Intro ML Homework 3

Name: Jaskin Kabir

Student ID: 801186717

Github: https://github.com/jaskinkabir/Intro_ML/tree/main/HM3

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl




path = 'diabetes.csv'

diabetes = pd.read_csv(path)
print(diabetes.head())
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6   
1            1       85             66             29        0  26.6   
2            8      183             64              0        0  23.3   
3            1       89             66             23       94  28.1   
4            0      137             40             35      168  43.1   

   DiabetesPedigreeFunction  Age  Outcome  
0                     0.627   50        1  
1                     0.351   31        0  
2                     0.672   32        1  
3                     0.167   21        0  
4                     2.288   33        1  
```

In [2]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X_data = diabetes.iloc[:, :-1].values
Y_data = diabetes.iloc[:, -1].values
X_train, X_test, Y_train, Y_test = train_test_split(X_data, Y_data, test_size=0.2,

scaler = StandardScaler()
scaler.fit(X_data)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

In [3]:
```
def gen_data(df: pd.DataFrame):
    if isinstance(df, pd.DataFrame):
        data = df.to_numpy()
```

```
    data = df
    X0 = np.ones((data.shape[0], 1))
    X = np.hstack((X0, data))
    return X
X_train = gen_data(X_train)
X_test = gen_data(X_test)
```

In [4]:

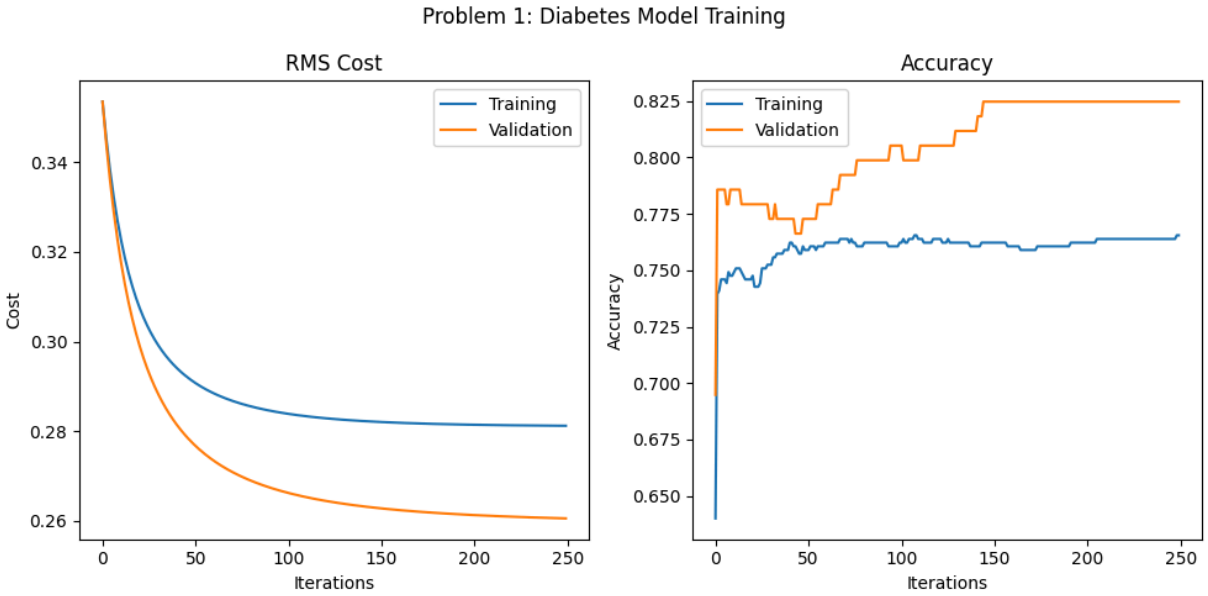
```
from sklearn import metrics


class Classifier:
    def __init__(self, X, Y, scaler, test_size = 0.2):
        self.X = X
        self.Y = Y
        self.scaler = scaler
        self.X = self.scaler.fit_transform(self.X)

        self.X_train, self.X_test, self.Y_train, self.Y_test = train_test_split(sel
        self.X_train = self.gen_data(self.X_train)
        self.X_test = self.gen_data(self.X_test)

        self.Y_train = self.Y_train.reshape(-1, 1)
        self.Y_test = self.Y_test.reshape(-1, 1)
    def predict(self, X):
        pass

    def gen_data(self, df: pd.DataFrame):
        if isinstance(df, pd.DataFrame):
            data = df.to_numpy()
        data = df
        X0 = np.ones((data.shape[0], 1))
        X = np.hstack((X0, data))
        return X
    def train(self):
        pass
    def validate(self):
        return [self.Y_test, np.round(self.predict(self.X_test))]

class LogisticRegression(Classifier):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

    def predict(self, X):
        return 1 / (1 + np.exp(-np.dot(X, self.theta)))



    def plot_cost(self, title, ax: plt.Axes):

        ax.plot(self.training_history, label='Training')
        ax.plot(self.test_history, label='Validation')
        ax.set_title(title)
        ax.set_xlabel('Iterations')
        ax.set_ylabel('Cost')
        ax.legend()
```

