Max Liu

Prof. Hernandez

CISB 62

9 december 2023

Final Project PDF

Project Overview:

By leveraging both traditional machine learning, utilizing a Naive Bayes classifier, and deep learning, employing a Long Short-Term Memory (LSTM) neural network, the project aims to automatically assess the sentiment of user reviews as positive or negative.

NLP Methods:

Deep Learning Models: RNN, or Recurrent Neural Network

```
def create_lstm_model(embedding_dim=50, lstm_units=32):
    model = Sequential([
        Embedding(input_dim=max_words, output_dim=embedding_dim,
input_length=max_len_values[-1]),
        LSTM(lstm_units),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
    return model
```

The model consists of three layers. The first layer is an Embedding layer, which is responsible for converting input sequences into dense vectors of fixed size. It takes three parameters: input_dim, the size of the vocabulary (set to max_words), output_dim, the dimension of the dense embedding, and input_length, the length of input sequences (set to max_len). The second layer is an LSTM layer, which is a type of recurrent layer designed to capture sequential dependencies and patterns in the input data. Finally, there is a Dense layer with a single unit and a sigmoid activation function, serving as the output layer for binary classification (positive or negative sentiment).

Deep Learning Part 2: LSTM

```
model = Sequential([
    Embedding(input_dim=max_words, output_dim=50,
input_length=max_len),
    LSTM(32),
    Dense(1, activation='sigmoid')
])
```

I'm using the same code snippet here because the RNN I used explicitly included an LSTM Layer. In this snippet, the LSTM layer is added to the sequential model using the LSTM(lstm_units) line. The embedding_dim and lstm_units are parameters that control the dimensions of the embedding layer and the LSTM layer, respectively. The LSTM layer processes the sequences, capturing long-term dependencies in the input data, which is essential for tasks such as sentiment analysis.

Deep Learning Part 3: Callbacks : Modelcheckpoint and earlystopping

```
# Define early stopping and model checkpoint callbacks for the LSTM
```

```
model
early_stopping = EarlyStopping(monitor='val_loss', patience=3,
restore_best_weights=True)
model_checkpoint = ModelCheckpoint('best_lstm_model.h5',
save_best_only=True)

# Define the LSTM model with callbacks
def create_lstm_model(embedding_dim=50, lstm_units=32):
    model = Sequential([
        Embedding(input_dim=max_words, output_dim=embedding_dim,
input_length=max_len_values[-1]),
        LSTM(lstm_units),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
    return model

# Train the LSTM model with callbacks
X_train, X_test, y_train, y_test = train_test_split(padded_sequences,
labels, test_size=0.2, random_state=42)
lstm_model = create_lstm_model()
lstm_model.fit(X_train, y_train, epochs=10, validation_data=(X_test,
y_test), callbacks=[early_stopping, model_checkpoint])
```

EarlyStopping Callback:

The EarlyStopping callback is designed to monitor a specified metric during training, which is typically the validation loss. If the validation loss does not improve for a certain number of consecutive epochs (controlled by the patience parameter, set to 3 in this code), training is halted early. This helps prevent overfitting and saves computational resources by stopping the training process when further improvement is unlikely.

The EarlyStopping callback is created with the following parameters:

monitor='val_loss': It monitors the validation loss during training.

patience=3: Training will stop if there is no improvement in the validation loss for 3 consecutive epochs.

restore_best_weights=True: Restores the model weights from the epoch with the best performance on the validation set.

ModelCheckpoint Callback:

The ModelCheckpoint callback saves the model's weights to a file whenever there is an improvement in the monitored metric (in this case, validation loss). The saved model weights are useful for later retrieval, particularly when training over multiple epochs. The saved model file is named 'best_lstm_model.h5', and it ensures that the weights of the model at the epoch with the best performance on the validation set are stored.

The ModelCheckpoint callback is created with the following parameters:

'best_lstm_model.h5': Specifies the filename to save the best model weights.

save_best_only=True: Only saves the model weights when there is an improvement in the monitored metric (validation loss).

CONCLUSION:

In this project, a text classification task is undertaken using two distinct approaches: Naive Bayes and Long Short-Term Memory (LSTM) neural networks. The dataset, a subset of Amazon Fine Food Reviews, is preprocessed by handling missing values and introducing a feature representing the length of the text. The Naive Bayes model is trained using CountVectorizer, and

its accuracy is evaluated. For the LSTM model, a neural network architecture is defined with an embedding layer, an LSTM layer, and a dense layer. The training process incorporates callbacks, specifically EarlyStopping and ModelCheckpoint, to monitor and optimize the validation loss. The LSTM model is trained and evaluated on the dataset. Additionally, the best model weights are saved for future use.

Github Link:
https://github.com/NMOD-Max/CISB-62-Final

Data Source:
https://www.kaggle.com/datasets/snap/amazon-fine-food-reviews