Arabic Sentiment Analysis

Amira Sayed Mohamed Ali Hemdan

Project NLP

Presented By:
Amira Sayed

Overview

1. Importing Libraries
2. Load and Split Dataset
3. Preprocessing: Tokenizing using BERT
4. Preparing Datasets
5. Defining BERT-Based Classification Model
6. Training the BERT Models
7. Evaluating the BERT Model
8. Implementing RNN-Based Models
9. RNN
10. LSTM
11. Bi-Directional LSTM

1. Importing Librarie

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import torch
from transformers import BertTokenizer, BertModel, GPT2Tokenizer, GPT2Model
import tensorflow as tf
from tensorflow.keras.layers import RNN, LSTM, Bidirectional, Dense, Embedding
from tensorflow.keras.models import Sequential
from torch.utils.data import DataLoader, Dataset
from torch import nn
from tqdm import tqdm
import matplotlib.pyplot as plt
import seaborn as sns
```

1. Load Data

```
data="/kaggle/input/330k-arabic-sentiment-reviews/arabic_sentiment_reviews.csv"
df=pd.read_csv(data)
```

```
df.head(5)
```

	label	content
0	1	النعال المريحة: أرتدي هذه النعال كثيرًا!فهي دا...
1	1	منتج جميل ، خدمة سيئة: لقد اشتريت زوجًا من الن...
2	1	جيد للأشياء الصغيرة: هذا يعمل بشكل جيد لالتقاط...
3	0	واهية للغاية: flimsyif ، فأنت تشتريه ، للغاية ، ...
4	1	Pop for Girls and Girly Boys ، والأشخاص الذين ...

```
df.shape
```

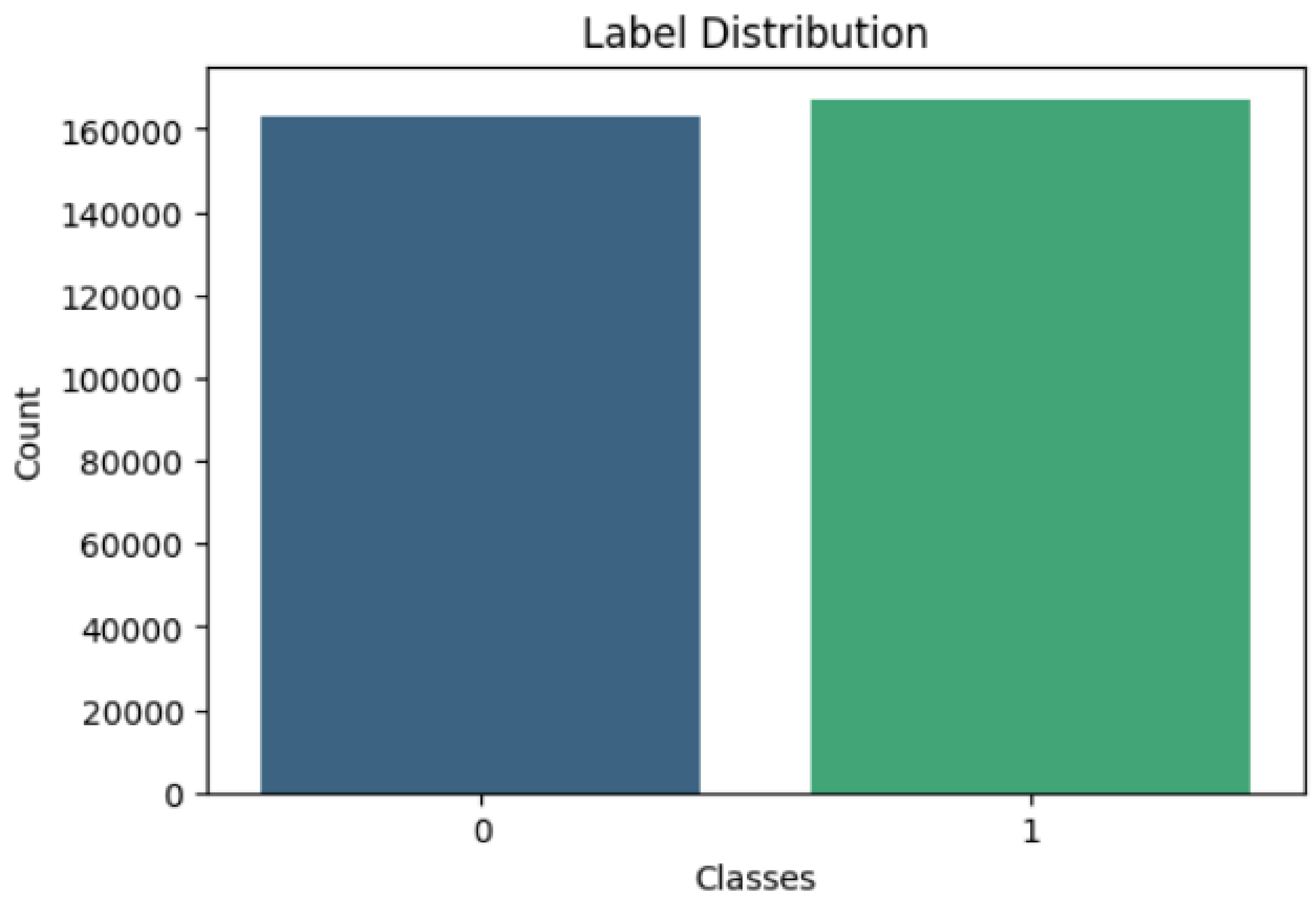
(329968, 2)

