Bank customer

October 9, 2024

[35]:
```
import pandas as pd
```

[37]:
```
#Step 1: Read the Dataset - Load the dataset
```

[51]:
```
df = pd.read_csv('/home/pc13/Downloads/archive/Churn_Modelling.csv')
```

[50]:
```
df.shape
```

[50]:
```
(10000, 14)
```

[52]:
```
# Display the first few rows of the dataset
print(df.head())
```

```
   RowNumber  CustomerId   Surname  CreditScore Geography  Gender  Age  \
0          1    15634602  Hargrave          619    France  Female   42
1          2    15647311      Hill          608     Spain  Female   41
2          3    15619304      Onio          502    France  Female   42
3          4    15701354      Boni          699    France  Female   39
4          5    15737888  Mitchell          850     Spain  Female   43

   Tenure    Balance  NumOfProducts  HasCrCard  IsActiveMember  \
0       2       0.00              1          1               1
1       1   83807.86              1          0               1
2       8  159660.80              3          1               0
3       1       0.00              2          0               0
4       2  125510.82              1          1               1

   EstimatedSalary  Exited
0        101348.88       1
1        112542.58       0
2        113931.57       1
3         93826.63       0
4         79084.10       0
```

[ ]:
```
#Step 2: Distinguish Feature and Target Set
```

[39]:
```python
from sklearn.model_selection import train_test_split

# Define features and target
X = df.drop(columns=['Exited', 'CustomerId'])  # 'Exited' is the target variable
y = df['Exited']

# Convert categorical variables to dummy/indicator variables
X = pd.get_dummies(X, drop_first=True)

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Output the shape of the datasets
print(f"Training set shape: {X_train.shape}, Test set shape: {X_test.shape}")
```

```
Training set shape: (8000, 2943), Test set shape: (2000, 2943)
```

[40]:
```python
#Step 3: Normalize the Train and Test Data
```

[41]:
```python
from sklearn.preprocessing import StandardScaler

# Initialize the StandardScaler
scaler = StandardScaler()

# Fit and transform the training data, transform the test data
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Output the mean and variance of the scaled training data
print(f"Mean of X_train: {X_train.mean(axis=0)}, Variance of X_train: {X_train.
↪var(axis=0)}")
```

```
Mean of X_train: [-1.59872116e-17  5.43565193e-16 -1.89626093e-16 …
-4.26325641e-17
 -7.19424520e-17  2.84217094e-17], Variance of X_train: [1. 1. 1. … 1. 1. 1.]
```

