Scratch_Implementation2

May 29, 2025

[28]:
```
import pandas as pd
import numpy as np
```

[29]:
```
df = pd.read_csv("KaggleV2-May-2016(Copy).csv")
df.head()
```

[29]:

	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

[30]:
```
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)
```

[31]:
```
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)
```

[32]:
```
def loss_function(a4, y):
    y = np.array(y)
    a4 = np.array(a4)
    eps = 1e-8
    weight_0 = np.sum(a4) / (2 * np.sum(y == 0) + eps)
    weight_1 = np.sum(a4) / (2 * np.sum(y == 1) + eps)

    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
```

[33]:
```
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
```

[34]:
```
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
```

[35]:
```
import time
start_time = time.time()

batch_size = 32
nums_batches = len(x_train) // batch_size


epochs = 20
for epoch in range(epochs):
    epoch_loss = 0
    indices = np.random.permutation(len(x_train))

    for i in range(nums_batches):
        batch_idx = indices[i*batch_size : (i+1)*batch_size]
        x_batch = x_train.iloc[batch_idx]  # Mini-batch
        y_batch = y_train.iloc[batch_idx]
        batch_loss = 0
        for j in range(x_batch.shape[0]):
            x = x_batch.iloc[j].to_numpy().reshape(1, -1)
            y = y_batch[j:j+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)

            batch_loss += loss_function(a4, y)

            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)
```

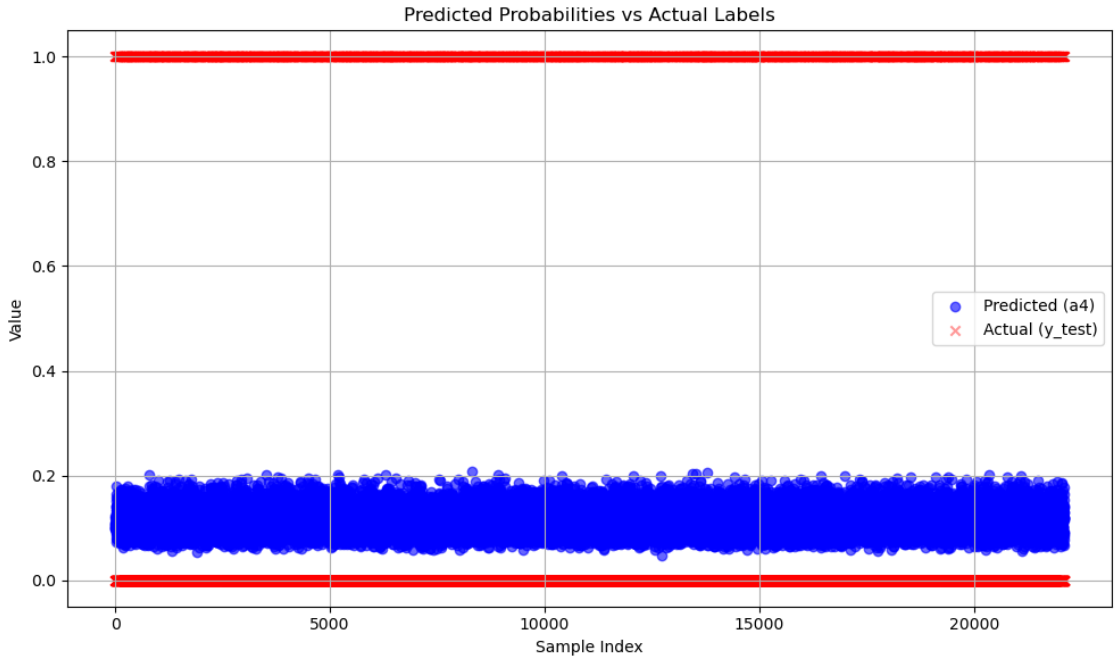
```
            dw3 = np.dot(a2.T, dz3)
            db3 = np.sum(dz3, axis=0, keepdims=True)

            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
            )
        epoch_loss += batch_loss/batch_size
    print(f"Epoch: {epoch+1}/{epochs}, Loss: {epoch_loss/nums_batches}")


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

