoder  
import numpy as np  
import pandas as pd  
from tqdm.notebook import tqdm  
```  
  
```python  
df = pd.read\_csv(  
 '/content/tweet\_emotions.csv',  
 usecols=['content', 'sentiment'],  
 dtype={'content': 'string', 'sentiment': 'category'}  
)  
```  
  
```python  
df = df.rename(columns={'content': 'tweet', 'sentiment': 'label'})  
#labels = ['neutral', 'sadness', 'happiness']  
#df = df.query('label in @labels')  
```  
  
```python  
#df['label'] = df['label'].replace('sadness', 'negative')  
#df['label'] = df['label'].replace('happiness', 'positive')  
#data = df.dropna().reset\_index(drop=True)  
```  
  
```python  
#data\_pos = data.query('label == "positive"')[:5000]  
#data\_neu = data.query('label == "neutral"')[:5000]  
#data\_neg = data.query('label == "negative"')[:5000]  
  
data\_final = df #.concat([data\_pos, data\_neu, data\_neg])  
```  
  
```python  
# Encode labels  
label\_encoder = LabelEncoder()  
data\_final['label\_enc'] = label\_encoder.fit\_transform(data\_final['label'])  
  
data\_final.rename(columns={'label':'label\_desc'},inplace=True)  
data\_final.rename(columns={'label\_enc':'label'},inplace=True)  
```  
  
```python  
label\_map = {}  
for index, row in data\_final[['label\_desc', 'label']].drop\_duplicates(keep='first').iterrows():  
 label\_map[row['label']] = row['label\_desc']  
label\_map  
```  
  
```python  
# Split data  
X = data\_final['tweet']  
y = data\_final['label']  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)  
X\_train = X\_train.reset\_index(drop=True)  
X\_test = X\_test.reset\_index(drop=True)  
y\_train = y\_train.reset\_index(drop=True)  
y\_test = y\_test.reset\_index(drop=True)  
```  
  
```python  
device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")  
```  
  
```python  
# Initialize BERT tokenizer and model  
tokenizer = BertTokenizer.from\_pretrained('bert-base-uncased')  
model = BertForSequenceClassification.from\_pretrained("bert-base-uncased", num\_labels=len(label\_encoder.classes\_)).to(device)  
  
# Optimizer and learning rate scheduler  
optimizer = AdamW(model.parameters(), lr=2e-5, eps=1e-8)  
epochs = 3  
```  
  
```python  
train\_inputs = [tokenizer.encode(sent, add\_special\_tokens=True, max\_length=256, pad\_to\_max\_length=True) for sent in X\_train]  
test\_inputs = [tokenizer.encode(sent, add\_special\_tokens=True, max\_length=256, pad\_to\_max\_length=True) for sent in X\_test]  
  
# Create attention masks  
train\_masks = [[float(i > 0) for i in seq] for seq in train\_inputs]  
test\_masks = [[float(i > 0) for i in seq] for seq in test\_inputs]  
  
# Convert all data into torch tensors  
train\_inputs = torch.tensor(train\_inputs)  
train\_masks = torch.tensor(train\_masks)  
train\_labels = torch.tensor(y\_train.values)  
  
test\_inputs = torch.tensor(test\_inputs)  
test\_masks = torch.tensor(test\_masks)  
test\_labels = torch.tensor(y\_test.values)  
```  
  
```python  
batch\_size = 32  
  
# Create DataLoader for training set  
train\_data = TensorDataset(train\_inputs, train\_masks, train\_labels)  
train\_sampler = RandomSampler(train\_data)  
train\_dataloader = DataLoader(train\_data, sampler=train\_sampler, batch\_size=batch\_size)  
  
# Create DataLoader for test set  
test\_data = TensorDataset(test\_inputs, test\_masks, test\_labels)  
test\_sampler = SequentialSampler(test\_data)  
test\_dataloader = DataLoader(test\_data, sampler=test\_sampler, batch\_size=batch\_size)  
```  
  
```python  
total\_steps = len(train\_dataloader) \* epochs  
scheduler = get\_linear\_schedule\_with\_warmup(optimizer, num\_warmup\_steps=0, num\_training\_steps=total\_steps)  
```  
  