[42]:
```python
#Step 4: Initialize and Build the Model
```

[43]:
```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Build the model
model = keras.Sequential([
    layers.Dense(32, activation='relu', input_shape=(X_train.shape[1],)),
    layers.Dense(16, activation='relu'),
```

```
    layers.Dense(1, activation='sigmoid')  # For binary classification
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32,␣
 ↪validation_split=0.2)

# Output training history
print(history.history)
```

```
/home/pc13/miniconda3/lib/python3.12/site-
packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
2024-10-09 13:36:57.898555: W
external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
75340800 exceeds 10% of free system memory.
```

```
Epoch 1/50
200/200              1s 3ms/step -
accuracy: 0.7058 - loss: 0.6054 - val_accuracy: 0.7944 - val_loss: 0.5141
Epoch 2/50
200/200              0s 2ms/step -
accuracy: 0.8088 - loss: 0.4159 - val_accuracy: 0.7713 - val_loss: 0.5427
Epoch 3/50
200/200              0s 2ms/step -
accuracy: 0.8347 - loss: 0.3509 - val_accuracy: 0.7531 - val_loss: 0.5723
Epoch 4/50
200/200              0s 2ms/step -
accuracy: 0.8639 - loss: 0.3021 - val_accuracy: 0.7369 - val_loss: 0.6044
Epoch 5/50
200/200              0s 2ms/step -
accuracy: 0.8704 - loss: 0.2827 - val_accuracy: 0.7287 - val_loss: 0.6332
Epoch 6/50
200/200              0s 2ms/step -
accuracy: 0.8831 - loss: 0.2568 - val_accuracy: 0.7181 - val_loss: 0.6699
Epoch 7/50
200/200              0s 2ms/step -
accuracy: 0.8852 - loss: 0.2541 - val_accuracy: 0.7175 - val_loss: 0.7098
Epoch 8/50
200/200              0s 2ms/step -
accuracy: 0.8959 - loss: 0.2343 - val_accuracy: 0.7244 - val_loss: 0.7259
Epoch 9/50
```

```
200/200            0s 2ms/step -
accuracy: 0.8968 - loss: 0.2255 - val_accuracy: 0.7119 - val_loss: 0.7891
Epoch 10/50
200/200            0s 2ms/step -
accuracy: 0.9078 - loss: 0.2051 - val_accuracy: 0.7169 - val_loss: 0.7816
Epoch 11/50
200/200            1s 3ms/step -
accuracy: 0.9125 - loss: 0.1910 - val_accuracy: 0.7119 - val_loss: 0.8578
Epoch 12/50
200/200            1s 2ms/step -
accuracy: 0.9224 - loss: 0.1759 - val_accuracy: 0.7144 - val_loss: 0.8983
Epoch 13/50
200/200            0s 2ms/step -
accuracy: 0.9317 - loss: 0.1634 - val_accuracy: 0.7206 - val_loss: 0.9857
Epoch 14/50
200/200            0s 2ms/step -
accuracy: 0.9431 - loss: 0.1363 - val_accuracy: 0.7169 - val_loss: 1.0089
Epoch 15/50
200/200            0s 2ms/step -
accuracy: 0.9505 - loss: 0.1248 - val_accuracy: 0.7262 - val_loss: 1.0446
Epoch 16/50
200/200            0s 2ms/step -
accuracy: 0.9572 - loss: 0.1087 - val_accuracy: 0.7337 - val_loss: 1.1106
Epoch 17/50
200/200            0s 2ms/step -
accuracy: 0.9613 - loss: 0.0965 - val_accuracy: 0.7269 - val_loss: 1.1864
Epoch 18/50
200/200            0s 2ms/step -
accuracy: 0.9681 - loss: 0.0858 - val_accuracy: 0.7306 - val_loss: 1.2437
Epoch 19/50
200/200            0s 2ms/step -
accuracy: 0.9740 - loss: 0.0753 - val_accuracy: 0.7275 - val_loss: 1.3100
Epoch 20/50
200/200            0s 2ms/step -
accuracy: 0.9775 - loss: 0.0678 - val_accuracy: 0.7312 - val_loss: 1.3680
Epoch 21/50
200/200            1s 3ms/step -
accuracy: 0.9803 - loss: 0.0566 - val_accuracy: 0.7344 - val_loss: 1.4062
Epoch 22/50
200/200            0s 2ms/step -
accuracy: 0.9859 - loss: 0.0445 - val_accuracy: 0.7375 - val_loss: 1.4660
Epoch 23/50
200/200            0s 2ms/step -
accuracy: 0.9879 - loss: 0.0416 - val_accuracy: 0.7331 - val_loss: 1.5544
Epoch 24/50
200/200            0s 2ms/step -
accuracy: 0.9897 - loss: 0.0339 - val_accuracy: 0.7369 - val_loss: 1.6068
Epoch 25/50
```

```
200/200              0s 2ms/step -
accuracy: 0.9919 - loss: 0.0298 - val_accuracy: 0.7337 - val_loss: 1.6703
Epoch 26/50
200/200              0s 2ms/step -
accuracy: 0.9929 - loss: 0.0315 - val_accuracy: 0.7375 - val_loss: 1.7006
Epoch 27/50
200/200              0s 2ms/step -
accuracy: 0.9940 - loss: 0.0227 - val_accuracy: 0.7369 - val_loss: 1.7706
Epoch 28/50
200/200              0s 2ms/step -
accuracy: 0.9931 - loss: 0.0262 - val_accuracy: 0.7412 - val_loss: 1.8225
Epoch 29/50
200/200              0s 2ms/step -
accuracy: 0.9967 - loss: 0.0178 - val_accuracy: 0.7394 - val_loss: 1.8540
Epoch 30/50
200/200              0s 2ms/step -
accuracy: 0.9967 - loss: 0.0179 - val_accuracy: 0.7437 - val_loss: 1.9330
Epoch 31/50
200/200              0s 2ms/step -
accuracy: 0.9971 - loss: 0.0161 - val_accuracy: 0.7400 - val_loss: 1.9963
Epoch 32/50
200/200              1s 3ms/step -
accuracy: 0.9964 - loss: 0.0142 - val_accuracy: 0.7425 - val_loss: 2.0427
Epoch 33/50
200/200              1s 3ms/step -
accuracy: 0.9967 - loss: 0.0127 - val_accuracy: 0.7381 - val_loss: 2.0862
Epoch 34/50
200/200              1s 3ms/step -
accuracy: 0.9969 - loss: 0.0145 - val_accuracy: 0.7494 - val_loss: 2.1251
Epoch 35/50
200/200              1s 3ms/step -
accuracy: 0.9973 - loss: 0.0106 - val_accuracy: 0.7519 - val_loss: 2.1650
Epoch 36/50
200/200              0s 2ms/step -
accuracy: 0.9967 - loss: 0.0112 - val_accuracy: 0.7519 - val_loss: 2.2162
Epoch 37/50
200/200              0s 2ms/step -
accuracy: 0.9979 - loss: 0.0116 - val_accuracy: 0.7431 - val_loss: 2.2667
Epoch 38/50
200/200              0s 2ms/step -
accuracy: 0.9981 - loss: 0.0087 - val_accuracy: 0.7481 - val_loss: 2.3361
Epoch 39/50
200/200              0s 2ms/step -
accuracy: 0.9983 - loss: 0.0081 - val_accuracy: 0.7487 - val_loss: 2.3949
