Deep Learning

Mini Project
On
MBTI Personality Classification with BiLSTM

By
Pratyush Mishra

INDEX

INTRODUCTION

This project aims to predict the personality types of individuals based on their social media posts using machine learning techniques. Specifically, the MBTI (Myers- Briggs Type Indicator) classification is used, which divides people into 16 different personality types, each representing combinations of four dichotomies:

- Extraversion vs. Introversion
- Sensing vs. Intuition
- Thinking vs. Feeling
- Judging vs. Perceiving

The objective of this project is to classify textual data (in the form of posts or statuses) into one of these 16 categories using a deep learning model. By utilizing a Bidirectional LSTM (BiLSTM) neural network, which is suited for sequential data like text, the model learns patterns within the posts to assign a personality type. This technique is well-suited for understanding the underlying patterns of communication and behavior found in text data.

OBJECTIVE

The main goals of the project are:

- Text Classification: The project will focus on the classification of textual data into one of the 16 MBTI personality types. Each individual's post will be categorized into a corresponding personality type.
- Preprocessing and Data Preparation: The raw text data needs to be cleaned, tokenized, and transformed into a format suitable for feeding into the model. This includes handling special characters, normalizing text, and converting labels into a numerical format.
- Model Development: Building a BiLSTM (Bidirectional Long Short-Term Memory) model that learns the sequence of words and can classify posts effectively. This model will be trained on a training set and validated on a separate test set to evaluate its performance.
- Model Evaluation: Once the model is trained, it is evaluated using metrics like accuracy, precision, recall, and F1-score to assess how well the model generalizes to unseen data.
- Model Improvement and Optimization: Based on the initial model's performance, we will experiment with various optimizations, such as changing the model architecture, tweaking hyperparameters, or employing techniques like early stopping to improve the overall accuracy and prevent overfitting.

TOOLS AND LIBRARIES REQUIRED

The project utilizes the following tools and libraries:

- Python: A high-level programming language known for its simplicity and versatility. It is widely used in data science and machine learning due to its vast ecosystem of libraries.
- TensorFlow: An open-source deep learning framework used to build and train neural networks. It provides high-level APIs such as Keras for easy model development and training.
- Keras: A high-level neural networks API written in Python, running on top of TensorFlow. It allows for quick prototyping and easy model building.
- NumPy: A library used for performing efficient mathematical operations and handling multi-dimensional arrays, which are essential in deep learning.
- Pandas: A library used for data manipulation and analysis. It allows loading, cleaning, and preprocessing large datasets in an efficient manner.
- Scikit-learn: A machine learning library for Python that provides easy-to-use tools for tasks such as data preprocessing (e.g., label encoding), splitting data into training and test sets, and evaluating models.
- Matplotlib & Seaborn: Visualization libraries that help in plotting data for a better understanding of model performance, including loss and accuracy curves during training.

DATASET AND PREPROCESSING

The dataset used in this project contains social media posts, which are labeled with their corresponding MBTI personality type. The preprocessing steps ensure that the raw data is transformed into a format suitable for input into the machine learning model. These steps include:

- Data Cleaning:
  The text data often contains irrelevant characters, such as special symbols, hyperlinks, and punctuations. A cleaning function is applied to remove these elements, normalize the text (convert it to lowercase), and ensure it is ready for tokenization.
- Tokenization:
  Tokenization involves converting the cleaned text into a sequence of words or tokens. This is achieved using a Tokenizer from the Keras library, which transforms words into numerical indices that can be fed into a neural network.
- Padding:
  To ensure that all input sequences are of the same length, padding is applied. This involves adding zeros at the end of shorter sequences until they reach the specified length (e.g., 300 words).
- Label Encoding:
  The labels (MBTI personality types) are encoded numerically, as deep learning models work with numerical data. The LabelEncoder from scikit-learn is used to convert the MBTI personality types into integer labels.

CODE IMPLEMENTATION

The code begins by importing the necessary libraries for data processing, model building, and evaluation. The dataset is loaded and cleaned, followed by the application of preprocessing techniques such as tokenization and padding. The core implementation includes building a BiLSTM model for text classification, compiling the model, and training it on the preprocessed data. The code also utilizes early stopping to prevent overfitting by monitoring the validation loss during training.

- Training: The model is trained with a dataset split into training and test sets. During training, we monitor the loss and accuracy to ensure the model is improving and generalizing well to unseen data.
- Evaluation: After training, the model is evaluated using various performance metrics to check its classification accuracy and ability to generalize to new data.