```python  
# Training loop  
for epoch in range(epochs):  
 print(f"Epoch {epoch + 1}/{epochs}")  
 model.train()  
  
 total\_loss = 0  
 train\_accuracy = 0  
 for step, batch in enumerate(tqdm(train\_dataloader, desc="Training")):  
 batch\_input\_ids, batch\_input\_mask, batch\_labels = batch  
 batch\_input\_ids = batch\_input\_ids.to(device)  
 batch\_input\_mask = batch\_input\_mask.to(device)  
 batch\_labels = batch\_labels.to(device)  
  
 model.zero\_grad()  
  
 outputs = model(batch\_input\_ids, token\_type\_ids=None, attention\_mask=batch\_input\_mask, labels=batch\_labels)  
 loss = outputs.loss  
 logits = outputs.logits  
  
 total\_loss += loss.item()  
 loss.backward()  
  
 torch.nn.utils.clip\_grad\_norm\_(model.parameters(), 1.0)  
 optimizer.step()  
 scheduler.step()  
  
 preds = torch.argmax(logits, dim=1)  
 train\_accuracy += (preds == batch\_labels).sum().item() / len(batch\_labels)  
  
 avg\_train\_loss = total\_loss / len(train\_dataloader)  
 avg\_train\_accuracy = train\_accuracy / len(train\_dataloader)  
 print(f"Training loss: {avg\_train\_loss:.4f}")  
 print(f"Training accuracy: {avg\_train\_accuracy:.4f}")  
  
 # Validation loop  
 model.eval()  
 val\_accuracy = 0  
 for batch in tqdm(test\_dataloader, desc="Validation"):  
 batch\_input\_ids, batch\_input\_mask, batch\_labels = batch  
 batch\_input\_ids = batch\_input\_ids.to(device)  
 batch\_input\_mask = batch\_input\_mask.to(device)  
 batch\_labels = batch\_labels.to(device)  
  
 with torch.no\_grad():  
 outputs = model(batch\_input\_ids, token\_type\_ids=None, attention\_mask=batch\_input\_mask, labels=batch\_labels)  
  
 logits = outputs.logits  
 preds = torch.argmax(logits, dim=1)  
 val\_accuracy += (preds == batch\_labels).sum().item() / len(batch\_labels)  
  
 avg\_val\_accuracy = val\_accuracy / len(test\_dataloader)  
 print(f"Validation accuracy: {avg\_val\_accuracy:.4f}")  
```  
  
```python  
model.save\_pretrained('./bert-emotion-classifier')  
tokenizer.save\_pretrained('./bert-emotion-classifier')  
```  
  
```python  
model\_path = './bert-emotion-classifier'  
tokenizer = BertTokenizer.from\_pretrained(model\_path)  
model = BertForSequenceClassification.from\_pretrained(model\_path)  
model.to(device)  
model.eval()  
```  
  
```python  
def predict\_emotion(text):  
 inputs = tokenizer.encode\_plus(  
 text,  
 add\_special\_tokens=True,  
 max\_length=128,  
 padding='max\_length',  
 return\_attention\_mask=True,  
 return\_tensors='pt',  
 truncation=True  
 )  
 input\_ids = inputs['input\_ids'].to(device)  
 attention\_mask = inputs['attention\_mask'].to(device)  
  
 with torch.no\_grad():  
 outputs = model(input\_ids, attention\_mask=attention\_mask)  
  
 logits = outputs.logits  
 predicted\_class = torch.argmax(logits, dim=1).item()  
 return label\_encoder.inverse\_transform([predicted\_class])[0]  
```  
  
```python  
example\_indices = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
example\_texts = X\_test.iloc[example\_indices].tolist()  
example\_labels = y\_test.iloc[example\_indices].tolist()  
  
for text, true\_label in zip(example\_texts, example\_labels):  
 predicted\_label = predict\_emotion(text)  
 print(f"Text: {text}")  
 print(f"True Label: {label\_map[true\_label]}")  
 print(f"Predicted Label: {predicted\_label}\n")  
```