IMDBClassifier

December 14, 2021

[1]:
```python
#Implementing the IMDB movie review classifier from Chapter 3.4 in Deep␣
↪Learning with Python

#Importing the necessary libraries
from keras.datasets import imdb
import numpy as np
from keras import models
from keras import layers
import matplotlib.pyplot as plt
```

[2]:
```python
#Importing and splitting the data into train and test data
(train_data, train_labels), (test_data, test_labels) = imdb.
↪load_data(num_words=10000)

#Looking at the data shape
print(train_data.shape)
print(test_data.shape)
```

```
(25000,)
(25000,)
```

[3]:
```python
#One-hot encoding the training and testing data
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results

x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

[4]:
```python
#Defining the layers of the neural network
model = models.Sequential()
model.add(layers.Dense(16, activation = 'relu', input_shape = (10000,)))
model.add(layers.Dense(16, activation = 'relu'))
```

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

[5]:
```
#Compiling the model and passing the optimizer, loss function, and required␣
↪metrics into the model
model.compile(optimizer = 'rmsprop', loss = 'binary_crossentropy', metrics =␣
↪['accuracy'])
```

[6]:
```
#Setting aside a validation dataset
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

[7]:
```
#Training the model with 20 epochs, a batch size of 512, and validating with␣
↪the validation data
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs = 20,
                    batch_size = 512,
                    validation_data = (x_val, y_val))
```

```
Epoch 1/20
30/30 [==============================] - 1s 39ms/step - loss: 0.5237 - accuracy:
0.7725 - val_loss: 0.3811 - val_accuracy: 0.8702
Epoch 2/20
30/30 [==============================] - 1s 38ms/step - loss: 0.3043 - accuracy:
0.9019 - val_loss: 0.2987 - val_accuracy: 0.8882
Epoch 3/20
30/30 [==============================] - 2s 51ms/step - loss: 0.2186 - accuracy:
0.9295 - val_loss: 0.2804 - val_accuracy: 0.8874
Epoch 4/20
30/30 [==============================] - 1s 32ms/step - loss: 0.1708 - accuracy:
0.9459 - val_loss: 0.2917 - val_accuracy: 0.8825
Epoch 5/20
30/30 [==============================] - 1s 32ms/step - loss: 0.1401 - accuracy:
0.9545 - val_loss: 0.2902 - val_accuracy: 0.8862
Epoch 6/20
30/30 [==============================] - 1s 26ms/step - loss: 0.1146 - accuracy:
0.9629 - val_loss: 0.3179 - val_accuracy: 0.8822
Epoch 7/20
30/30 [==============================] - 1s 27ms/step - loss: 0.0897 - accuracy:
0.9743 - val_loss: 0.3210 - val_accuracy: 0.8835
Epoch 8/20
30/30 [==============================] - 1s 27ms/step - loss: 0.0743 - accuracy:
0.9801 - val_loss: 0.3353 - val_accuracy: 0.8823
Epoch 9/20
30/30 [==============================] - 1s 27ms/step - loss: 0.0611 - accuracy:
```

```
0.9833 - val_loss: 0.3700 - val_accuracy: 0.8800
Epoch 10/20
30/30 [==============================] - 1s 27ms/step - loss: 0.0514 - accuracy:
0.9871 - val_loss: 0.3848 - val_accuracy: 0.8794
Epoch 11/20
30/30 [==============================] - 1s 30ms/step - loss: 0.0400 - accuracy:
0.9908 - val_loss: 0.4275 - val_accuracy: 0.8771
Epoch 12/20
30/30 [==============================] - 1s 28ms/step - loss: 0.0286 - accuracy:
0.9945 - val_loss: 0.4498 - val_accuracy: 0.8725
Epoch 13/20
30/30 [==============================] - 1s 38ms/step - loss: 0.0245 - accuracy:
0.9953 - val_loss: 0.4900 - val_accuracy: 0.8738
Epoch 14/20
30/30 [==============================] - 2s 54ms/step - loss: 0.0193 - accuracy:
0.9965 - val_loss: 0.5133 - val_accuracy: 0.8705
Epoch 15/20
30/30 [==============================] - 1s 35ms/step - loss: 0.0135 - accuracy:
0.9986 - val_loss: 0.5606 - val_accuracy: 0.8664
Epoch 16/20
30/30 [==============================] - 1s 27ms/step - loss: 0.0100 - accuracy:
0.9996 - val_loss: 0.7403 - val_accuracy: 0.8511
Epoch 17/20
30/30 [==============================] - 1s 31ms/step - loss: 0.0084 - accuracy:
0.9988 - val_loss: 0.6209 - val_accuracy: 0.8681
Epoch 18/20
30/30 [==============================] - 1s 31ms/step - loss: 0.0103 - accuracy:
0.9979 - val_loss: 0.6545 - val_accuracy: 0.8674
Epoch 19/20
30/30 [==============================] - 1s 28ms/step - loss: 0.0035 - accuracy:
0.9999 - val_loss: 0.6872 - val_accuracy: 0.8671
Epoch 20/20
30/30 [==============================] - 1s 30ms/step - loss: 0.0064 - accuracy:
0.9987 - val_loss: 0.7188 - val_accuracy: 0.8664
```

