Sophia Godfrey

Student ID: 801149485

Homework 3

Intro to ML

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import metrics

# Load data
data = pd.read_csv("diabetes.csv")

(data.head())
```


{"summary":"{\n \"name\": \"(data\",\n \"rows\": 5,\n \"fields\":
[\n   {\n   \"column\": \"Pregnancies\",\n   \"properties\": {\
n   \"dtype\": \"number\",\n   \"std\": 3,\n   \"min\":
0,\n   \"max\": 8,\n   \"num_unique_values\": 4,\n
\"samples\": [\n   1,\n   0,\n   6\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Glucose\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\": 41,\n
\"min\": 85,\n   \"max\": 183,\n   \"num_unique_values\":
5,\n   \"samples\": [\n   85,\n   137,\n
183\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"BloodPressure\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 12,\n   \"min\": 40,\n
\"max\": 72,\n   \"num_unique_values\": 4,\n   \"samples\":
[\n   66,\n   40,\n   72\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"SkinThickness\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
14,\n   \"min\": 0,\n   \"max\": 35,\n
\"num_unique_values\": 4,\n   \"samples\": [\n   29,\n
23,\n   35\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Insulin\",\n   \"properties\": {\n   \"dtype\": \"number\",\
n   \"std\": 76,\n   \"min\": 0,\n   \"max\": 168,\n
\"num_unique_values\": 3,\n   \"samples\": [\n   0,\n
94,\n   168\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"BMI\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 7.749387072536769,\n   \"min\": 23.3,\n   \"max\":

```
43.1,\n        \"num_unique_values\": 5,\n        \"samples\": [\n
26.6,\n          43.1,\n          23.3\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"DiabetesPedigreeFunction\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.8456568452983751,\n        \"min\": 0.167,\n        \"max\": 2.288,\
n        \"num_unique_values\": 5,\n        \"samples\": [\n
0.351,\n          2.288,\n          0.672\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Age\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 10,\n        \"min\": 21,\n
\"max\": 50,\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          31,\n          33,\n          32\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Outcome\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 0,\n
\"min\": 0,\n        \"max\": 1,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          0,\n          1\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}

X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values.reshape(-1, 1)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)

# Standardize
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Add bias term
X_train = np.hstack([np.ones((X_train.shape[0], 1)), X_train])
X_test = np.hstack([np.ones((X_test.shape[0], 1)), X_test])