Epoch 40/50
200/200              0s 2ms/step -
accuracy: 0.9964 - loss: 0.0119 - val_accuracy: 0.7513 - val_loss: 2.4292
Epoch 41/50
```

```
200/200              0s 2ms/step -
accuracy: 0.9970 - loss: 0.0086 - val_accuracy: 0.7456 - val_loss: 2.4132
Epoch 42/50
200/200              1s 3ms/step -
accuracy: 0.9979 - loss: 0.0063 - val_accuracy: 0.7456 - val_loss: 2.4626
Epoch 43/50
200/200              1s 2ms/step -
accuracy: 0.9984 - loss: 0.0065 - val_accuracy: 0.7494 - val_loss: 2.4742
Epoch 44/50
200/200              0s 2ms/step -
accuracy: 0.9990 - loss: 0.0053 - val_accuracy: 0.7519 - val_loss: 2.6020
Epoch 45/50
200/200              0s 2ms/step -
accuracy: 0.9992 - loss: 0.0054 - val_accuracy: 0.7506 - val_loss: 2.5558
Epoch 46/50
200/200              0s 2ms/step -
accuracy: 0.9980 - loss: 0.0092 - val_accuracy: 0.7419 - val_loss: 2.6279
Epoch 47/50
200/200              0s 2ms/step -
accuracy: 0.9971 - loss: 0.0097 - val_accuracy: 0.7594 - val_loss: 2.5883
Epoch 48/50
200/200              0s 2ms/step -
accuracy: 0.9959 - loss: 0.0160 - val_accuracy: 0.7525 - val_loss: 2.5387
Epoch 49/50
200/200              1s 3ms/step -
accuracy: 0.9944 - loss: 0.0191 - val_accuracy: 0.7456 - val_loss: 2.5016
Epoch 50/50
200/200              0s 2ms/step -
accuracy: 0.9972 - loss: 0.0116 - val_accuracy: 0.7563 - val_loss: 2.5447
{'accuracy': [0.7549999952316284, 0.8125, 0.8364062309265137,
0.8528125286102295, 0.8665624856948853, 0.8732812404632568, 0.8785937428474426,
0.8860937356948853, 0.8910937309265137, 0.9014062285423279, 0.9059374928474426,
0.91796875, 0.9262499809265137, 0.9315624833106995, 0.942187488079071,
0.9496874809265137, 0.9573437571525574, 0.9637500047683716, 0.9714062213897705,
0.9765625, 0.9801562428474426, 0.9829687476158142, 0.9870312213897705,
0.9871875047683716, 0.9898437261581421, 0.9917187690734863, 0.9921875,
0.9937499761581421, 0.9950000047683716, 0.9964062571525574, 0.9965624809265137,
0.9962499737739563, 0.9957812428474426, 0.99609375, 0.9971874952316284,
0.9965624809265137, 0.9976562261581421, 0.9982812404632568, 0.9981250166893005,
0.9971874952316284, 0.9981250166893005, 0.9976562261581421, 0.9984375238418579,
0.999218761920929, 0.9989062547683716, 0.9975000023841858, 0.9962499737739563,
0.9959375262260437, 0.9950000047683716, 0.996874988079071], 'loss':
[0.5601981282234192, 0.412230521440506, 0.34983593225479126,
0.31614336371421814, 0.29304665327072144, 0.27458441257476807,
0.2608238160610199, 0.24377702176570892, 0.23097918927669525,
0.21321630477905273, 0.1997666358947754, 0.18386124074459076,
0.1677486002445221, 0.15334337949752808, 0.13562412559986115,
0.11958098411560059, 0.104751355946064, 0.09151662141084671,
```

```
0.07862219214439392, 0.0673275962471962, 0.058343637734651566,
0.04943905770778656, 0.04263339564204216, 0.037815384566783905,
0.03324383497238159, 0.03129428252577782, 0.027008483186364174,
0.024421803653240204, 0.02057233266532421, 0.017779359593987465,
0.01656091958284378, 0.015143761411309242, 0.014990261755883694,
0.015423358418047428, 0.011869342066347599, 0.011964371427893639,
0.010659548453986645, 0.008586808107793331, 0.008896760642528534,
0.010534089989960194, 0.008305300958454609, 0.007050278130918741,
0.0066164471209049225, 0.004999118857085705, 0.005917804781347513,
0.009906486608088017, 0.013881074264645576, 0.016141420230269432,
0.016124999150633812, 0.012920962646603584], 'val_accuracy':
[0.7943750023841858, 0.7712500095367432, 0.753125011920929, 0.7368749976158142,
0.7287499904632568, 0.7181249856948853, 0.7174999713897705, 0.7243750095367432,
0.7118750214576721, 0.7168750166893005, 0.7118750214576721, 0.7143750190734863,
0.7206249833106995, 0.7168750166893005, 0.7262499928474426, 0.7337499856948853,
0.7268750071525574, 0.7306249737739563, 0.7275000214576721, 0.731249988079071,
0.734375, 0.737500011920929, 0.7331249713897705, 0.7368749976158142,
0.7337499856948853, 0.737500011920929, 0.7368749976158142, 0.7412499785423279,
0.7393749952316284, 0.7437499761581421, 0.7400000095367432, 0.7425000071525574,
0.7381250262260437, 0.7493749856948853, 0.7518749833106995, 0.7518749833106995,
0.7431250214576721, 0.7481250166893005, 0.7487499713897705, 0.7512500286102295,
0.7456250190734863, 0.7456250190734863, 0.7493749856948853, 0.7518749833106995,
0.7506250143051147, 0.7418749928474426, 0.7593749761581421, 0.7524999976158142,
0.7456250190734863, 0.7562500238418579], 'val_loss': [0.5140730142593384,
0.5426977276802063, 0.5722506046295166, 0.6044119596481323, 0.6331974864006042,
0.6698675751686096, 0.709787905216217, 0.7259223461151123, 0.7891015410423279,
0.7815655469894409, 0.8578264713287354, 0.8983173966407776, 0.9856798052787781,
1.0088597536087036, 1.044642448425293, 1.1106300354003906, 1.1864399909973145,
1.243656039237976, 1.3099699020385742, 1.3680219650268555, 1.406174898147583,
1.466016411781311, 1.55442476272583, 1.6068129539489746, 1.6702532768249512,
1.7006173133850098, 1.7706114053726196, 1.8225336074829102, 1.8539929389953613,
1.933018445968628, 1.9963420629501343, 2.042693853378296, 2.086225748062134,
2.1250882148742676, 2.1650307178497314, 2.2161989212036133, 2.266688823699951,
2.3361470699310303, 2.3949286937713623, 2.4292209148406982, 2.4132468700408936,
2.4626195430755615, 2.4742307662963867, 2.6020114421844482, 2.5558199882507324,
2.627865791320801, 2.5882883071899414, 2.5387377738952637, 2.501645803451538,
2.5447041988372803]}
```


