Scratch_Implementation2

May 29, 2025

[20]:
```
import pandas as pd
import numpy as np
```

[21]:
```
df = pd.read_csv("KaggleV2-May-2016(Copy).csv")
df.head()
```

[21]:

	Gender	Age	Neighbourhood	Scholarship	Hipertension	Diabetes	\
0	0	62	1	0	1	0	
1	1	56	1	0	0	0	
2	0	62	2	0	0	0	
3	0	8	3	0	0	0	
4	0	56	1	0	1	1	

	Alcoholism	Handcap	SMS_received	No-show	Scheduled_Day	Scheduled_Month	\
0	0	0	0	0	29	4	
1	0	0	0	0	29	4	
2	0	0	0	0	29	4	
3	0	0	0	0	29	4	
4	0	0	0	0	29	4	

	Appointment_Day	Appointment_Month
0	29	4
1	29	4
2	29	4
3	29	4
4	29	4

[22]:
```
from sklearn.model_selection import train_test_split

x = df.drop(["No-show"], axis=1)
y = df["No-show"]

x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.2, random_state=27)
```

[23]:
```
def sigmoid(z):
    # z = np.clip(z, -500, 500)
    return 1/ (1 + np.exp(-z))
```

```
    # return z
def relu(z):
    # return np.maximum(0,z)
    return z
def leaky_relu(z, alpha = 0.01):
    return np.where(x > 0, x, alpha * x)
def relu_derivative(Z):
    return Z>0
def leaky_relu_derivative(x, alpha=0.01):
    return np.where(x > 0, 1, alpha)
def batchnorm(x):
    mu = np.mean(x, axis=0)
    var = np.var(x, axis=0)
    return (x - mu) / np.sqrt(var + 1e-8)
```

[24]:
```
def loss_function(a4, y):
    y = np.array(y)
    a4 = np.array(a4)

    weight_0 = np.sum(a4) / (2 * np.sum(y == 0))
    weight_1 = np.sum(a4) / (2 * np.sum(y == 1))

    loss = -np.mean(weight_1 * y * np.log(a4 + 1e-8) + 0.1 * weight_0 * (1 - y)␣
 ↪* np.log(1 - a4 + 1e-8))
    return loss
```

[25]:
```
np.random.seed(27)
input_size = 13
layer1 = 24
layer2 = 16
layer3 = 8
output_size = 1

w1 = np.random.randn(input_size, layer1) * 0.05
b1 = np.zeros((1, layer1))
w2 = np.random.randn(layer1, layer2) * 0.05
b2 = np.zeros((1, layer2))
w3 = np.random.randn(layer2, layer3) * 0.05
b3 = np.zeros((1, layer3))
w4 = np.random.randn(layer3, output_size) * 0.05
b4 = np.zeros((1, output_size))

lr = 0.05
```

[26]:
```
def update_parameters(w1, b1, w2, b2, w3, b3, w4, b4,
                      dw1, db1, dw2, db2, dw3, db3, dw4, db4, lr):
    w1 -= lr * dw1
```

```
    b1 -= lr * db1
    w2 -= lr * dw2
    b2 -= lr * db2
    w3 -= lr * dw3
    b3 -= lr * db3
    w4 -= lr * dw4
    b4 -= lr * db4
    return w1, b1, w2, b2, w3, b3, w4, b4
```

[27]:
```
import time
start_time = time.time()

epochs = 20
for epoch in range(epochs):

    z1 = np.dot(x_train, w1) + b1
    a1 = relu(z1)
    z2 = np.dot(a1, w2) + b2
    a2 = relu(z2)
    z3 = np.dot(a2, w3) + b3
    a3 = relu(z3)
    z4 = np.dot(a3, w4) + b4
    a4 = sigmoid(z4)

    current_loss = loss_function(a4, y_train.values.reshape(-1,1))
    print(f"Epoch {epoch+1}/{epochs}, Loss: {current_loss}")


    for i in range(x_train.shape[0]):
        x = x_train.iloc[i].to_numpy().reshape(1, -1)
        y = y_train[i:i+1].values


        z1 = np.dot(x, w1) + b1
        a1 = relu(z1)
        z2 = np.dot(a1, w2) + b2
        a2 = relu(z2)
        z3 = np.dot(a2, w3) + b3
        a3 = relu(z3)
        z4 = np.dot(a3, w4) + b4
        a4 = sigmoid(z4)

        dz4 = a4 - y
        dw4 = np.dot(a3.T, dz4)
        db4 = np.sum(dz4, axis=0, keepdims=True)

        da3 = np.dot(dz4, w4.T)
```