# Sigmoid function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Cost function (Binary Cross-Entropy)
def compute_cost(X, y, theta):
    m = len(y)
    h = sigmoid(X @ theta)
    epsilon = 1e-5  # to prevent log(0)
    cost = (-1/m) * np.sum(y*np.log(h + epsilon) + (1-y)*np.log(1-h +
epsilon))
    return cost
```

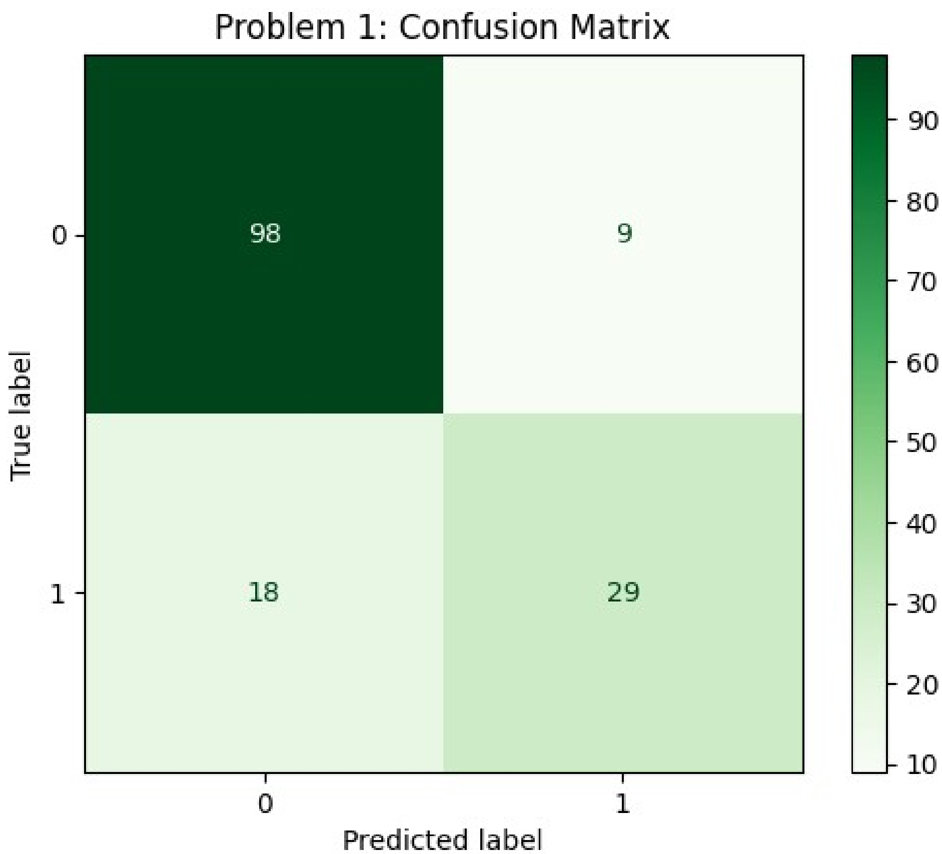
```python
# Gradient Descent
def logistic_regression(X_train, y_train, X_test, y_test, alpha=0.01,
epochs=200):
    m, n = X_train.shape
    theta = np.zeros((n, 1))
    train_costs, test_costs, train_acc, test_acc = [], [], [], []

    for i in range(epochs):
        # Predictions
        h = sigmoid(X_train @ theta)
        # Gradient
        gradient = (1/m) * X_train.T @ (h - y_train)
        theta -= alpha * gradient

        # Record cost and accuracy
        train_cost = compute_cost(X_train, y_train, theta)
        test_cost = compute_cost(X_test, y_test, theta)
        train_pred = (sigmoid(X_train @ theta) >= 0.5).astype(int)
        test_pred = (sigmoid(X_test @ theta) >= 0.5).astype(int)
        train_acc.append(metrics.accuracy_score(y_train, train_pred))
        test_acc.append(metrics.accuracy_score(y_test, test_pred))
        train_costs.append(train_cost)
        test_costs.append(test_cost)

    return theta, train_costs, test_costs, train_acc, test_acc

# Train model
theta, train_costs, test_costs, train_acc, test_acc =
logistic_regression(X_train, y_train, X_test, y_test, alpha=0.1,
epochs=250)

# Final predictions
y_pred = (sigmoid(X_test @ theta) >= 0.5).astype(int)

# Metrics
print("\nProblem 1: Diabetes Logistic Regression Results")
print(f"Accuracy:  {metrics.accuracy_score(y_test, y_pred):.4f}")
print(f"Precision: {metrics.precision_score(y_test, y_pred):.4f}")
print(f"Recall:    {metrics.recall_score(y_test, y_pred):.4f}")
print(f"F1 Score:  {metrics.f1_score(y_test, y_pred):.4f}")
```

```
Problem 1: Diabetes Logistic Regression Results
Accuracy:  0.8247
Precision: 0.7632
Recall:    0.6170
F1 Score:  0.6824
```