[47]:
```
# Neural Network Training Progress Overview

## Key Components

- **Epochs**: Each epoch indicates one complete cycle through the training dataset. In this output, the model was trained for 50 epochs.

- **Batch Progress**: The `200/200` indicates that the training process has completed 200 batches for that epoch.
```

```
- **Metrics**:
  - **accuracy**: The accuracy of the model on the training set for that epoch.
  - **loss**: The training loss, which measures how well the model is␣
 ↪performing (lower is better).
  - **val_accuracy**: The accuracy of the model on the validation set (data not␣
 ↪seen by the model during training).
  - **val_loss**: The loss on the validation set.

## Observations

- **Training Accuracy**: Starts around 70.58% in the first epoch and improves␣
 ↪to about 99.22% by the 50th epoch, indicating the model is learning well.

- **Training Loss**: Starts at 0.6054 and decreases to 0.0054, showing that the␣
 ↪model is reducing its error on the training data.

- **Validation Metrics**:
  - Validation accuracy starts at 79.44% and fluctuates, eventually reaching 75.
 ↪63%. The slight drop or fluctuation in validation accuracy compared to␣
 ↪training accuracy can indicate overfitting, where the model learns the␣
 ↪training data too well but does not generalize effectively to unseen data.
  - Validation loss increases initially before decreasing but stabilizes,␣
 ↪suggesting the model's performance on unseen data may not improve beyond a␣
 ↪certain point.
```

```
  Cell In[47], line 17
    - **Training Accuracy**: Starts around 70.58% in the first epoch and␣
 ↪improves to about 99.22% by the 50th epoch, indicating the model is learning␣
 ↪well.
                                                                              ␣
 ↪                            ^
SyntaxError: invalid decimal literal
```

[ ]:
```
#Step 5: Evaluate the Model
```

[33]:
```
from sklearn.metrics import accuracy_score, confusion_matrix

# Make predictions
y_pred = (model.predict(X_test) > 0.5).astype("int32")

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```python
# Print confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
```

```
63/63            0s 2ms/step
Accuracy: 0.74
Confusion Matrix:
[[1360  247]
 [ 270  123]]
```

[ ]:

```python

```

[ ]:

```python

```