```
df.tail(5)
```

	label	
329995	0	DOA: لامة التجارية الجديدة من
329996	0	ل معها: المنتج كان على ما يرام
329997	0	SDK Sansa Leather Case: لا
329998	0	لقًا: حسنًا ، لقد اشتريت هذا
329999	1	نعال رائعة!أنها ناعمة جدا وم

2.Data Visualization

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(6, 4))
sns.countplot(x=df["label"], palette="viridis")
plt.title("Label Distribution")
plt.xlabel("Classes")
plt.ylabel("Count")
plt.show()
```



3.Data Cleaning


```
[6]: df.isnull().sum()

[6]: label      0
     content    0
     dtype: int64

+ Code   + Markdown

[7]: df.duplicated().sum()

[7]: 32

[8]: df = df.drop_duplicates()

[9]: df.duplicated().sum()

[9]: 0
```

4. Splitting Data into Training & Testing

```
# Split data 70% for training, 30% for testing
X_train, X_test, y_train, y_test = train_test_split(df['content'], df['label'], test_size=0.3, random_state=42)
```

5.Preprocessing: Tokenizing using BERT

```
# Preprocess text (tokenization using BERT tokenizer)
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
```

```
tokenizer_config.json: 100% 48.0/48.0 [00:00<00:00, 4.63kB/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 1.47MB/s]
tokenizer.json: 100% 466k/466k [00:00<00:00, 2.94MB/s]
config.json: 100% 570/570 [00:00<00:00, 62.9kB/s]
```

```
def encode_texts(texts, tokenizer, max_length=512):
    inputs = tokenizer(texts.tolist(), padding=True, truncation=True, max_length=max_length, return_tensors='pt')
    return inputs
```

6. Preparing Datasets

```python
# Prepare datasets for fine-tuning
class TextDataset(Dataset):
    def __init__(self, inputs, labels):
        self.inputs = inputs
        self.labels = labels

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

    def __getitem__(self, idx):
        return {key: val[idx] for key, val in self.inputs.items()}, self.labels.iloc[idx]

train_dataset = TextDataset(train_inputs, y_train)
test_dataset = TextDataset(test_inputs, y_test)
```

```python
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

7. Defining BERT-Based Classification Model