```python
# Confusion Matrix
disp = metrics.ConfusionMatrixDisplay.from_predictions(y_test, y_pred,
```

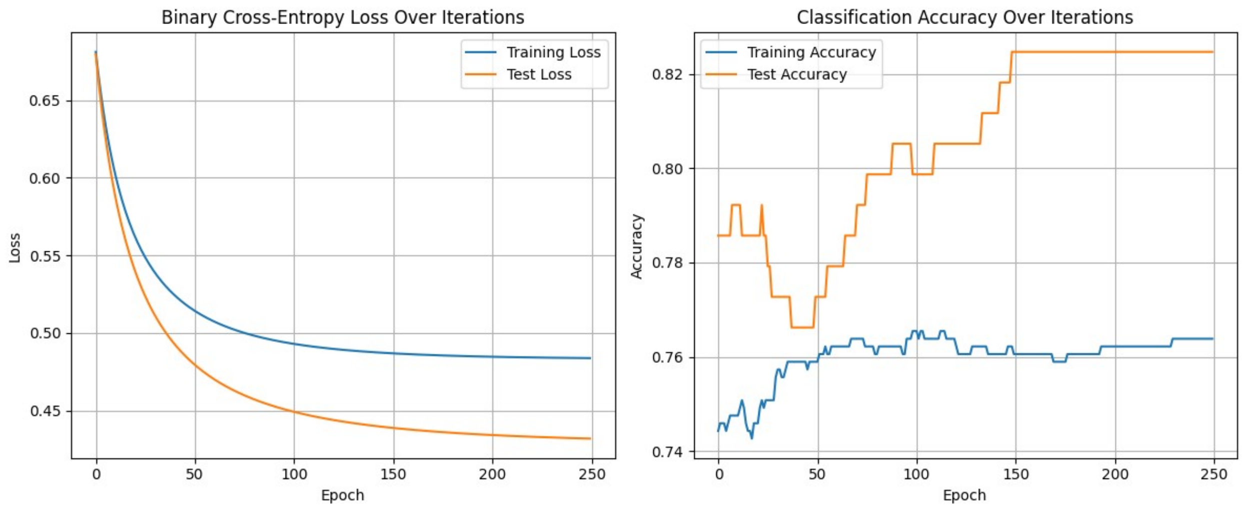
```
cmap='Greens')
disp.ax_.set_title("Problem 1: Confusion Matrix")
```

```
Text(0.5, 1.0, 'Problem 1: Confusion Matrix')
```




```
# Plot cost and accuracy
fig, ax = plt.subplots(1, 2, figsize=(12, 5))
ax[0].plot(train_costs, label='Training Loss')
ax[0].plot(test_costs, label='Test Loss')
ax[0].set_title("Binary Cross-Entropy Loss Over Iterations")
ax[0].set_xlabel("Epoch")
ax[0].set_ylabel("Loss")
ax[0].legend()
ax[0].grid(True)

ax[1].plot(train_acc, label='Training Accuracy')
ax[1].plot(test_acc, label='Test Accuracy')
ax[1].set_title("Classification Accuracy Over Iterations")
ax[1].set_xlabel("Epoch")
ax[1].set_ylabel("Accuracy")
ax[1].legend()
ax[1].grid(True)
```

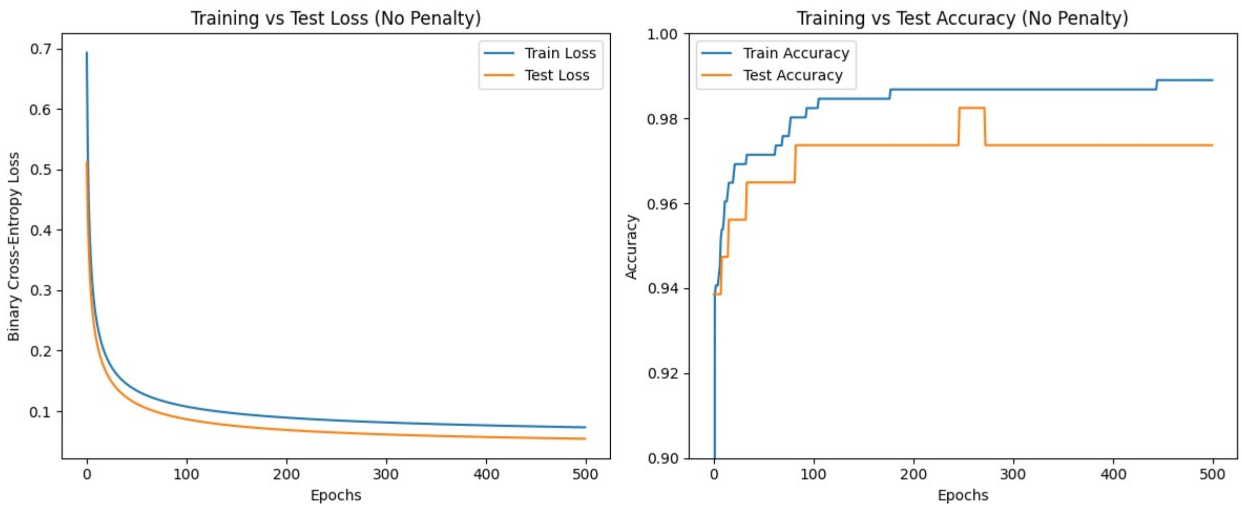
```
plt.tight_layout()
plt.show()
```