[8]:

```
#Obtaining the training history and viewing its keys
history_dict = history.history
history_dict.keys()
```

[8]:

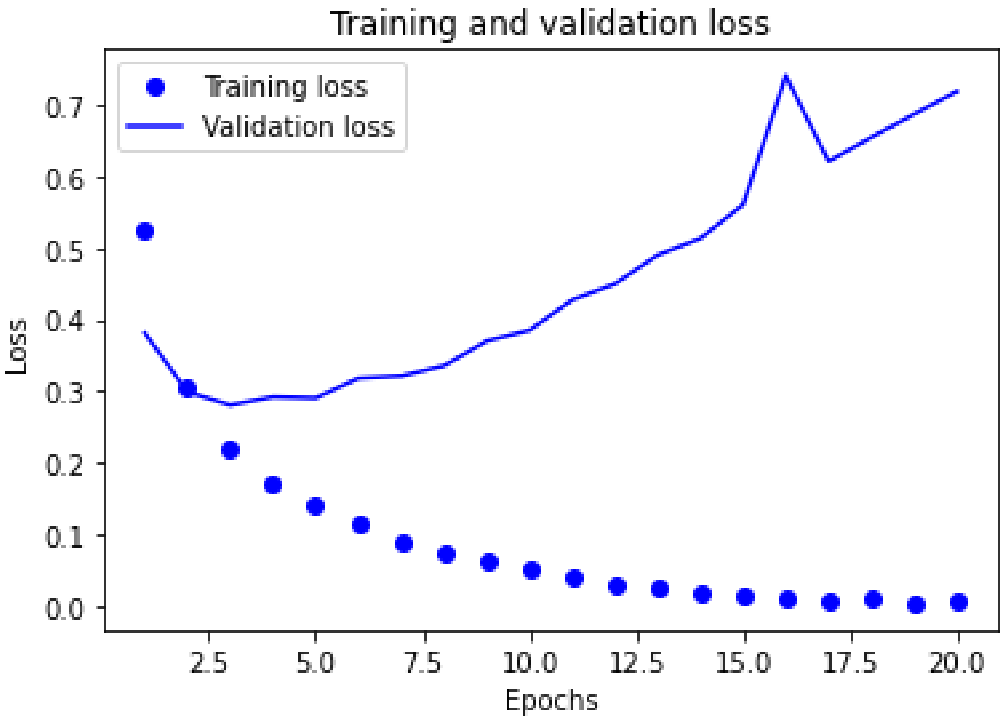
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