```
# Define BERT-based model for classification
class BertForSequenceClassification(nn.Module):
    def __init__(self, dropout=0.3):
        super(BertForSequenceClassification, self).__init__()
        self.bert = BertModel.from_pretrained('bert-base-uncased')
        self.dropout = nn.Dropout(dropout)
        self.fc = nn.Linear(self.bert.config.hidden_size, 2)  # 2 classes

    def forward(self, input_ids, attention_mask):
        outputs = self.bert(input_ids, attention_mask=attention_mask)
        pooler_output = outputs.pooler_output
        output = self.dropout(pooler_output)
        return self.fc(output)
```

]:

```
# Initialize BERT model for classification
model_bert = BertForSequenceClassification()
```

8.Training the BERT

```
# Train and evaluate the BERT model
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model_bert.to(device)

optimizer = torch.optim.Adam(model_bert.parameters(), lr=2e-5)
criterion = nn.CrossEntropyLoss()
```

```python
def train_model(model, train_loader, optimizer, criterion, device, patience=3):
    model.train()
    total_loss = 0
    best_loss = float('inf')
    epochs_no_improve = 0

    for epoch in range(10):  # Maximum number of epochs
        epoch_loss = 0
        for batch in tqdm(train_loader):
            inputs, labels = batch
            input_ids = inputs['input_ids'].to(device)
            attention_mask = inputs['attention_mask'].to(device)
            labels = labels.to(device)

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

            epoch_loss += loss.item()

        avg_loss = epoch_loss / len(train_loader)
        print(f"Epoch {epoch+1}, Loss: {avg_loss}")

        if avg_loss < best_loss:
            best_loss = avg_loss
            epochs_no_improve = 0
        else:
            epochs_no_improve += 1
            if epochs_no_improve == patience:
                print(f'Early stopping at epoch {epoch+1}')
                break

    return best_loss
```

9 Evaluating the BERT Model

```
Evaluating BERT Model:
100%|██████████| 3094/3094 [51:16<00:00,  1.01it/s]
              precision    recall  f1-score   support

           0       0.81      0.91      0.86     48812
           1       0.90      0.80      0.85     50179

    accuracy                           0.85     98991
   macro avg       0.86      0.85      0.85     98991
weighted avg       0.86      0.85      0.85     98991
```

```python
def evaluate_model(model, test_loader, device):
    model.eval()
    y_true, y_pred = [], []
    with torch.no_grad():
        for batch in tqdm(test_loader):
            inputs, labels = batch
            input_ids = inputs['input_ids'].to(device)
            attention_mask = inputs['attention_mask'].to(device)
            labels = labels.to(device)

            outputs = model(input_ids, attention_mask)
            _, preds = torch.max(outputs, dim=1)

            y_true.extend(labels.cpu().numpy())
            y_pred.extend(preds.cpu().numpy())

    return classification_report(y_true, y_pred)
```

```python
# Evaluate BERT model
print("Evaluating BERT Model:")
print(evaluate_model(model_bert, test_loader, device))
```

10. Implementing RNN, LSTM , bi-Rnn, bi-lstm

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, LSTM, Bidirectional, Dense
from tensorflow.keras.callbacks import EarlyStopping

# Function to build an RNN model
def build_rnn_model(input_dim, output_dim):
    model = Sequential([
        Embedding(input_dim=input_dim, output_dim=128, input_length=512),
        SimpleRNN(128, return_sequences=True),
        SimpleRNN(128),
        Dense(output_dim, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model

# Function to build an LSTM model
def build_lstm_model(input_dim, output_dim):
    model = Sequential([
        Embedding(input_dim=input_dim, output_dim=128, input_length=512),
        LSTM(128, return_sequences=True),
        LSTM(128),
        Dense(output_dim, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model

# Function to build a Bi-Directional RNN model
def build_bi_rnn_model(input_dim, output_dim):
    model = Sequential([
        Embedding(input_dim=input_dim, output_dim=128, input_length=512),
        Bidirectional(SimpleRNN(128, return_sequences=True)),
        Bidirectional(SimpleRNN(128)),
        Dense(output_dim, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model

# Function to build a Bi-Directional LSTM model
def build_bi_lstm_model(input_dim, output_dim):
    model = Sequential([
        Embedding(input_dim=input_dim, output_dim=128, input_length=512),
        Bidirectional(LSTM(128, return_sequences=True)),
        Bidirectional(LSTM(128)),
        Dense(output_dim, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model
```