```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import metrics

# Sigmoid Function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Logistic Regression with Gradient Descent
def logistic_regression(X_train, y_train, X_test, y_test, alpha=0.1,
epochs=500, lmbda=0.0):
    m, n = X_train.shape
    theta = np.zeros((n, 1))
    y_train = y_train.reshape(-1,1)
    y_test = y_test.reshape(-1,1)

    # To record metrics
    train_cost, test_cost = [], []
    train_acc, test_acc = [], []

    for i in range(epochs):
        # Training predictions
        z_train = np.dot(X_train, theta)
        pred_train = sigmoid(z_train)
        error_train = pred_train - y_train

        # Gradient with L2 penalty
        grad = (1/m) * np.dot(X_train.T, error_train) + lmbda * theta
        theta -= alpha * grad

        # Compute Training Loss
        loss_train = -(1/m) * np.sum(
            y_train*np.log(np.clip(pred_train,1e-15,1-1e-15)) +
            (1-y_train)*np.log(np.clip(1-pred_train,1e-15,1-1e-15))
        ) + (lmbda/2)*np.sum(theta**2)
        train_cost.append(loss_train)

        # Training Accuracy
        train_acc.append(metrics.accuracy_score(y_train,
np.round(pred_train)))

        # Compute Test Loss and Accuracy
        z_test = np.dot(X_test, theta)
        pred_test = sigmoid(z_test)
        loss_test = -(1/len(y_test)) * np.sum(
            y_test*np.log(np.clip(pred_test,1e-15,1-1e-15)) +
```

```
            (1-y_test)*np.log(np.clip(1-pred_test,1e-15,1-1e-15))
        ) + (lmbda/2)*np.sum(theta**2)
        test_cost.append(loss_test)
        test_acc.append(metrics.accuracy_score(y_test,
np.round(pred_test)))

    return theta, train_cost, test_cost, train_acc, test_acc

# Predict Function
def predict(X, theta):
    return np.round(sigmoid(np.dot(X, theta)))

# Load and Prepare Data
data = load_breast_cancer()
X = data.data
y = data.target

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Add bias term
X_scaled = np.hstack([np.ones((X_scaled.shape[0],1)), X_scaled])

# Train/Test Split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.2, random_state=8)

# Problem 2a: Logistic Regression without Penalty
theta, train_loss, test_loss, train_acc, test_acc =
logistic_regression(
    X_train, y_train, X_test, y_test, alpha=0.1, epochs=500, lmbda=0.0
)

y_pred = predict(X_test, theta)

# Plot Loss & Accuracy
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(train_loss, label="Train Loss")
plt.plot(test_loss, label="Test Loss")
plt.xlabel("Epochs")
plt.ylabel("Binary Cross-Entropy Loss")
plt.title("Training vs Test Loss (No Penalty)")
plt.legend()

plt.subplot(1,2,2)
plt.plot(train_acc, label="Train Accuracy")
plt.plot(test_acc, label="Test Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
```

