ASSIGNMENT 8

IMDb Review Classification: Feedforward, CNN, RNN, LSTM

In this task, we are going to do sentiment classification on a movie review dataset. We are going to build a feedforward net, a convolutional neural net, a recurrent net and combine one or more of them to understand performance of each of them. A sentence can be thought of as a sequence of words that collectively represent meaning. Individual words impact the meaning. Thus, the context matters; words that occur earlier in the sentence influence the sentence's structure and meaning in the latter part of the sentence (e.g., Jose asked Anqi if she were going to the library today). Likewise, words that occur later in a sentence can affect the meaning of earlier words (e.g., Apple is an interesting company). As we have seen in lecture, if we wish to make use of a full sentence's context in both directions, then we should use a bi-directional RNN (e.g., Bi-LSTM). For the purpose of this tutorial, we are going to

restrict ourselves to only uni-directional RNNs.

!pip install --upgrade tensorflow import pandas as pd

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad_sequences from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, Dense, LSTM, SimpleRNN, Conv1D,

MaxPooling1D, Flatten, Dropout import numpy as np np.random.seed(1) vocabulary_size = 10000

max_review_length = 500

train_data = pd.read_csv("train.csv") test_data = pd.read_csv("test.csv")

X_train = train_data['review'] # Change 'review' to the name of your text column

y_train = train_data['sentiment'].map({'positive': 1, 'negative': 0}) # Map sentiment to 1/0

X_test = test_data['review']

y_test = test_data['sentiment'].map({'positive': 1, 'negative': 0})

tokenizer = Tokenizer(num_words=vocabulary_size) tokenizer.fit_on_texts(X_train)

X_train_sequences = tokenizer.texts_to_sequences(X_train)

X_test_sequences = tokenizer.texts_to_sequences(X_test)

X_train_padded = pad_sequences(X_train_sequences, maxlen=max_review_length) X_test_padded = pad_sequences(X_test_sequences, maxlen=max_review_length)

def build_feedforward_model(): model = Sequential()

model.add(Embedding(vocabulary_size, 32, input_length=max_review_length)) model.add(Flatten())

model.add(Dense(250, activation='relu')) model.add(Dropout(0.5)) model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

return model

def build_cnn_model(): model = Sequential()

model.add(Embedding(vocabulary_size, 32, input_length=max_review_length)) model.add(Conv1D(32, kernel_size=3, activation='relu')) model.add(MaxPooling1D(pool_size=2))

model.add(Flatten())

model.add(Dense(250, activation='relu')) model.add(Dropout(0.5)) model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

return model

def build_rnn_model(): model = Sequential()

model.add(Embedding(vocabulary_size, 32, input_length=max_review_length)) model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2)) model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

return model

max_features = vocabulary_size maxlen = max_review_length

def build_lstm_model():

model = Sequential()

```
Training Feedforward Model...
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument
  warnings.warn(
Epoch 1/5
391/391 - 44s - 113ms/step - accuracy: 0.7923 - loss: 0.4129 - val_accuracy: 0.8751 - val_loss: 0.295
Epoch 2/5
391/391 - 40s - 104ms/step - accuracy: 0.9550 - loss: 0.1279 - val_accuracy: 0.8562 - val_loss: 0.379
Epoch 3/5
391/391 - 40s - 103ms/step - accuracy: 0.9944 - loss: 0.0222 - val_accuracy: 0.8588 - val_loss: 0.506
Epoch 4/5
391/391 - 40s - 102ms/step - accuracy: 0.9996 - loss: 0.0034 - val_accuracy: 0.8548 - val_loss: 0.569
Epoch 5/5
391/391 - 43s - 111ms/step - accuracy: 0.9998 - loss: 0.0011 - val_accuracy: 0.8578 - val_loss: 0.612
782/782 - 10s - 13ms/step - accuracy: 0.8578 - loss: 0.6129
Test Accuracy: 0.8578
```

model.add(Embedding(max_features, 128, input_length=maxlen)) model.add(LSTM(128)) # LSTM layer

model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

return model

def train_and_evaluate_model(model, X_train, y_train, X_test, y_test): model.fit(X_train, y_train, epochs=5, batch_size=64,

validation_data=(X_test, y_test), verbose=2)

loss, accuracy = model.evaluate(X_test, y_test, verbose=2) print(f"Test Accuracy: {accuracy:.4f}")

print("Training Feedforward Model...") feedforward_model = build_feedforward_model()

train_and_evaluate_model(feedforward_model, X_train_padded, y_train, X_test_padded, y_test)

