Income Classification

June 10, 2025

[1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

[3]:
```python
data = pd.read_csv('adult.csv')
print("Null values per column:")
print(data.isnull().sum())
```

```
Null values per column:
age                0
workclass          0
fnlwgt             0
education          0
educational-num    0
marital-status     0
occupation         0
relationship       0
race               0
gender             0
capital-gain       0
capital-loss       0
hours-per-week     0
native-country     0
income             0
dtype: int64
```

[5]:
```python
data = data.drop(columns=['fnlwgt'])
```

[7]:
```python
data['income'] = data['income'].apply(lambda x: 1 if x.strip() ==
'>50K' else 0)
```

[9]:
```python
categorical_cols = data.select_dtypes(include=['object']).columns
for col in categorical_cols:
 unique_values = data[col].unique()
 mapping = {value: idx for idx, value in enumerate(unique_values)}
 data[col] = data[col].map(mapping)
```

[11]:
```
X = data.drop('income', axis=1).values
y = data['income'].values
```

[13]:
```
X_mean = X.mean(axis=0)
X_std = X.std(axis=0)
X_scaled = (X - X_mean) / X_std
```

[15]:
```
def train_test_split_manual(X, y, test_size=0.2, random_state=None):
    if random_state is not None:
        np.random.seed(random_state)
    indices = np.arange(X.shape[0])
    np.random.shuffle(indices)
    split_index = int(X.shape[0] * (1 - test_size))
    train_indices = indices[:split_index]
    test_indices = indices[split_index:]
    X_train, X_test = X[train_indices], X[test_indices]
    y_train, y_test = y[train_indices], y[test_indices]
    return X_train, X_test, y_train, y_test
```

[17]:
```
X_train, X_test, y_train, y_test = train_test_split_manual(X_scaled,
y, test_size=0.2, random_state=42)
```

[19]:
```
model = Sequential([
 Dense(60, activation='relu', input_shape=(X_train.shape[1],)),
 Dense(30, activation='relu'),
 Dense(15, activation='relu'),
 Dense(7, activation='relu'),
 Dense(1, activation='sigmoid')
])
```

```
C:\Users\sajee\anaconda3\Lib\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)
```

[21]:
```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 60)	840
dense_1 (Dense)	(None, 30)	1,830
dense_2 (Dense)	(None, 15)	465

```
 dense_3 (Dense)                      (None, 7)                          112

 dense_4 (Dense)                      (None, 1)                            8
```

Total params: 3,255 (12.71 KB)

Trainable params: 3,255 (12.71 KB)

Non-trainable params: 0 (0.00 B)

[23]:
```
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
```

[25]:
```
history = model.fit(X_train, y_train, epochs=50, batch_size=30,
validation_split=0.2, verbose=1)
```

```
Epoch 1/50
1042/1042            11s 5ms/step -
accuracy: 0.8105 - loss: 0.4055 - val_accuracy: 0.8412 - val_loss: 0.3343
Epoch 2/50
1042/1042            5s 4ms/step -
accuracy: 0.8478 - loss: 0.3216 - val_accuracy: 0.8522 - val_loss: 0.3250
Epoch 3/50
1042/1042            5s 5ms/step -
accuracy: 0.8485 - loss: 0.3238 - val_accuracy: 0.8528 - val_loss: 0.3224
Epoch 4/50
1042/1042            5s 5ms/step -
accuracy: 0.8571 - loss: 0.3097 - val_accuracy: 0.8522 - val_loss: 0.3214
Epoch 5/50
1042/1042            4s 4ms/step -
accuracy: 0.8553 - loss: 0.3143 - val_accuracy: 0.8535 - val_loss: 0.3208
Epoch 6/50
1042/1042            5s 4ms/step -
accuracy: 0.8586 - loss: 0.3101 - val_accuracy: 0.8531 - val_loss: 0.3209
Epoch 7/50
1042/1042            5s 4ms/step -
accuracy: 0.8599 - loss: 0.3081 - val_accuracy: 0.8537 - val_loss: 0.3184
Epoch 8/50
1042/1042            5s 4ms/step -
accuracy: 0.8607 - loss: 0.3014 - val_accuracy: 0.8513 - val_loss: 0.3213
Epoch 9/50
1042/1042            5s 4ms/step -
accuracy: 0.8606 - loss: 0.3052 - val_accuracy: 0.8518 - val_loss: 0.3219
Epoch 10/50
```