```
plt.title("Training vs Test Accuracy (No Penalty)")
plt.ylim(0.9, 1.0)
plt.legend()
plt.tight_layout()
plt.show()
```

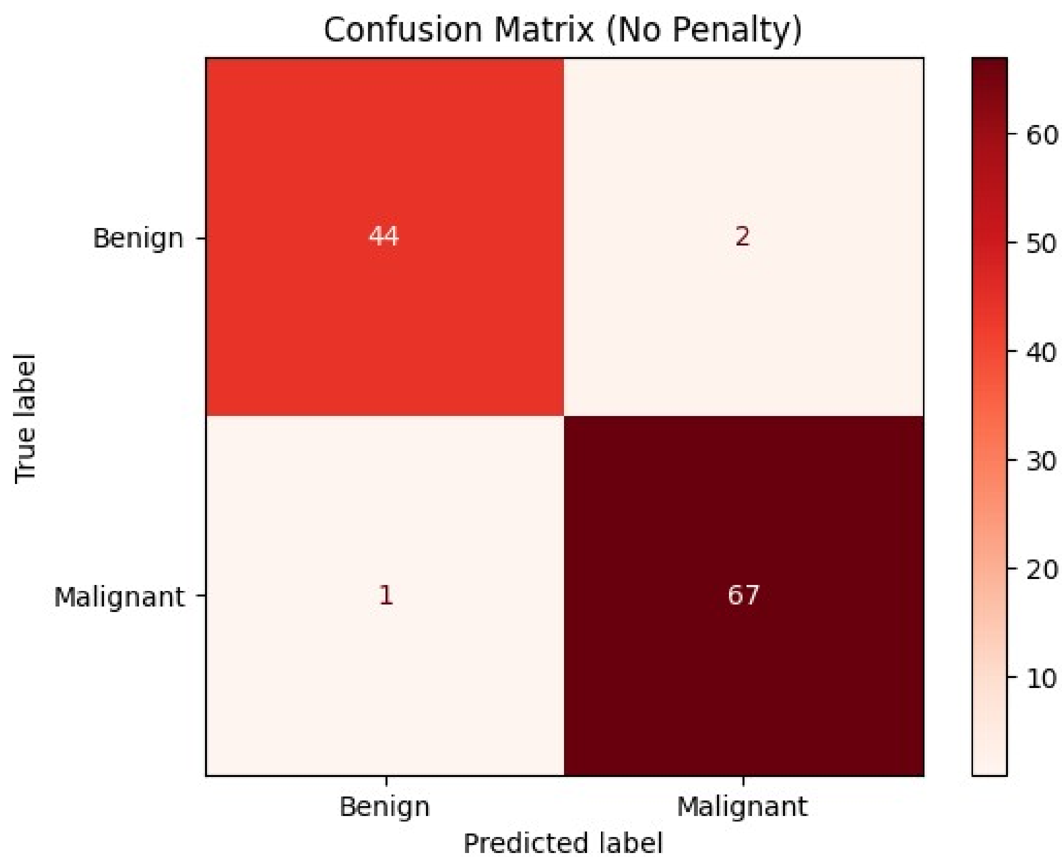



```
# Metrics
print("Problem 2a: Logistic Regression Metrics (No Penalty)")
print(f"Accuracy:  {metrics.accuracy_score(y_test, y_pred):.4f}")
print(f"Precision: {metrics.precision_score(y_test, y_pred):.4f}")
print(f"Recall:    {metrics.recall_score(y_test, y_pred):.4f}")
print(f"F1 Score:  {metrics.f1_score(y_test, y_pred):.4f}")
```