[9]:

```
#Plotting the training loss and the validation loss for each epoch

loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1, len(loss_values) + 1)
```

```python
plt.plot(epochs, loss_values, 'bo', label = 'Training loss')
plt.plot(epochs, val_loss_values, 'b', label = 'Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

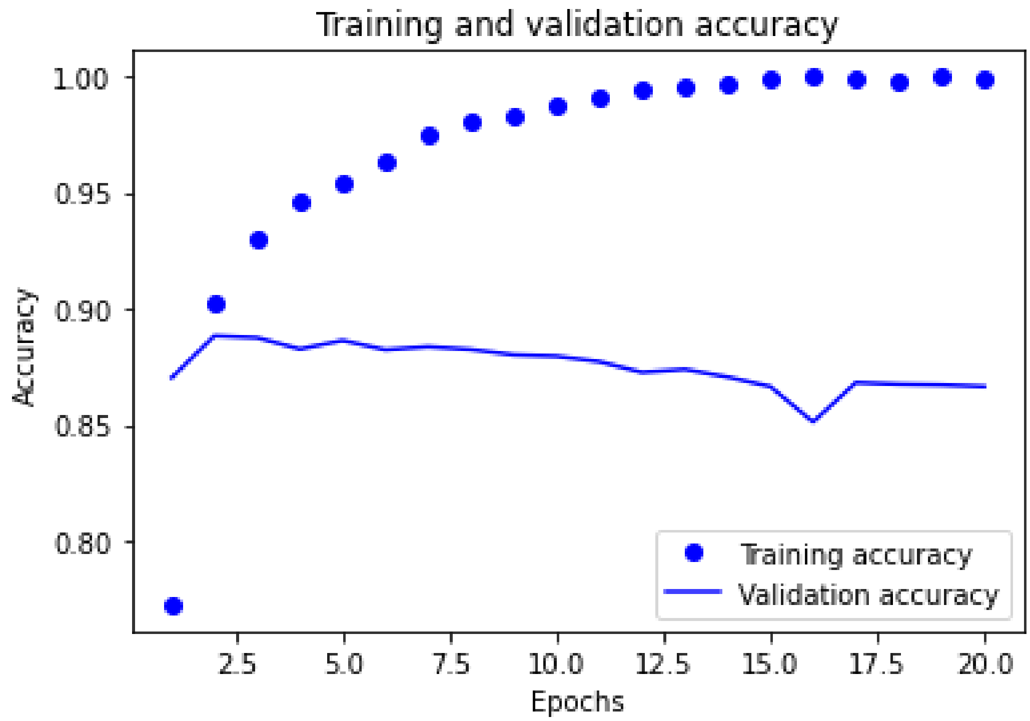


[10]:
```python
#Plotting the training accuracy and the validation accuracy for each epoch

plt.clf()

acc_values = history_dict['accuracy']
val_acc_values = history_dict['val_accuracy']

plt.plot(epochs, acc_values, 'bo', label = 'Training accuracy')
plt.plot(epochs, val_acc_values, 'b', label = 'Validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



[11]:
```
#Evaluating the model on the test dataset
results = model.evaluate(x_test, y_test)
print('The loss is {} ({} epochs)'.format(round(results[0], 4), len(epochs)))
print('The accuracy is {} percent ({} epochs)'.format(round(results[1], 4)*100,␣
↪len(epochs)))
```

```
782/782 [==============================] - 2s 2ms/step - loss: 0.7851 -
accuracy: 0.8513
The loss is 0.7851 (20 epochs)
The accuracy is 85.13 percent (20 epochs)
```

[12]:
```
#Retraining the model, this time with 4 epochs, and viewing its loss and␣
↪accuracy

#Defining the layers
model = models.Sequential()
model.add(layers.Dense(16, activation = 'relu', input_shape = (10000,)))
model.add(layers.Dense(16, activation = 'relu'))
model.add(layers.Dense(1, activation='sigmoid'))

#Defining the optimizer, loss function, and metrics
model.compile(optimizer = 'rmsprop', loss = 'binary_crossentropy', metrics =␣
↪['accuracy'])
```

```
#Fitting the model with the training data, 4 epochs, and a batch size of 512
model.fit(x_train, y_train, epochs = 4, batch_size = 512)
```

```
Epoch 1/4
49/49 [==============================] - 0s 9ms/step - loss: 0.4490 - accuracy:
0.8247
Epoch 2/4
49/49 [==============================] - 0s 9ms/step - loss: 0.2608 - accuracy:
0.9100
Epoch 3/4
49/49 [==============================] - 0s 8ms/step - loss: 0.2016 - accuracy:
0.9288
Epoch 4/4
49/49 [==============================] - 0s 7ms/step - loss: 0.1698 - accuracy:
0.9394
```

[12]:
```
<tensorflow.python.keras.callbacks.History at 0x7faf03ff2490>
```

[13]:
```
#Obtaining the results
results = model.evaluate(x_test, y_test)
print('The loss is {} ({} epochs)'.format(round(results[0], 4), 4))
print('The accuracy is {} percent ({} epochs)'.format(round(results[1], 4)*100,␣
 ↪4))
```

```
782/782 [==============================] - 2s 2ms/step - loss: 0.2910 -
accuracy: 0.8839
The loss is 0.291 (4 epochs)
The accuracy is 88.39 percent (4 epochs)
```

[14]:
```
#Generating predictions using the test data
#Generating the likelihood of movie reviews being positive
model.predict(x_test)
```

[14]:
```
array([[0.20587534],
       [0.99967194],
       [0.88956904],
       …,
       [0.13893089],
       [0.07124329],
       [0.6224241 ]], dtype=float32)
```