11. Evaluate models after training

accuracy 91%

```python
rnn_model.fit(train_inputs['input_ids'], y_train, epochs=10, batch_size=32, validation_split=0.2, callbacks=[early_stopping])
lstm_model.fit(train_inputs['input_ids'], y_train, epochs=10, batch_size=32, validation_split=0.2, callbacks=[early_stopping])
bi_rnn_model.fit(train_inputs['input_ids'], y_train, epochs=10, batch_size=32, validation_split=0.2, callbacks=[early_stopping])
bi_lstm_model.fit(train_inputs['input_ids'], y_train, epochs=10, batch_size=32, validation_split=0.2, callbacks=[early_stopping])
```

```
Epoch 3/10
5775/5775 ━━━━━━━━━━━━━━━━━━━━ 566s 98ms/step - accuracy: 0.5014 - loss: 0.6977 - val_accuracy: 0.4902 - val_loss: 0.7008
Epoch 4/10
5775/5775 ━━━━━━━━━━━━━━━━━━━━ 567s 98ms/step - accuracy: 0.5066 - loss: 0.6976 - val_accuracy: 0.4902 - val_loss: 0.7116
Epoch 5/10
5775/5775 ━━━━━━━━━━━━━━━━━━━━ 568s 98ms/step - accuracy: 0.5044 - loss: 0.6980 - val_accuracy: 0.4927 - val_loss: 0.6940
Epoch 1/10
5775/5775 ━━━━━━━━━━━━━━━━━━━━ 210s 36ms/step - accuracy: 0.5077 - loss: 0.6932 - val_accuracy: 0.5147 - val_loss: 0.6927
Epoch 2/10
5775/5775 ━━━━━━━━━━━━━━━━━━━━ 206s 36ms/step - accuracy: 0.5108 - loss: 0.6928 - val_accuracy: 0.5147 - val_loss: 0.6927
Epoch 3/10
5775/5775 ━━━━━━━━━━━━━━━━━━━━ 207s 36ms/step - accuracy: 0.5175 - loss: 0.6909 - val_accuracy: 0.5257 - val_loss: 0.6895
Epoch 4/10
5775/5775 ━━━━━━━━━━━━━━━━━━━━ 208s 36ms/step - accuracy: 0.5244 - loss: 0.6897 - val_accuracy: 0.5258 - val_loss: 0.6892
Epoch 5/10
5775/5775 ━━━━━━━━━━━━━━━━━━━━ 207s 36ms/step - accuracy: 0.5224 - loss: 0.6901 - val_accuracy: 0.4902 - val_loss: 0.6935
Epoch 6/10
5775/5775 ━━━━━━━━━━━━━━━━━━━━ 208s 36ms/step - accuracy: 0.5077 - loss: 0.6928 - val_accuracy: 0.5145 - val_loss: 0.6940
Epoch 7/10
5775/5775 ━━━━━━━━━━━━━━━━━━━━ 208s 36ms/step - accuracy: 0.5105 - loss: 0.6927 - val_accuracy: 0.5144 - val_loss: 0.6925
Epoch 1/10
...
Epoch 9/10
5775/5775 ━━━━━━━━━━━━━━━━━━━━ 431s 75ms/step - accuracy: 0.8952 - loss: 0.2470 - val_accuracy: 0.8234 - val_loss: 0.4234
Epoch 10/10
5775/5775 ━━━━━━━━━━━━━━━━━━━━ 431s 75ms/step - accuracy: 0.9071 - loss: 0.2232 - val_accuracy: 0.8232 - val_loss: 0.4388
```