```
Problem 2a: Logistic Regression Metrics (No Penalty)
Accuracy:  0.9737
Precision: 0.9710
Recall:    0.9853
F1 Score:  0.9781
```

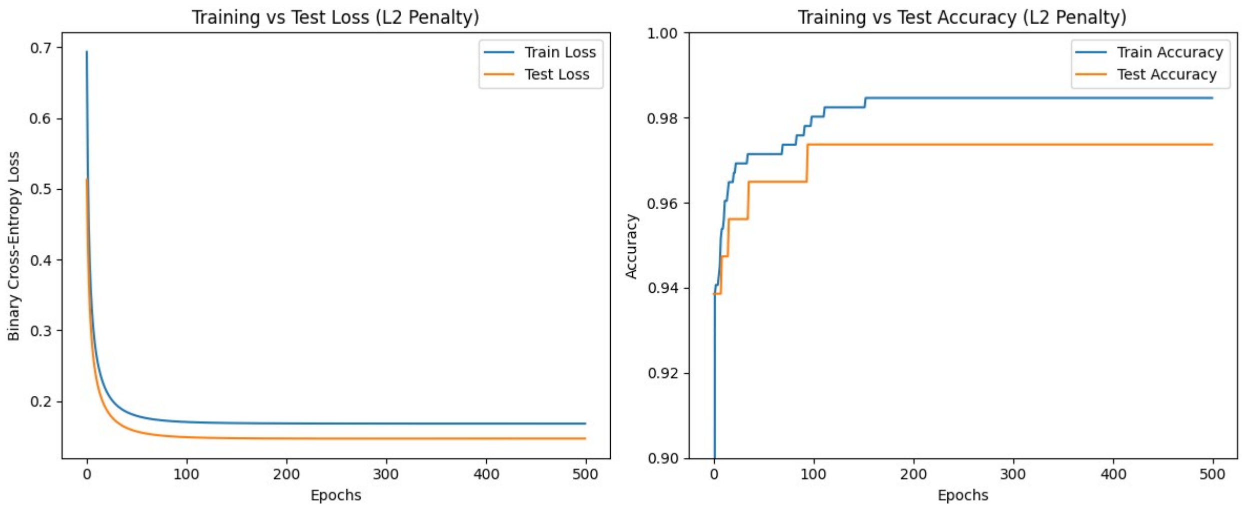
```
# Confusion Matrix
disp = metrics.ConfusionMatrixDisplay.from_predictions(
    y_test, y_pred, cmap='Reds', display_labels=['Benign',
'Malignant']
)
disp.ax_.set_title("Confusion Matrix (No Penalty)")
```

```
Text(0.5, 1.0, 'Confusion Matrix (No Penalty)')
```




```
# Problem 2b: Logistic Regression with L2 Penalty
theta_pen, train_loss_pen, test_loss_pen, train_acc_pen, test_acc_pen
= logistic_regression(
    X_train, y_train, X_test, y_test, alpha=0.1, epochs=500,
lmbda=0.05
)

y_pred_pen = predict(X_test, theta_pen)

# Plot Loss & Accuracy
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(train_loss_pen, label="Train Loss")
plt.plot(test_loss_pen, label="Test Loss")
plt.xlabel("Epochs")
plt.ylabel("Binary Cross-Entropy Loss")
plt.title("Training vs Test Loss (L2 Penalty)")
plt.legend()

plt.subplot(1,2,2)
plt.plot(train_acc_pen, label="Train Accuracy")
plt.plot(test_acc_pen, label="Test Accuracy")
plt.xlabel("Epochs")
```