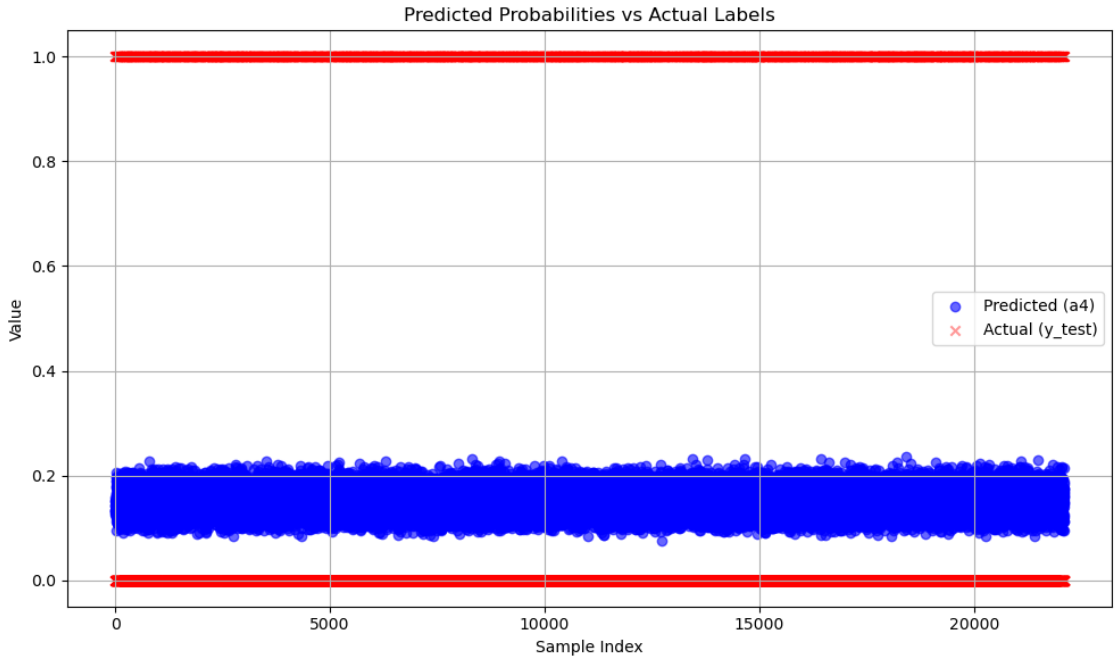
```
        dz3 = da3 * relu_derivative(z3)
        dw3 = np.dot(a2.T, dz3)
        db3 = np.sum(dz3, axis=0, keepdims=True)

        da2 = np.dot(dz3, w3.T)
        dz2 = da2 * relu_derivative(z2)
        dw2 = np.dot(a1.T, dz2)
        db2 = np.sum(dz2, axis=0, keepdims=True)

        da1 = np.dot(dz2, w2.T)
        dz1 = da1 * relu_derivative(z1)
        dw1 = np.dot(x.T, dz1)
        db1 = np.sum(dz1, axis=0, keepdims=True)


        w1, b1, w2, b2, w3, b3, w4, b4 = update_parameters(
            w1, b1, w2, b2, w3, b3, w4, b4,
            dw1, db1, dw2, db2, dw3, db3, dw4, db4,
            lr
        )

end_time = time.time()
print(f"Time required for training process {end_time-start_time}")
```

```
Epoch 1/20, Loss: 0.19026837038559136
Epoch 2/20, Loss: 0.14278510075756742
Epoch 3/20, Loss: 0.14278542060020663
Epoch 4/20, Loss: 0.14276326082724272
Epoch 5/20, Loss: 0.1427450537625947
Epoch 6/20, Loss: 0.14272874727561402
Epoch 7/20, Loss: 0.1427133827719675
Epoch 8/20, Loss: 0.1426986432833867
Epoch 9/20, Loss: 0.14268442245349267
Epoch 10/20, Loss: 0.14267068046194126
Epoch 11/20, Loss: 0.14265739849989564
Epoch 12/20, Loss: 0.14264456445953405
Epoch 13/20, Loss: 0.14263216843795457
Epoch 14/20, Loss: 0.14262020132420522
Epoch 15/20, Loss: 0.14260865435449874
Epoch 16/20, Loss: 0.14259751897136586
Epoch 17/20, Loss: 0.1425867867781917
Epoch 18/20, Loss: 0.1425764495236834
Epoch 19/20, Loss: 0.14256649909572996
Epoch 20/20, Loss: 0.142556927518218
Time required for training process 117.60946297645569
```