Code-

import pandas as pd import

numpy as np import tensor-

flow as tf

from tensorflow import keras

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout, Bidirectional

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad_sequences from

sklearn.utils import resample

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score im-

port matplotlib.pyplot as plt

import re

import seaborn as sns

```
import warnings


warnings.filterwarnings("ignore")


# Load dataset

file_path = "/Users/Pratyush/Documents/mbti_1.csv" df

= pd.read_csv(file_path)


# Text cleaning function def

clean_text(text):

    text = text.lower()

    text = re.sub(r'http\S+|www.\S+', '', text) text =

    re.sub(r'[^a-zA-Z ]', '', text)

    text = re.sub(r'\s+', ' ', text).strip() re-

    turn text


df['cleaned_posts'] = df['posts'].apply(clean_text)


# Encode MBTI labels label_encoder =

LabelEncoder()

df['label'] = label_encoder.fit_transform(df['type'])


# Balance the dataset

balanced_df = df.groupby('label', group_keys=False).apply(lambda x: resample(x, re-
place=True, n_samples=df['label'].value_counts().max(), random_state=42))
```

```
# Tokenization

tokenizer = Tokenizer(num_words=10000, oov_token="<OOV>") tokenizer.fit_on_-

texts(balanced_df['cleaned_posts'])

sequences = tokenizer.texts_to_sequences(balanced_df['cleaned_posts'])

padded_sequences = pad_sequences(sequences, maxlen=300, padding='post', truncat-
ing='post')


# Split data

X_train, X_test, y_train, y_test = train_test_split(padded_sequences,
balanced_df['label'], test_size=0.2, random_state=42, stratify=balanced_df['label'])


# Build neural network mod-

el = Sequential([

    Embedding(input_dim=10000, output_dim=128),

    Bidirectional(LSTM(64, return_sequences=True)),

    LSTM(32),

    Dropout(0.3),

    Dense(16, activation='relu'), Dense(len(label_encoder.-

    classes_), activation='softmax')

])


# Compile model

model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['ac-
curacy'])


# Train the model with early stopping

early_stopping = keras.callbacks.EarlyStopping(monitor='val_loss', patience=3,
restore_best_weights=True)
```

```
history = model.fit(X_train, y_train, epochs=20, batch_size=64,
validation_data=(X_test, y_test), callbacks=[early_stopping], verbose=1)


# Model evaluation

y_pred = np.argmax(model.predict(X_test), axis=1) ac-

curacy = accuracy_score(y_test, y_pred)

precision = precision_score(y_test, y_pred, average='weighted') recall =

recall_score(y_test, y_pred, average='weighted')

f1 = f1_score(y_test, y_pred, average='weighted')


# Display evaluation metrics metric-

s_table = f"""

<div style='font-size:18px; font-family:Arial; padding:12px; border:3px solid #FF9800;
background-color:#FFF3CD; color:#FF9800; font-weight:bold;'>

MODEL EVALUATION METRICS:<br>

<b>ACCURACY:</b> {accuracy:.4f}<br>

<b>PRECISION:</b> {precision:.4f}<br>

<b>RECALL:</b> {recall:.4f}<br>

<b>F1 SCORE:</b> {f1:.4f}<br>

</div>

"""

display(HTML(metrics_table))


# Plot training history plt.-

figure(figsize=(12, 5))

sns.set(style="whitegrid")
```

```
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy', color='blue') plt.plot(histo-
ry.history['val_accuracy'], label='Validation Accuracy', color='green') plt.xlabel('Epochs')
plt.ylabel('Accuracy') plt.leg-
end()
plt.title("Training vs Validation Accuracy")

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss', color='red') plt.plot(histo-
ry.history['val_loss'], label='Validation Loss', color='purple')
plt.xlabel('Epochs')
plt.ylabel('Loss') plt.legend()
plt.title("Training vs Validation Loss")

plt.show()

# Save trained model model.save("mbti_mod-
el.h5")
```

MODEL ARCHITECTURE AND TRAINING

The BiLSTM architecture is chosen for this project because it can learn both forward and backward relationships in the text, which is crucial for understanding the full context of sentences. Here's a brief explanation of the model components:

- Embedding Layer: The first layer is an Embedding layer, which maps words to dense vectors. This helps the model understand semantic relationships between words.
- Bidirectional LSTM Layer: A Bidirectional LSTM (BiLSTM) layer is used to process sequences from both directions (forward and backward), capturing context more effectively than a regular LSTM.
- Dense Layer: A fully connected layer that outputs the predictions for the MBTI personality types. The number of neurons in this layer is equal to the number of classes (16 MBTI types).
- Dropout Layer: A regularization technique that helps prevent overfitting by randomly dropping units during training.
- Softmax Activation: The final output layer uses a softmax activation function, which is typically used for multi-class classification problems.

Training involves feeding data through the model, calculating loss, and updating the weights using backpropagation. Adam optimizer is used for training, and early stopping ensures the model doesn't overfit by halting training if the validation performance stops improving.