```
1042/1042            5s 4ms/step -
accuracy: 0.8590 - loss: 0.3064 - val_accuracy: 0.8502 - val_loss: 0.3202
Epoch 11/50
1042/1042            5s 4ms/step -
accuracy: 0.8578 - loss: 0.3068 - val_accuracy: 0.8518 - val_loss: 0.3182
Epoch 12/50
1042/1042            4s 4ms/step -
accuracy: 0.8598 - loss: 0.3040 - val_accuracy: 0.8500 - val_loss: 0.3198
Epoch 13/50
1042/1042            5s 4ms/step -
accuracy: 0.8626 - loss: 0.2993 - val_accuracy: 0.8486 - val_loss: 0.3190
Epoch 14/50
1042/1042            5s 4ms/step -
accuracy: 0.8614 - loss: 0.2974 - val_accuracy: 0.8518 - val_loss: 0.3202
Epoch 15/50
1042/1042            5s 4ms/step -
accuracy: 0.8624 - loss: 0.3006 - val_accuracy: 0.8525 - val_loss: 0.3208
Epoch 16/50
1042/1042            5s 4ms/step -
accuracy: 0.8624 - loss: 0.2954 - val_accuracy: 0.8522 - val_loss: 0.3209
Epoch 17/50
1042/1042            5s 5ms/step -
accuracy: 0.8669 - loss: 0.2942 - val_accuracy: 0.8422 - val_loss: 0.3348
Epoch 18/50
1042/1042            5s 4ms/step -
accuracy: 0.8619 - loss: 0.2941 - val_accuracy: 0.8502 - val_loss: 0.3209
Epoch 19/50
1042/1042            5s 4ms/step -
accuracy: 0.8665 - loss: 0.2914 - val_accuracy: 0.8505 - val_loss: 0.3206
Epoch 20/50
1042/1042            5s 4ms/step -
accuracy: 0.8667 - loss: 0.2895 - val_accuracy: 0.8489 - val_loss: 0.3225
Epoch 21/50
1042/1042            5s 4ms/step -
accuracy: 0.8663 - loss: 0.2877 - val_accuracy: 0.8473 - val_loss: 0.3225
Epoch 22/50
1042/1042            5s 4ms/step -
accuracy: 0.8672 - loss: 0.2907 - val_accuracy: 0.8532 - val_loss: 0.3217
Epoch 23/50
1042/1042            5s 4ms/step -
accuracy: 0.8644 - loss: 0.2921 - val_accuracy: 0.8480 - val_loss: 0.3210
Epoch 24/50
1042/1042            5s 4ms/step -
accuracy: 0.8658 - loss: 0.2892 - val_accuracy: 0.8517 - val_loss: 0.3207
Epoch 25/50
1042/1042            5s 4ms/step -
accuracy: 0.8688 - loss: 0.2866 - val_accuracy: 0.8512 - val_loss: 0.3222
Epoch 26/50
```

