Assignment 1 : Advanced Machine Learnig

by- Vaishnavi Haripuri

student ID- 811285838

Email- vharipur@kent.edu

Importing data

```
from tensorflow.keras.datasets import imdb
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt

(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

Printing data

```
print(train_data[0])
print(train_labels[0])
[1, 14, 22, 16, ..., 178, 32]
1
print(train_data[0][:10])
# Print the first 10 elements
```

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670,
1
```

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65]
```

```
max([max(sequence) for sequence in train_data])
```

```
9999
```

Preparing 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
```

Division of training data and test data

Vectorizing

```
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
```

converting labels to floats

```
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

Taking validation set out

```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

Hypertuning as per conditions

Using 1 hidden layer

using more hidden units

using mse loss function

using tanh activation

Building network

```
model = keras.Sequential([
    layers.Dense(64, activation='tanh', input_shape=(10000,)),
    # One hidden layer
    layers.Dense(1, activation='sigmoid')
])
```

Compilation

```
model.compile(optimizer="rmsprop",
              loss="mse",
              metrics=["accuracy"])
```

Training model

```
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val),
                    verbose=1)
```

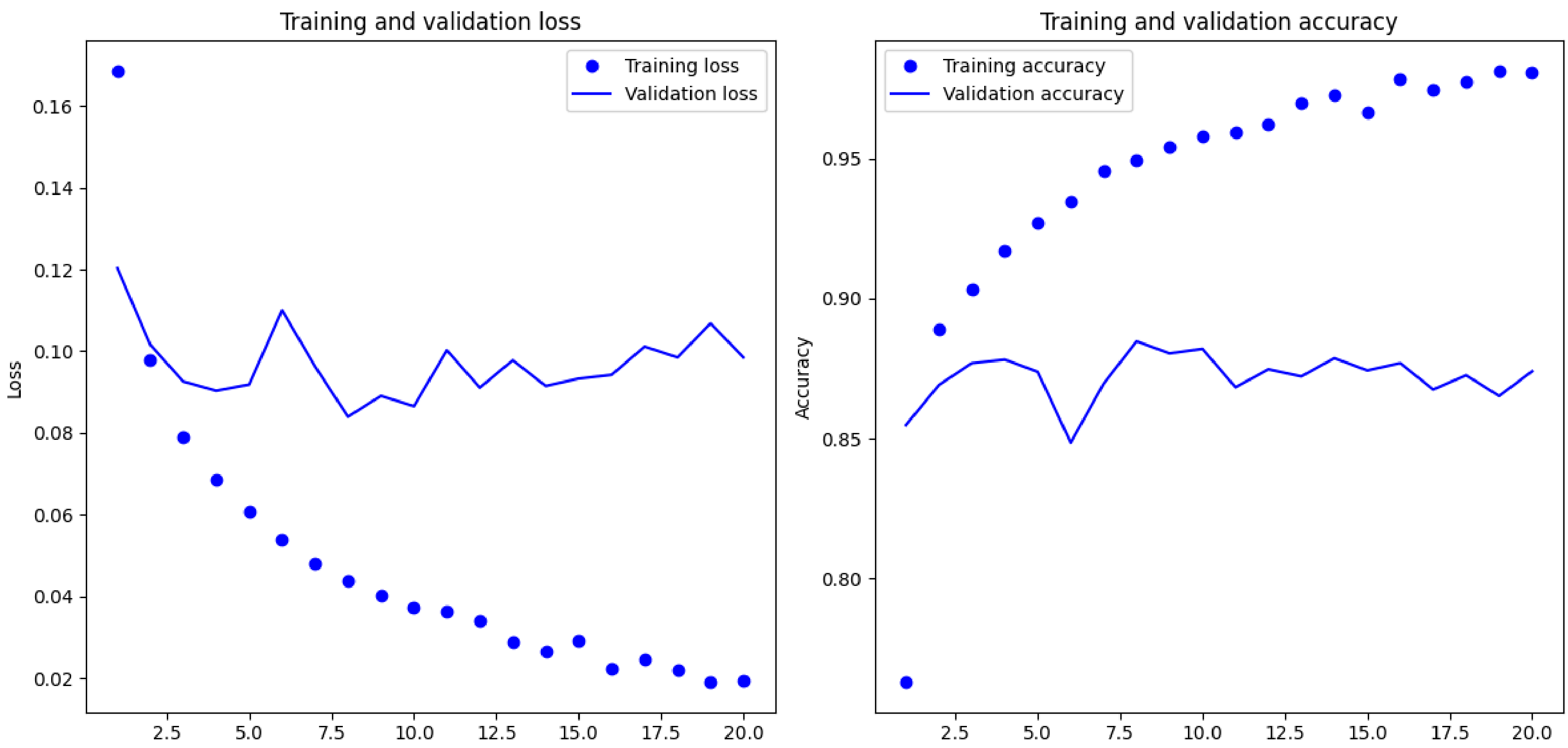
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 129ms/step - accuracy: 0.6632 - loss: 0.2053 - val_accuracy: 0.8548 - val_loss: 0.1203
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 65ms/step - accuracy: 0.8939 - loss: 0.0993 - val_accuracy: 0.8691 - val_loss: 0.1016
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 71ms/step - accuracy: 0.9007 - loss: 0.0822 - val_accuracy: 0.8769 - val_loss: 0.0926
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 73ms/step - accuracy: 0.9180 - loss: 0.0687 - val_accuracy: 0.8782 - val_loss: 0.0903
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 121ms/step - accuracy: 0.9356 - loss: 0.0573 - val_accuracy: 0.8738 - val_loss: 0.0918
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 66ms/step - accuracy: 0.9395 - loss: 0.0524 - val_accuracy: 0.8485 - val_loss: 0.1100
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 79ms/step - accuracy: 0.9433 - loss: 0.0498 - val_accuracy: 0.8696 - val_loss: 0.0963
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 66ms/step - accuracy: 0.9476 - loss: 0.0444 - val_accuracy: 0.8847 - val_loss: 0.0840