MODEL EVALUATION

After training the model, its performance is assessed using various evaluation metrics:

- Accuracy: Measures the proportion of correct predictions.
- Precision: Indicates how many of the predicted positive labels were actually correct.
- Recall: Indicates how many of the actual positive labels were correctly predicted.
- F1-Score: The harmonic mean of precision and recall, giving a balance between the two.

These metrics help evaluate the overall performance and ability of the model to generalize on new data.

VISUALIZATION AND INSIGHTS, MODEL OUTPUTS AND OBSERVATIONS

Data visualization plays a key role in understanding how the model is performing:

- Accuracy vs. Epochs: This plot helps visualize how the model's accuracy improves over epochs.
- Loss vs. Epochs: This plot shows how the loss decreases over time, indicating that the model is learning.
- Confusion Matrix: This visualization shows how the model is performing across different classes, highlighting areas where the model is making errors.

These visualizations provide insights into the model's behavior and can help identify areas for improvement.

Once the model is trained and evaluated, the final predictions are compared to the true labels in the test set. This section discusses:

- Accuracy and Errors: The overall accuracy of the model and the types of errors it makes, such as misclassifications between similar personality types.
- Model Behavior: Analysis of which posts or personality types are harder for the model to classify, possibly due to ambiguities in the data or subtle language cues.

```
MODEL TRAINING IS STARTING...
- THE MODEL WILL GO THROUGH MULTIPLE EPOCHS.
- IT WILL OPTIMIZE WEIGHTS TO MINIMIZE LOSS.
- ACCURACY WILL IMPROVE OVER TIME.
- EARLY STOPPING IS USED TO PREVENT OVERFITTING.
```

```
Epoch 1/20
367/367 ━━━━━━━━━━━━━━━━━━━━ 376s 981ms/step - accuracy: 0.1651 - loss: 2.5073 - val_accuracy: 0.4963 - val_loss: 1.5642
Epoch 2/20
367/367 ━━━━━━━━━━━━━━━━━━━━ 500s 1s/step - accuracy: 0.5377 - loss: 1.3966 - val_accuracy: 0.6606 - val_loss: 1.0333
Epoch 3/20
367/367 ━━━━━━━━━━━━━━━━━━━━ 469s 1s/step - accuracy: 0.7182 - loss: 0.8595 - val_accuracy: 0.7573 - val_loss: 0.7638
Epoch 4/20
367/367 ━━━━━━━━━━━━━━━━━━━━ 408s 1s/step - accuracy: 0.8120 - loss: 0.5943 - val_accuracy: 0.7834 - val_loss: 0.7108
Epoch 5/20
367/367 ━━━━━━━━━━━━━━━━━━━━ 475s 1s/step - accuracy: 0.8654 - loss: 0.4495 - val_accuracy: 0.8444 - val_loss: 0.5987
Epoch 6/20
367/367 ━━━━━━━━━━━━━━━━━━━━ 420s 1s/step - accuracy: 0.9125 - loss: 0.3122 - val_accuracy: 0.8480 - val_loss: 0.6518
Epoch 7/20
367/367 ━━━━━━━━━━━━━━━━━━━━ 357s 974ms/step - accuracy: 0.9290 - loss: 0.2666 - val_accuracy: 0.8714 - val_loss: 0.5858
Epoch 8/20
367/367 ━━━━━━━━━━━━━━━━━━━━ 347s 945ms/step - accuracy: 0.9492 - loss: 0.1925 - val_accuracy: 0.8784 - val_loss: 0.6049
Epoch 9/20
367/367 ━━━━━━━━━━━━━━━━━━━━ 344s 936ms/step - accuracy: 0.9544 - loss: 0.1679 - val_accuracy: 0.8808 - val_loss: 0.6260
Epoch 10/20
367/367 ━━━━━━━━━━━━━━━━━━━━ 346s 943ms/step - accuracy: 0.9620 - loss: 0.1391 - val_accuracy: 0.8852 - val_loss: 0.5856
Epoch 11/20
367/367 ━━━━━━━━━━━━━━━━━━━━ 343s 933ms/step - accuracy: 0.9691 - loss: 0.1171 - val_accuracy: 0.8862 - val_loss: 0.6570
Epoch 12/20
367/367 ━━━━━━━━━━━━━━━━━━━━ 350s 954ms/step - accuracy: 0.9679 - loss: 0.1164 - val_accuracy: 0.8883 - val_loss: 0.6387
Epoch 13/20
367/367 ━━━━━━━━━━━━━━━━━━━━ 365s 908ms/step - accuracy: 0.9738 - loss: 0.0976 - val_accuracy: 0.8931 - val_loss: 0.6393
```

```
184/184 ━━━━━━━━━━━━━━━━━━━━ 16s 84ms/step
```

MODEL EVALUATION METRICS:
ACCURACY: 0.8852
PRECISION: 0.8837
RECALL: 0.8852
F1 SCORE: 0.8827