[28]:
```
z1 = np.dot(x_test, w1) + b1
a1 = relu(z1)
z2 = np.dot(a1, w2) + b2
a2 = relu(z2)
z3 = np.dot(a2, w3) + b3
a3 = relu(z3)
z4 = np.dot(a3, w4) + b4
a4 = sigmoid(z4)
print(a4)
```

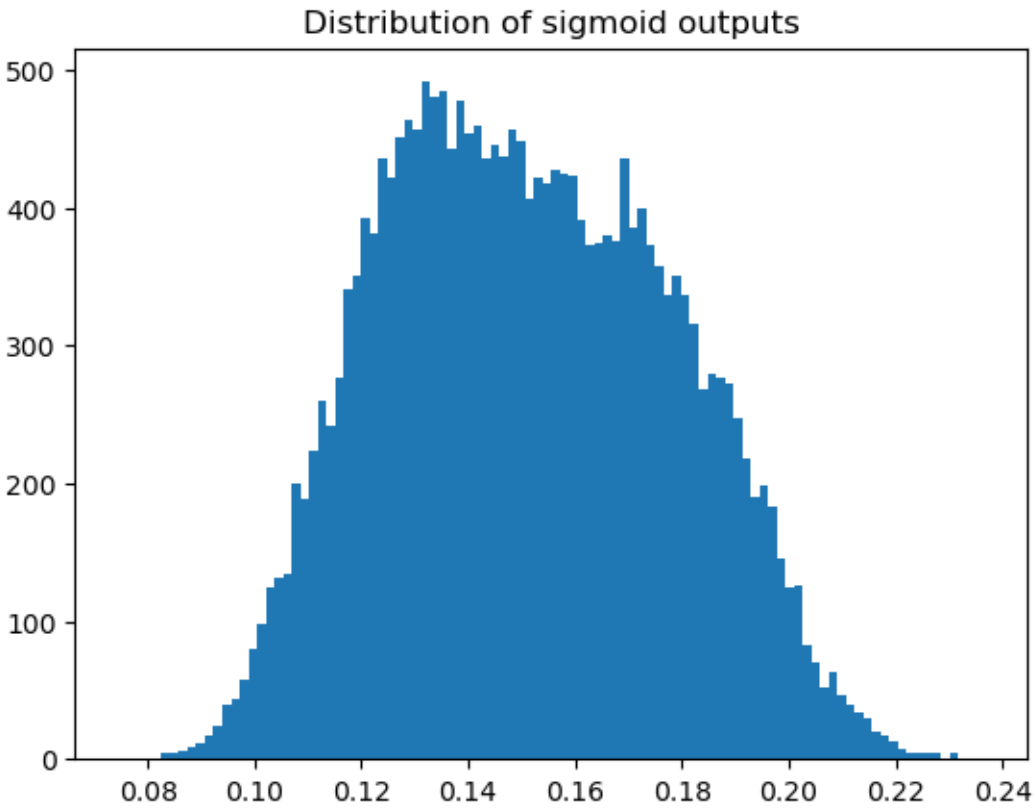
```
[[0.14273904]
 [0.15794383]
 [0.13305887]
 …
 [0.18500064]
 [0.16979645]
 [0.15977562]]
```