```
1042/1042            5s 4ms/step -
accuracy: 0.8642 - loss: 0.2902 - val_accuracy: 0.8479 - val_loss: 0.3324
Epoch 27/50
1042/1042            4s 4ms/step -
accuracy: 0.8658 - loss: 0.2886 - val_accuracy: 0.8489 - val_loss: 0.3236
Epoch 28/50
1042/1042            5s 4ms/step -
accuracy: 0.8663 - loss: 0.2847 - val_accuracy: 0.8473 - val_loss: 0.3248
Epoch 29/50
1042/1042            5s 4ms/step -
accuracy: 0.8648 - loss: 0.2861 - val_accuracy: 0.8508 - val_loss: 0.3294
Epoch 30/50
1042/1042            5s 5ms/step -
accuracy: 0.8715 - loss: 0.2775 - val_accuracy: 0.8445 - val_loss: 0.3287
Epoch 31/50
1042/1042            5s 5ms/step -
accuracy: 0.8686 - loss: 0.2823 - val_accuracy: 0.8491 - val_loss: 0.3263
Epoch 32/50
1042/1042            5s 5ms/step -
accuracy: 0.8700 - loss: 0.2806 - val_accuracy: 0.8484 - val_loss: 0.3253
Epoch 33/50
1042/1042            5s 5ms/step -
accuracy: 0.8704 - loss: 0.2818 - val_accuracy: 0.8477 - val_loss: 0.3255
Epoch 34/50
1042/1042            5s 5ms/step -
accuracy: 0.8673 - loss: 0.2837 - val_accuracy: 0.8477 - val_loss: 0.3303
Epoch 35/50
1042/1042            5s 5ms/step -
accuracy: 0.8728 - loss: 0.2760 - val_accuracy: 0.8463 - val_loss: 0.3262
Epoch 36/50
1042/1042            5s 5ms/step -
accuracy: 0.8686 - loss: 0.2789 - val_accuracy: 0.8495 - val_loss: 0.3271
Epoch 37/50
1042/1042            5s 5ms/step -
accuracy: 0.8688 - loss: 0.2838 - val_accuracy: 0.8496 - val_loss: 0.3308
Epoch 38/50
1042/1042            5s 5ms/step -
accuracy: 0.8715 - loss: 0.2774 - val_accuracy: 0.8491 - val_loss: 0.3309
Epoch 39/50
1042/1042            5s 5ms/step -
accuracy: 0.8699 - loss: 0.2781 - val_accuracy: 0.8475 - val_loss: 0.3338
Epoch 40/50
1042/1042            5s 5ms/step -
accuracy: 0.8759 - loss: 0.2721 - val_accuracy: 0.8475 - val_loss: 0.3326
Epoch 41/50
1042/1042            5s 5ms/step -
accuracy: 0.8709 - loss: 0.2769 - val_accuracy: 0.8459 - val_loss: 0.3322
Epoch 42/50
```

```
1042/1042               5s 4ms/step -
accuracy: 0.8741 - loss: 0.2711 - val_accuracy: 0.8499 - val_loss: 0.3332
Epoch 43/50
1042/1042               5s 5ms/step -
accuracy: 0.8730 - loss: 0.2718 - val_accuracy: 0.8490 - val_loss: 0.3320
Epoch 44/50
1042/1042               5s 5ms/step -
accuracy: 0.8737 - loss: 0.2728 - val_accuracy: 0.8499 - val_loss: 0.3370
Epoch 45/50
1042/1042               5s 4ms/step -
accuracy: 0.8703 - loss: 0.2770 - val_accuracy: 0.8454 - val_loss: 0.3406
Epoch 46/50
1042/1042               5s 5ms/step -
accuracy: 0.8727 - loss: 0.2746 - val_accuracy: 0.8475 - val_loss: 0.3405
Epoch 47/50
1042/1042               5s 5ms/step -
accuracy: 0.8718 - loss: 0.2783 - val_accuracy: 0.8480 - val_loss: 0.3399
Epoch 48/50
1042/1042               5s 5ms/step -
accuracy: 0.8728 - loss: 0.2712 - val_accuracy: 0.8358 - val_loss: 0.3532
Epoch 49/50
1042/1042               5s 5ms/step -
accuracy: 0.8755 - loss: 0.2697 - val_accuracy: 0.8461 - val_loss: 0.3390
Epoch 50/50
1042/1042               5s 5ms/step -
accuracy: 0.8767 - loss: 0.2668 - val_accuracy: 0.8477 - val_loss: 0.3442
```

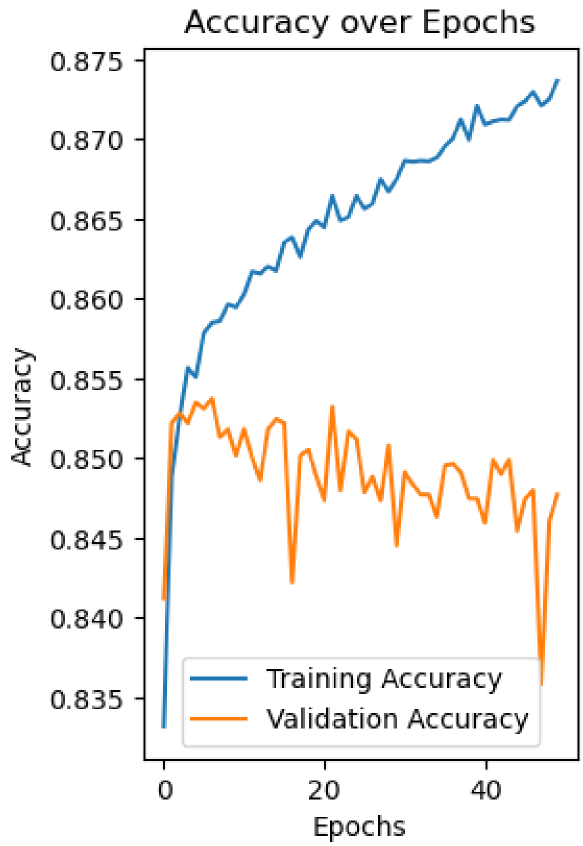
[31]:
```
plt.figure(figsize=(12, 5))
```

[31]:
```
<Figure size 1200x500 with 0 Axes>
```