```
Epoch: 1/20, Loss: 0.03406656789528292
Epoch: 2/20, Loss: 0.03389426170021447
Epoch: 3/20, Loss: 0.033899987356134864
Epoch: 4/20, Loss: 0.033901420172528465
Epoch: 5/20, Loss: 0.03389854490235255
Epoch: 6/20, Loss: 0.03390767348095878
Epoch: 7/20, Loss: 0.03390504065721396
Epoch: 8/20, Loss: 0.03391170442176555
Epoch: 9/20, Loss: 0.03391734582099624
Epoch: 10/20, Loss: 0.03392124624338785
Epoch: 11/20, Loss: 0.03393491119062967
Epoch: 12/20, Loss: 0.03393324650375503
Epoch: 13/20, Loss: 0.033934197635404305
Epoch: 14/20, Loss: 0.033965702156870836
Epoch: 15/20, Loss: 0.0339407104142966
Epoch: 16/20, Loss: 0.03396889702016474
Epoch: 17/20, Loss: 0.0339635363391369
Epoch: 18/20, Loss: 0.03397371526055185
Epoch: 19/20, Loss: 0.03397628622942658
Epoch: 20/20, Loss: 0.033982355536859715
Time required for training process 186.40725803375244
```

[36]:
```
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)
```

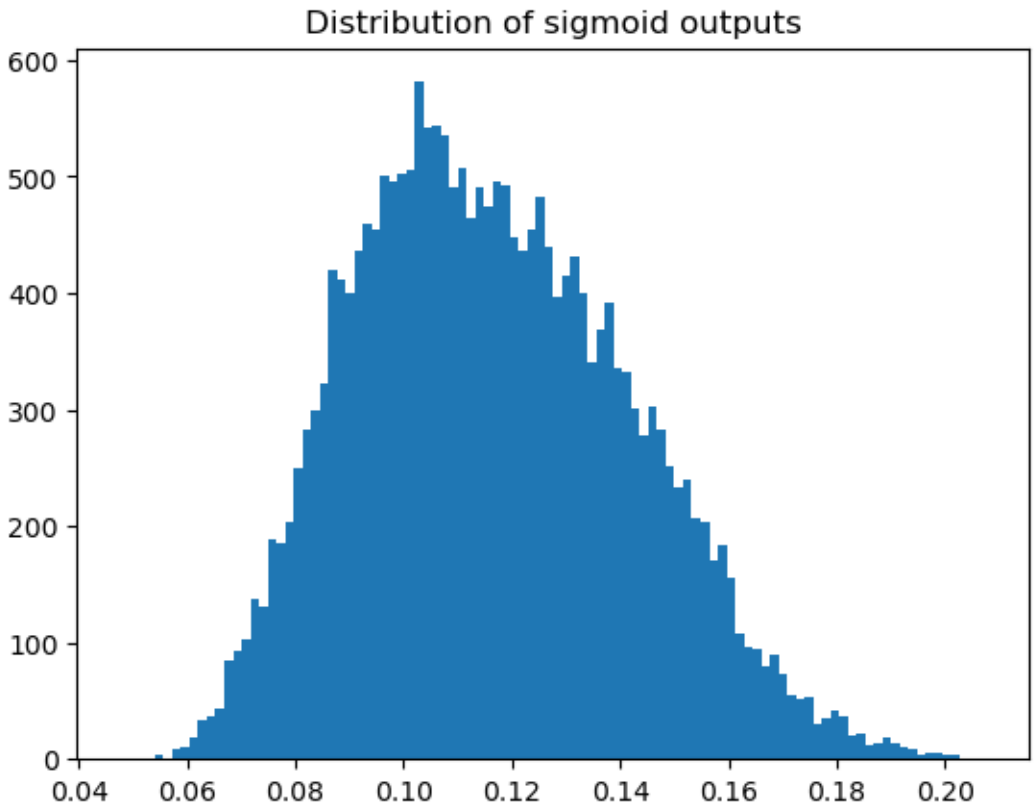
```
[[0.10424788]
 [0.12237453]
 [0.1007271 ]
 …
 [0.16398776]
 [0.13747538]
 [0.11958361]]
```

[37]:
```
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()
```



[38]:
```
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)]
```

[39]:
```
import matplotlib.pyplot as plt
plt.hist(a4, bins=100)
plt.title("Distribution of sigmoid outputs")
plt.show()
```

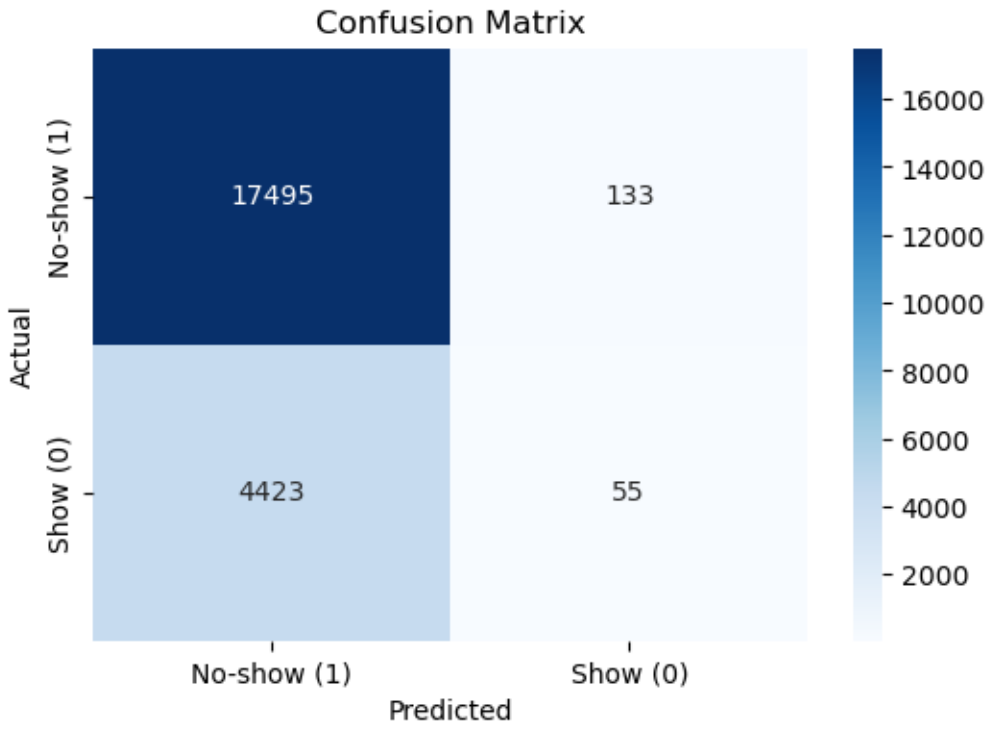


[45]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score,␣
 ↪recall_score, confusion_matrix, classification_report
from memory_profiler import memory_usage
y_pred = (a4 >= 0.14).astype(int)
y_true = y_test.values.reshape(-1, 1)
acc = accuracy_score(y_true, y_pred)
f1 = f1_score(y_true, 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}")
```