OUTPUT

print("\nTraining CNN Model...") cnn_model = build_cnn_model()

train_and_evaluate_model(cnn_model, X_train_padded, y_train, X_test_padded, y_test)

```
Training CNN Model...
Epoch 1/5
391/391 - 44s - 112ms/step - accuracy: 0.7844 - loss: 0.4092 - val_accuracy: 0.8902 - val_loss: 0.265
Epoch 2/5
391/391 - 39s - 100ms/step - accuracy: 0.9306 - loss: 0.1835 - val_accuracy: 0.8890 - val_loss: 0.274
Epoch 3/5
391/391 - 39s - 100ms/step - accuracy: 0.9659 - loss: 0.1005 - val_accuracy: 0.8796 - val_loss: 0.324
Epoch 4/5
391/391 - 36s - 91ms/step - accuracy: 0.9861 - loss: 0.0462 - val_accuracy: 0.8744 - val_loss: 0.4317
Epoch 5/5
391/391 - 41s - 106ms/step - accuracy: 0.9948 - loss: 0.0186 - val_accuracy: 0.8717 - val_loss: 0.558
782/782 - 9s - 12ms/step - accuracy: 0.8717 - loss: 0.5582
Test Accuracy: 0.8717
```

OUTPUT

print("\nTraining RNN Model...") rnn_model = build_rnn_model()

train_and_evaluate_model(rnn_model, X_train_padded, y_train, X_test_padded, y_test)

```
Training RNN Model...
Epoch 1/5
391/391 - 474s - 1s/step - accuracy: 0.7658 - loss: 0.4872 - val_accuracy: 0.8569 - val_loss: 0.3434
Epoch 2/5
391/391 - 473s - 1s/step - accuracy: 0.8589 - loss: 0.3436 - val_accuracy: 0.8367 - val_loss: 0.3789
Epoch 3/5
391/391 - 501s - 1s/step - accuracy: 0.8838 - loss: 0.2938 - val_accuracy: 0.7885 - val_loss: 0.4587
Epoch 4/5
391/391 - 510s - 1s/step - accuracy: 0.8925 - loss: 0.2797 - val_accuracy: 0.7416 - val_loss: 0.5209
Epoch 5/5
391/391 - 498s - 1s/step - accuracy: 0.8677 - loss: 0.3217 - val_accuracy: 0.8352 - val_loss: 0.4001
782/782 - 113s - 144ms/step - accuracy: 0.8352 - loss: 0.4001
Test Accuracy: 0.8352
```

OUTPUT

print("\nTraining LSTM Model...") lstm_model = build_lstm_model()

train_and_evaluate_model(lstm_model, X_train_padded, y_train, X_test_padded, y_test)

OUTPUT

```
Training LSTM Model...
Epoch 1/5
391/391 - 764s - 2s/step - accuracy: 0.7941 - loss: 0.4323 - val_accuracy: 0.8681 - val_loss: 0.3451
Epoch 2/5
391/391 - 792s - 2s/step - accuracy: 0.8897 - loss: 0.2729 - val_accuracy: 0.8793 - val_loss: 0.3050
Epoch 3/5
391/391 - 772s - 2s/step - accuracy: 0.9278 - loss: 0.1968 - val_accuracy: 0.8715 - val_loss: 0.3105
Epoch 4/5
391/391 - 772s - 2s/step - accuracy: 0.9454 - loss: 0.1505 - val_accuracy: 0.8571 - val_loss: 0.3478
Epoch 5/5
391/391 - 807s - 2s/step - accuracy: 0.9518 - loss: 0.1302 - val_accuracy: 0.8658 - val_loss: 0.4121
782/782 - 254s - 324ms/step - accuracy: 0.8658 - loss: 0.4121
Test Accuracy: 0.8658
```

BY: Tanishq Dublish

Roll NO.: 102117154
Group: CS6