```
<Figure size 1200x500 with 0 Axes>
```

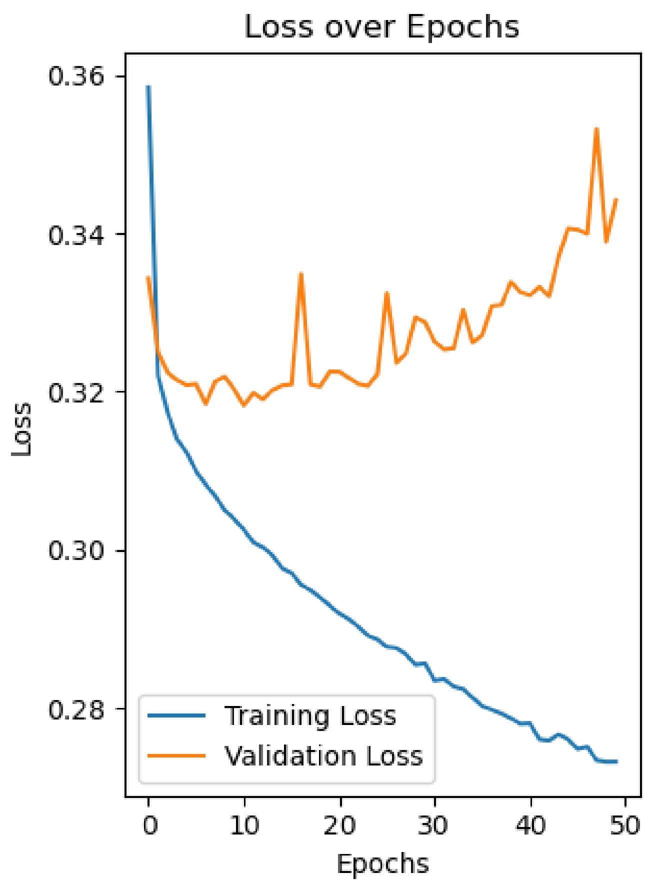
[33]:
```
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Accuracy over Epochs')
```

[33]:
```
Text(0.5, 1.0, 'Accuracy over Epochs')
```



[35]:

```
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Loss over Epochs')
plt.tight_layout()
plt.show()
```



[37]:
```
def tanh(x):
 return np.tanh(x)
```

[39]:
```
def tanh_derivative(x):
 return 1 - np.tanh(x) ** 2
```

[41]:
```
def elu(x, alpha=1.0):
 return np.where(x >= 0, x, alpha * (np.exp(x) - 1))
```

[43]:
```
def elu_derivative(x, alpha=1.0):
 return np.where(x >= 0, 1, alpha * np.exp(x))
```

[45]:
```
x_vals = np.linspace(-10, 10, 100)
```

[47]:
```
outputs_tanh = tanh(x_vals)
derivatives_tanh = tanh_derivative(x_vals)
outputs_elu = elu(x_vals)
derivatives_elu = elu_derivative(x_vals)
```

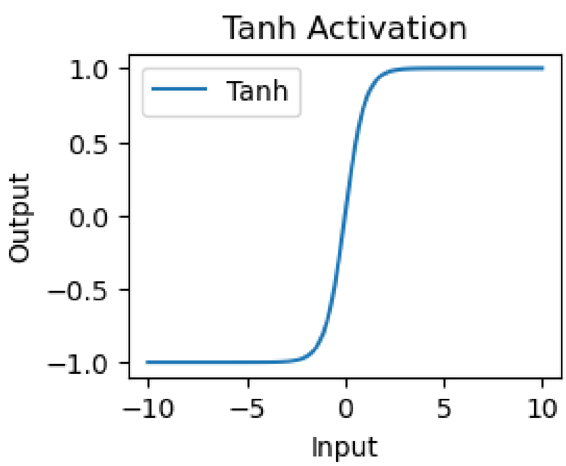
[49]:
```python
plt.figure(figsize=(12, 10))
```

[49]: <Figure size 1200x1000 with 0 Axes>

<Figure size 1200x1000 with 0 Axes>

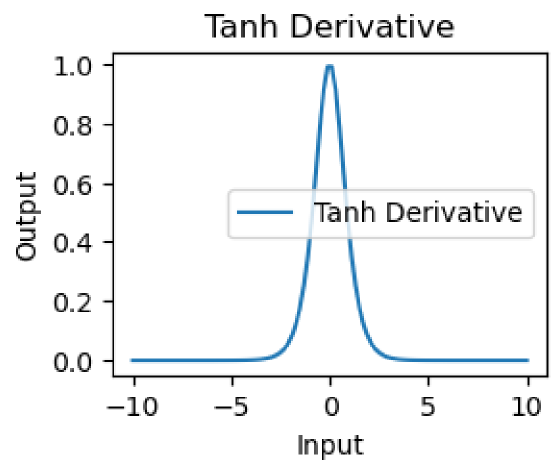
[51]:
```python
plt.subplot(2, 2, 1)
plt.plot(x_vals, outputs_tanh, label='Tanh')
plt.title('Tanh Activation')
plt.xlabel('Input')
plt.ylabel('Output')
plt.legend()
```

[51]: <matplotlib.legend.Legend at 0x22350c2f9e0>



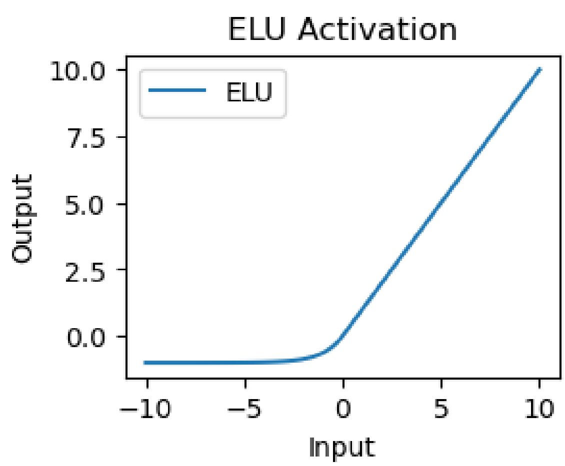
[53]:
```python
plt.subplot(2, 2, 2)
plt.plot(x_vals, derivatives_tanh, label='Tanh Derivative')
plt.title('Tanh Derivative')
plt.xlabel('Input')
plt.ylabel('Output')
plt.legend()
```

[53]: <matplotlib.legend.Legend at 0x22351488410>



[55]:
```
plt.subplot(2, 2, 3)
plt.plot(x_vals, outputs_elu, label='ELU')
plt.title('ELU Activation')
plt.xlabel('Input')
plt.ylabel('Output')
plt.legend()
```

[55]: <matplotlib.legend.Legend at 0x2234f35e7e0>



[57]:
```
plt.subplot(2, 2, 4)
plt.plot(x_vals, derivatives_elu, label='ELU Derivative')
plt.title('ELU Derivative')
plt.xlabel('Input')
plt.ylabel('Output')
```

```
plt.legend()
```

[57]: <matplotlib.legend.Legend at 0x22350901df0>