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 84ms/step - accuracy: 0.9555 - loss: 0.0396 - val_accuracy: 0.8804 - val_loss: 0.0891
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 76ms/step - accuracy: 0.9566 - loss: 0.0382 - val_accuracy: 0.8820 - val_loss: 0.0865
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 70ms/step - accuracy: 0.9606 - loss: 0.0357 - val_accuracy: 0.8683 - val_loss: 0.1002
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 63ms/step - accuracy: 0.9658 - loss: 0.0326 - val_accuracy: 0.8747 - val_loss: 0.0911
Epoch 13/20
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 98ms/step - accuracy: 0.9758 - loss: 0.0255 - val_accuracy: 0.8722 - val_loss: 0.0978
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 62ms/step - accuracy: 0.9715 - loss: 0.0269 - val_accuracy: 0.8787 - val_loss: 0.0915
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 82ms/step - accuracy: 0.9671 - loss: 0.0287 - val_accuracy: 0.8743 - val_loss: 0.0933
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 80ms/step - accuracy: 0.9820 - loss: 0.0201 - val_accuracy: 0.8769 - val_loss: 0.0942
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 119ms/step - accuracy: 0.9756 - loss: 0.0242 - val_accuracy: 0.8675 - val_loss: 0.1011
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 88ms/step - accuracy: 0.9827 - loss: 0.0189 - val_accuracy: 0.8726 - val_loss: 0.0985
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 77ms/step - accuracy: 0.9847 - loss: 0.0168 - val_accuracy: 0.8652 - val_loss: 0.1068
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 75ms/step - accuracy: 0.9844 - loss: 0.0174 - val_accuracy: 0.8740 - val_loss: 0.0985
```

Plotting loss and accuracy scores

```
history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
accuracy = history_dict['accuracy']
val_accuracy = history_dict['val_accuracy']

epochs = range(1, len(loss_values) + 1)


plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
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.subplot(1, 2, 2)
plt.plot(epochs, accuracy, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```



Double-click (or enter) to edit

```
predictions = model.predict(x_test)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step
```

Double-click (or enter) to edit

```
binary_predictions = (predictions > 0.5).astype(int)
```

Double-click (or enter) to edit

```
print("Predictions vs Actual:")
for i in range(5):
    print(f"Predicted: {binary_predictions[i][0]}, Actual: {y_test[i]}")
```

```
Predictions vs Actual:
Predicted: 0, Actual: 0.0
Predicted: 1, Actual: 1.0
Predicted: 0, Actual: 1.0
Predicted: 1, Actual: 0.0
Predicted: 0, Actual: 1.0
```