[29]:
```
import matplotlib.pyplot as plt

y_pred_probs = a4.flatten()
y_actual = y_test.values.flatten()

plt.figure(figsize=(10, 6))
plt.scatter(range(len(y_pred_probs)), y_pred_probs, label='Predicted (a4)',␣
 ↪alpha=0.6, color='blue')
plt.scatter(range(len(y_actual)), y_actual, label='Actual (y_test)', alpha=0.4,␣
 ↪color='red', marker='x')
plt.title('Predicted Probabilities vs Actual Labels')
plt.xlabel('Sample Index')
plt.ylabel('Value')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



[30]:
```
from sklearn.metrics import precision_recall_curve
y_test_arr = y_test.values.reshape(-1, 1)
precision, recall, thresholds = precision_recall_curve(y_test_arr, a4)
f1_scores = 2 * precision * recall / (precision + recall + 1e-8)
best_threshold = thresholds[np.argmax(f1_scores)]
```

[31]:
```
import matplotlib.pyplot as plt
plt.hist(a4, bins=100)
plt.title("Distribution of sigmoid outputs")
plt.show()
```

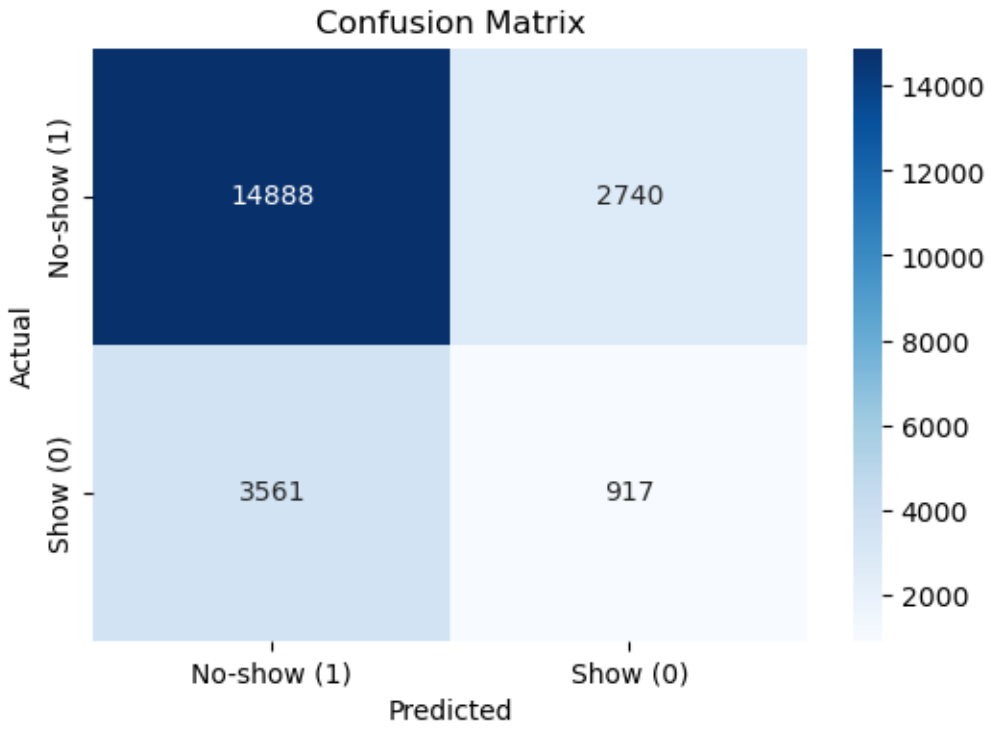


[32]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score,␣
 ↪recall_score, confusion_matrix, classification_report
from memory_profiler import memory_usage
y_pred = (a4 >= 0.18).astype(int)
y_true = y_test.values.reshape(-1, 1)
acc = accuracy_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred, zero_division=1)
pur = precision_score(y_true, y_pred, zero_division=1)
rec = recall_score(y_true, y_pred, zero_division=1)
report = classification_report(y_true, y_pred, zero_division=1)
print("Accuracy:", acc)
print("F1 Score:", f1)
print("Precision:", pur)
print("Recall:", rec)
print("Classification Report:\n", report)
print(f"Time required for training process {end_time-start_time}")
```