```python
plt.ylabel("Accuracy")
plt.title("Training vs Test Accuracy (L2 Penalty)")
plt.ylim(0.9, 1.0)
plt.legend()
plt.tight_layout()
plt.show()
```

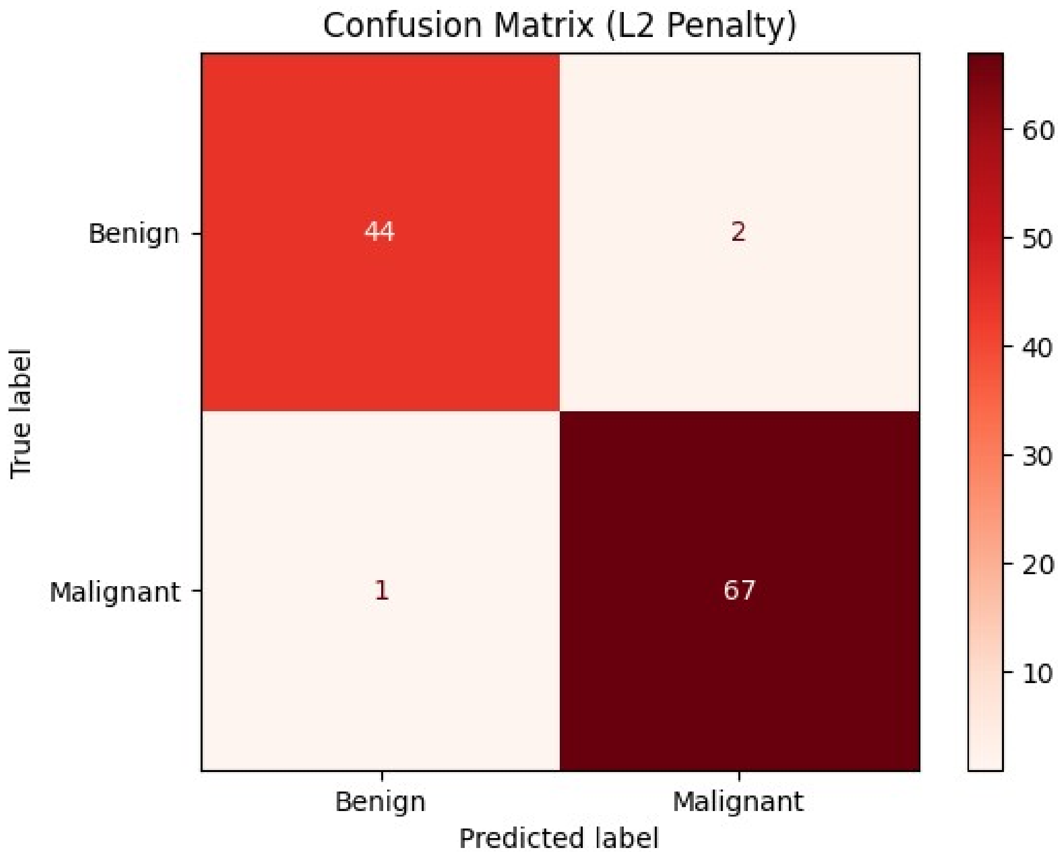



```python
# Metrics
print("\nProblem 2b: Logistic Regression Metrics (L2 Penalty)")
print(f"Accuracy:  {metrics.accuracy_score(y_test, y_pred_pen):.4f}")
print(f"Precision: {metrics.precision_score(y_test, y_pred_pen):.4f}")
print(f"Recall:    {metrics.recall_score(y_test, y_pred_pen):.4f}")
print(f"F1 Score:  {metrics.f1_score(y_test, y_pred_pen):.4f}")
```

```
Problem 2b: Logistic Regression Metrics (L2 Penalty)
Accuracy:  0.9737
Precision: 0.9710
Recall:    0.9853
F1 Score:  0.9781
```

```python
# Confusion Matrix
disp = metrics.ConfusionMatrixDisplay.from_predictions(
    y_test, y_pred_pen, cmap='Reds', display_labels=['Benign', 
'Malignant']
)
disp.ax_.set_title("Confusion Matrix (L2 Penalty)")
```

```
Text(0.5, 1.0, 'Confusion Matrix (L2 Penalty)')
```


Confusion Matrix (L2 Penalty)
Benign
Malignant
44
2
1
67
True label
Predicted label
Benign
Malignant
60
50
40
30
20
10