Adding dropout

```
model_dropout = keras.Sequential([
    layers.Dense(64, activation='tanh', input_shape=(10000,)),
    layers.Dropout(0.5),  # 50% dropout
    layers.Dense(1, activation='sigmoid')
])
```

```
model_dropout.compile(optimizer='rmsprop', loss='mse', metrics=['accuracy'])
```

```
history_dropout = model_dropout.fit(partial_x_train,
                                    partial_y_train,
                                    epochs=20,
                                    batch_size=512,
                                    validation_data=(x_val, y_val),
                                    verbose=1)
```

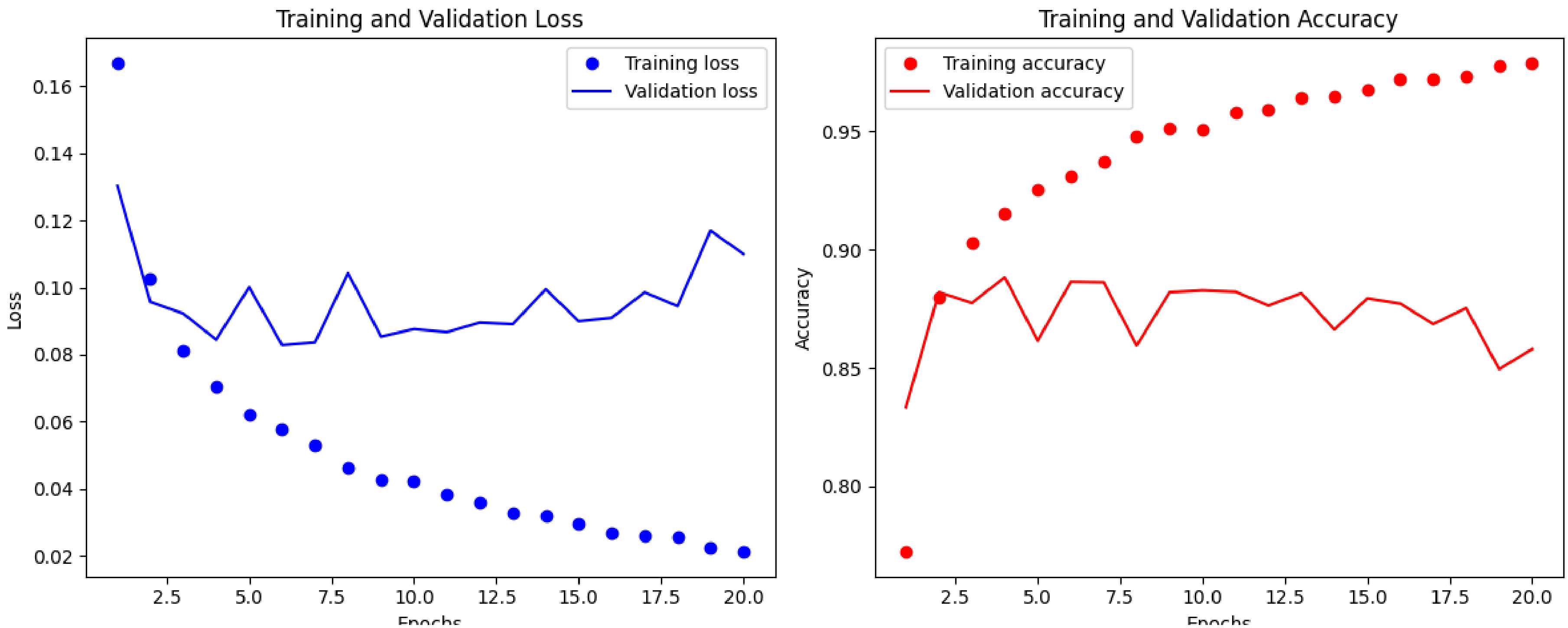
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 151ms/step - accuracy: 0.6838 - loss: 0.2010 - val_accuracy: 0.8335 - val_loss: 0.1303
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 80ms/step - accuracy: 0.8777 - loss: 0.1068 - val_accuracy: 0.8820 - val_loss: 0.0957
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 72ms/step - accuracy: 0.9056 - loss: 0.0820 - val_accuracy: 0.8775 - val_loss: 0.0922
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 120ms/step - accuracy: 0.9174 - loss: 0.0707 - val_accuracy: 0.8883 - val_loss: 0.0845
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 6s 193ms/step - accuracy: 0.9284 - loss: 0.0610 - val_accuracy: 0.8615 - val_loss: 0.1001
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 6s 65ms/step - accuracy: 0.9273 - loss: 0.0593 - val_accuracy: 0.8864 - val_loss: 0.0829
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 77ms/step - accuracy: 0.9448 - loss: 0.0500 - val_accuracy: 0.8862 - val_loss: 0.0837
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 96ms/step - accuracy: 0.9501 - loss: 0.0461 - val_accuracy: 0.8595 - val_loss: 0.1043
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 114ms/step - accuracy: 0.9471 - loss: 0.0451 - val_accuracy: 0.8820 - val_loss: 0.0854
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 67ms/step - accuracy: 0.9516 - loss: 0.0418 - val_accuracy: 0.8828 - val_loss: 0.0876
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 73ms/step - accuracy: 0.9555 - loss: 0.0393 - val_accuracy: 0.8822 - val_loss: 0.0867
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 76ms/step - accuracy: 0.9589 - loss: 0.0358 - val_accuracy: 0.8764 - val_loss: 0.0896
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 132ms/step - accuracy: 0.9637 - loss: 0.0329 - val_accuracy: 0.8816 - val_loss: 0.0891
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 80ms/step - accuracy: 0.9623 - loss: 0.0333 - val_accuracy: 0.8662 - val_loss: 0.0995
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - accuracy: 0.9650 - loss: 0.0300 - val_accuracy: 0.8794 - val_loss: 0.0900
Epoch 16/20
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 68ms/step - accuracy: 0.9760 - loss: 0.0249 - val_accuracy: 0.8772 - val_loss: 0.0910
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 79ms/step - accuracy: 0.9753 - loss: 0.0235 - val_accuracy: 0.8686 - val_loss: 0.0986
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 102ms/step - accuracy: 0.9758 - loss: 0.0245 - val_accuracy: 0.8753 - val_loss: 0.0945
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 82ms/step - accuracy: 0.9806 - loss: 0.0204 - val_accuracy: 0.8495 - val_loss: 0.1169
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 85ms/step - accuracy: 0.9746 - loss: 0.0239 - val_accuracy: 0.8580 - val_loss: 0.1100
```

Plotting after adding dropout