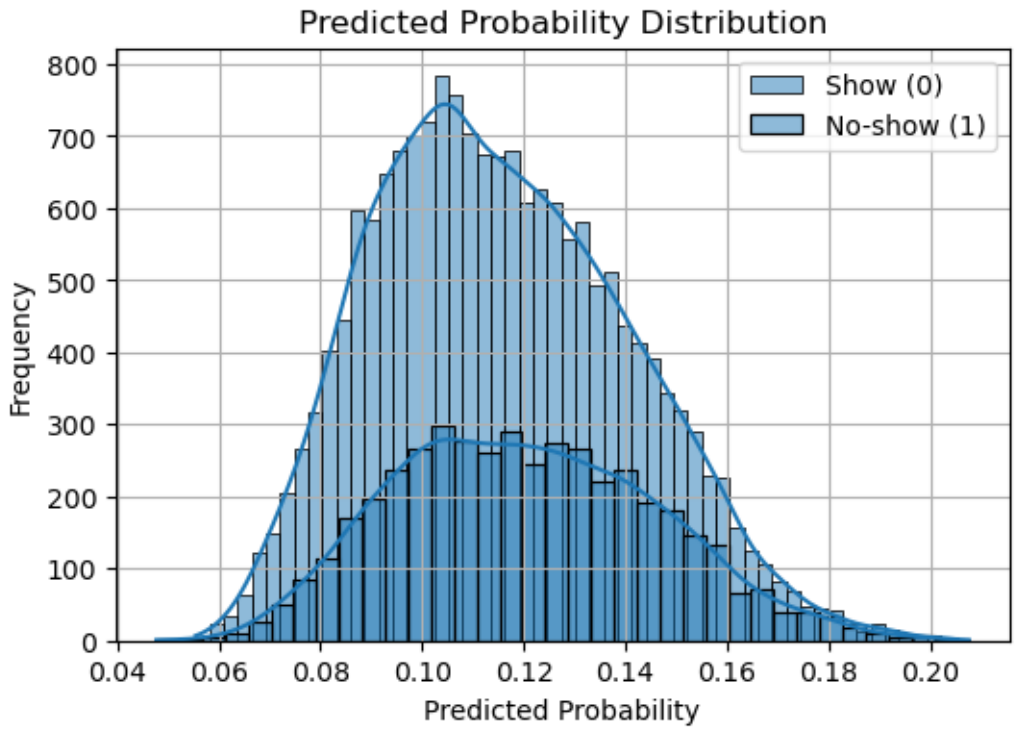
```
Accuracy: 0.7021170722880666
F1 Score: 0.24197076090710257
Precision: 0.2497030173437871
Recall: 0.2347029924073247
```

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

           0       0.81      0.82      0.81     17628
           1       0.25      0.23      0.24      4478

    accuracy                           0.70     22106
   macro avg       0.53      0.53      0.53     22106
weighted avg       0.70      0.70      0.70     22106

Time required for training process 186.40725803375244
```

[41]:
```
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()
```



[42]:
```
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
800
700
600
500
400
300
200
100
0
0.04
0.06
0.08
0.10
0.12
0.14
0.16
0.18
0.20
Predicted Probability