Text Classification 2

Alekhya Pinnamaneni

The data set used in this notebook contains 25,000 reviews of various movies on IMDB and their corresponsing sentiment labels. Each email is either labeled/classified as positive or negative sentiment. The models created in this notebook should be able to predict whether each movie review in the test dataset is positive or negative.

In [ ]:
```
# Load data
import tensorflow as tf
import numpy as np
from tensorflow.keras import datasets, layers, models
(X_train, y_train), (X_test, y_test) = datasets.imdb.load_data(num_words=100
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-dat
asets/imdb.npz
17464789/17464789 [==============================] - 0s 0us/step
```

In [ ]:
```
# Define vectorizer function
def vectorizer(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results
```

In [ ]:
```
# Vectorize train and test data
X_train = vectorizer(X_train)
X_test = vectorizer(X_test)
```

In [ ]:
```
# Vectorize train and test labels
y_train = np.asarray(y_train).astype('float32')
y_test = np.asarray(y_test).astype('float32')
```

In [ ]:
```
# Create a validation dataset to use for training the models
X_val = X_train[:10000]
X_train = X_train[10000:]

y_val = y_train[:10000]
y_train = y_train[10000:]
```

Sequential Model

In [ ]:
```
# Create the sequential model
seq = models.Sequential()
seq.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
seq.add(layers.Dense(16, activation='relu'))
seq.add(layers.Dense(8, activation='relu'))
seq.add(layers.Dense(1, activation='sigmoid'))
```

In [ ]:
```
# Compile the model
seq.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accur
```

In [ ]:
```
# Train the sequential model on the train dataset
history = seq.fit(X_train, y_train, epochs=20, batch_size=512, validation_da
```

```
Epoch 1/20
30/30 [==============================] - 3s 62ms/step - loss: 0.6041 - accur
acy: 0.6549 - val_loss: 0.5342 - val_accuracy: 0.7997
Epoch 2/20
30/30 [==============================] - 1s 38ms/step - loss: 0.4894 - accur
acy: 0.8427 - val_loss: 0.5046 - val_accuracy: 0.7945
Epoch 3/20
30/30 [==============================] - 1s 33ms/step - loss: 0.4410 - accur
acy: 0.8905 - val_loss: 0.4772 - val_accuracy: 0.8534
Epoch 4/20
30/30 [==============================] - 1s 36ms/step - loss: 0.4103 - accur
acy: 0.9175 - val_loss: 0.5117 - val_accuracy: 0.8201
Epoch 5/20
30/30 [==============================] - 1s 34ms/step - loss: 0.3892 - accur
acy: 0.9321 - val_loss: 0.5010 - val_accuracy: 0.8430
Epoch 6/20
30/30 [==============================] - 1s 35ms/step - loss: 0.3668 - accur
acy: 0.9454 - val_loss: 0.4541 - val_accuracy: 0.8814
Epoch 7/20
30/30 [==============================] - 1s 34ms/step - loss: 0.3479 - accur
acy: 0.9558 - val_loss: 0.4613 - val_accuracy: 0.8739
Epoch 8/20
30/30 [==============================] - 1s 35ms/step - loss: 0.3350 - accur
acy: 0.9627 - val_loss: 0.5219 - val_accuracy: 0.8596
Epoch 9/20
30/30 [==============================] - 2s 57ms/step - loss: 0.3206 - accur
acy: 0.9678 - val_loss: 0.6146 - val_accuracy: 0.8343
Epoch 10/20
30/30 [==============================] - 2s 51ms/step - loss: 0.3098 - accur
acy: 0.9719 - val_loss: 0.5598 - val_accuracy: 0.8557
Epoch 11/20
30/30 [==============================] - 1s 35ms/step - loss: 0.2990 - accur
acy: 0.9751 - val_loss: 0.5567 - val_accuracy: 0.8617
Epoch 12/20
30/30 [==============================] - 1s 37ms/step - loss: 0.2887 - accur
acy: 0.9795 - val_loss: 0.5249 - val_accuracy: 0.8670
Epoch 13/20
30/30 [==============================] - 1s 34ms/step - loss: 0.2788 - accur
acy: 0.9819 - val_loss: 0.6543 - val_accuracy: 0.8503
```

```
Epoch 14/20
30/30 [==============================] - 1s 34ms/step - loss: 0.2734 - accur
acy: 0.9811 - val_loss: 0.5783 - val_accuracy: 0.8662
Epoch 15/20
30/30 [==============================] - 1s 42ms/step - loss: 0.2661 - accur
acy: 0.9825 - val_loss: 0.6283 - val_accuracy: 0.8618
Epoch 16/20
30/30 [==============================] - 1s 34ms/step - loss: 0.2554 - accur
acy: 0.9863 - val_loss: 0.6151 - val_accuracy: 0.8654
Epoch 17/20
30/30 [==============================] - 1s 36ms/step - loss: 0.2512 - accur
acy: 0.9859 - val_loss: 0.6349 - val_accuracy: 0.8639
Epoch 18/20
30/30 [==============================] - 1s 33ms/step - loss: 0.2461 - accur
acy: 0.9850 - val_loss: 0.6439 - val_accuracy: 0.8623
Epoch 19/20
30/30 [==============================] - 1s 34ms/step - loss: 0.2410 - accur
acy: 0.9852 - val_loss: 0.6378 - val_accuracy: 0.8631
Epoch 20/20
30/30 [==============================] - 1s 50ms/step - loss: 0.2325 - accur
acy: 0.9877 - val_loss: 0.6829 - val_accuracy: 0.8631
```