```
# Retrieve the history of loss and accuracy from the training process
history_dict = history_dropout.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
accuracy = history_dict['accuracy']
val_accuracy = history_dict['val_accuracy']

epochs = range(1, len(loss_values) + 1)

# Plotting the loss and accuracy

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)  # 1 row, 2 columns, plot 1
plt.plot(epochs, loss_values, 'bo', label='Training loss')  # 'bo' is for blue dot
plt.plot(epochs, val_loss_values, 'b', label='Validation loss')  # 'b' is for solid blue line
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()


plt.subplot(1, 2, 2)  # 1 row, 2 columns, plot 2
plt.plot(epochs, accuracy, 'ro', label='Training accuracy')  # 'ro' is for red dot
```

```
plt.plot(epochs, val_accuracy, 'r', label='Validation accuracy')  # 'r' is for solid red line
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()  # Adjusts subplot spacing to prevent overlap
plt.show()
```



Evaluate the model on the test set

```
results = model_dropout.evaluate(x_test, y_test)
print("Test Loss, Test Accuracy:", results)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8464 - loss: 0.1201
Test Loss, Test Accuracy: [0.11903790384531021, 0.8479599952697754]
```

Make predictions on the test set

```
predictions = model_dropout.predict(x_test)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step
```

Convert probabilities to binary predictions (0 or 1)

```
predicted_classes = (predictions > 0.5).astype(int)
```

Print the first few predictions and their corresponding actual labels

```
for i in range(5):
    print(f"Predicted: {predicted_classes[i][0]}, Actual: {y_test[i]}")
```

```
Predicted: 0, Actual: 0.0
Predicted: 1, Actual: 1.0
Predicted: 1, Actual: 1.0
Predicted: 1, Actual: 0.0
Predicted: 1, Actual: 1.0
```

Double-click (or enter) to edit

Predicted: 1 means the model predicts a positive review.

Predicted: 0 means the model predicts a negative review.

Double-click (or enter) to edit