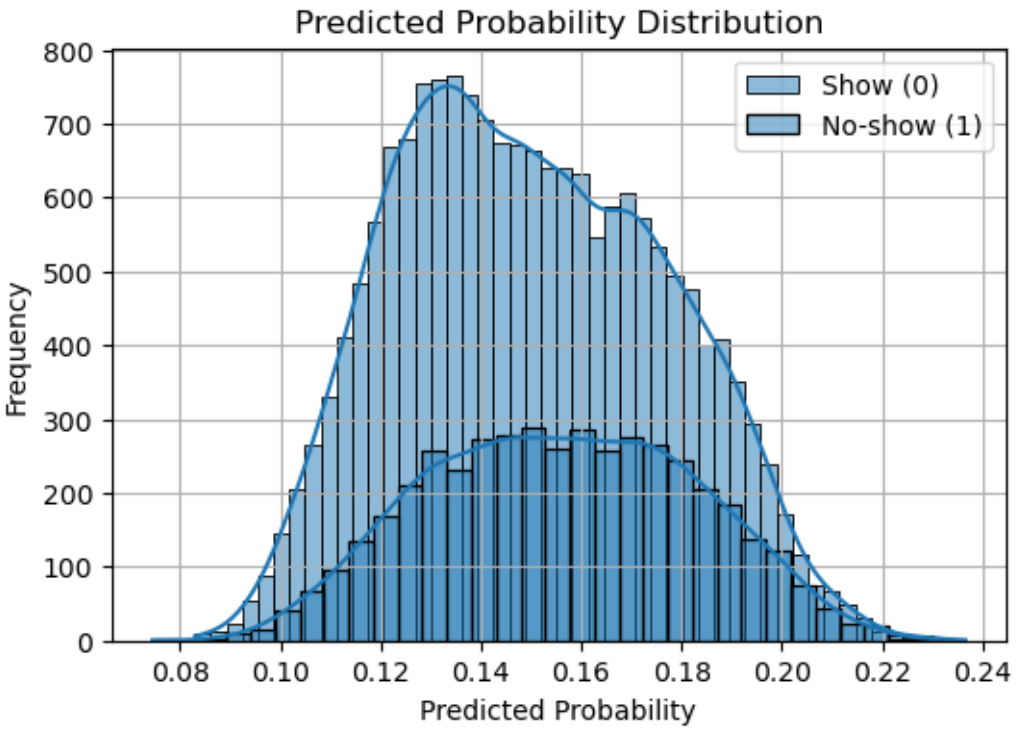
```
Accuracy: 0.714964263095992
F1 Score: 0.2254456054087277
Precision: 0.2507519824993164
Recall: 0.20477891916033944
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.84      0.83     17628
           1       0.25      0.20      0.23      4478

    accuracy                           0.71     22106
   macro avg       0.53      0.52      0.53     22106
weighted avg       0.69      0.71      0.70     22106

Time required for training process 117.60946297645569
```

[33]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

cm = confusion_matrix(y_true, y_pred)

labels = ['No-show (1)', 'Show (0)']

plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels,␣
 ↪yticklabels=labels)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



[34]:
```
plt.figure(figsize=(6, 4))
sns.histplot(a4[y_true.flatten() == 0], label='Show (0)', color='blue',␣
 ↪kde=True)
sns.histplot(a4[y_true.flatten() == 1], label='No-show (1)', color='red',␣
 ↪kde=True)
plt.title('Predicted Probability Distribution')
plt.xlabel('Predicted Probability')
plt.ylabel('Frequency')
plt.legend()
plt.grid(True)
plt.show()
```


Predicted Probability Distribution
Show (0)
No-show (1)
Frequency
Predicted Probability
0
100
200
300
400
500
600
700
800
0.08
0.10
0.12
0.14
0.16
0.18
0.20
0.22
0.24