In [ ]:

```
# Use the model to predict on the test dataset
from sklearn.metrics import classification_report
pred = seq.predict(X_test)
pred = [1.0 if p>= 0.5 else 0.0 for p in pred]
```

```
782/782 [==============================] - 2s 2ms/step
```

In [ ]:

```
# Print the classification report
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

         0.0       0.83      0.89      0.86     12500
         1.0       0.88      0.81      0.85     12500

    accuracy                           0.85     25000
   macro avg       0.86      0.85      0.85     25000
weighted avg       0.86      0.85      0.85     25000
```

RNN Model

In [ ]:

```
# Create the RNN model
rnn = models.Sequential()
rnn.add(layers.Embedding(10000, 32))
rnn.add(layers.SimpleRNN(32))
rnn.add(layers.Dense(1, activation='sigmoid'))
```

In [ ]:
```
# Compile the model
rnn.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accur
```

In [ ]:
```
# Train the RNN model on the train dataset
history = rnn.fit(X_train, y_train, epochs=10, batch_size=128, validation_da
```

```
Epoch 1/10
94/94 [==============================] - 1525s 16s/step - loss: 0.6951 - acc
uracy: 0.5043 - val_loss: 0.6931 - val_accuracy: 0.5033
Epoch 2/10
94/94 [==============================] - 1535s 16s/step - loss: 0.7000 - acc
uracy: 0.5022 - val_loss: 0.6929 - val_accuracy: 0.5097
Epoch 3/10
94/94 [==============================] - 1492s 16s/step - loss: 0.6934 - acc
uracy: 0.5050 - val_loss: 0.6933 - val_accuracy: 0.5053
Epoch 4/10
94/94 [==============================] - 1454s 15s/step - loss: 0.6935 - acc
uracy: 0.5058 - val_loss: 0.6931 - val_accuracy: 0.5030
Epoch 5/10
77/94 [=======================>......] - ETA: 4:20 - loss: 0.6936 - accuracy
: 0.4953
```

In [ ]:
```
# Use the model to predict on the test dataset
from sklearn.metrics import classification_report
pred = rnn.predict(X_test)
pred = [1.0 if p>= 0.5 else 0.0 for p in pred]
```

In [ ]:
```
# Print the classification report
print(classification_report(y_test, pred))
```

Embedding Model

In [ ]:
```
# Create the model
embed = models.Sequential()
embed.add(layers.Embedding(max_features, 8, input_length=maxlen))
embed.add(layers.Flatten())
embed.add(layers.Dense(16, activation='relu'))
embed.add(layers.Dense(1, activation='sigmoid'))
```

In [ ]:
```
# Compile the model
embed.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc
```

In [ ]:
```
# Train the model on the train dataset
history = embed.fit(X_train, y_train, epochs=10, batch_size=32, validation_d
```

In [ ]:
```
# Use the model to predict on the test dataset
from sklearn.metrics import classification_report
pred = embed.predict(X_test)
pred = [1.0 if p>= 0.5 else 0.0 for p in pred]
```

In [ ]:
```
# Print the classification report
print(classification_report(y_test, pred))
```

Analysis

Out of the three models, the sequential model surpirisingly had the most accurate results on the test dataset out of the three models in this notebook. This could be because of the fairly simple nature of the IMDB dataset. Finding the sentiment of a movie review is not an extremely nuanced and complex task for a deep learning model, which is why the simple sequential model had better results than the more complex RNN and Embedding models.