```
    def plot_accuracy(self, title, ax: plt.Axes):
        ax.plot(self.training_accuracy, label='Training')
        ax.plot(self.test_accuracy, label='Validation')
        ax.set_title(title)
        ax.set_xlabel('Iterations')
        ax.set_ylabel('Accuracy')
        ax.legend()

    def train(self, lmbda, alpha, epochs):
        self.training_history = []
        self.test_history = []
        self.training_accuracy = []
        self.test_accuracy = []
        self.theta = np.zeros((self.X.shape[1]+1,1))
        for _ in range(epochs):
            pred = self.predict(self.X_train)
            test_pred = self.predict(self.X_test)
            error = np.subtract(pred, self.Y_train)
            test_error = test_pred - self.Y_test
            gradient = (1/self.X_train.shape[0]) * np.dot(self.X_train.T, error) +
            self.theta -= alpha * gradient

            J = np.sqrt(np.sum(error ** 2) / (2 * self.X_train.shape[0]))
            J_test = np.sqrt(np.sum(test_error ** 2) / (2 * self.X_test.shape[0]))

            self.training_history.append(J)
            self.test_history.append(J_test)

            self.training_accuracy.append(metrics.accuracy_score(self.Y_train, np.r
            self.test_accuracy.append(metrics.accuracy_score(self.Y_test, np.round(
```

In [5]:

```
Diabetes_Predictor = LogisticRegression(X_data, Y_data, StandardScaler(),test_size=
Diabetes_Predictor.train(lmbda=0.0, alpha=0.1, epochs=250)
print(f"Train cost: {Diabetes_Predictor.training_history[-1]}")
print(f"Test cost: {Diabetes_Predictor.test_history[-1]}")

mpl.rcParams['figure.figsize'] = [10,5]

fig, ax = plt.subplots(1,2,sharex=True, squeeze=True)
fig.suptitle("Problem 1: Diabetes Model Training")

Diabetes_Predictor.plot_cost('RMS Cost', ax[0])

Diabetes_Predictor.plot_accuracy('Accuracy', ax[1])
plt.tight_layout()
```

```
Train cost: 0.2811995752702271
Test cost: 0.26052563801592754
```

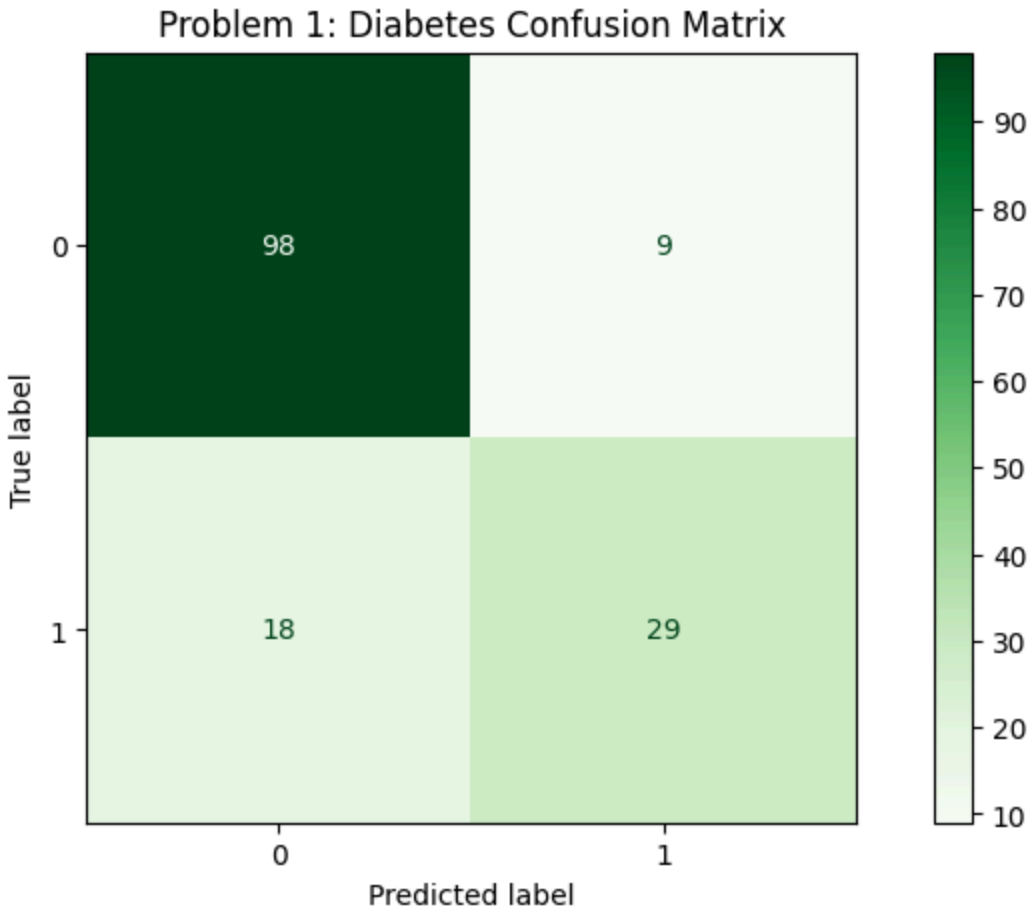


In [6]:
```
Y_pred = np.round(Diabetes_Predictor.predict(Diabetes_Predictor.X_test))
print("Problem 1: Diabetes Classifier Metrics\n")
print(f"Accuracy: {metrics.accuracy_score(Diabetes_Predictor.Y_test, Y_pred)}")
print(f"Precision: {metrics.precision_score(Diabetes_Predictor.Y_test, Y_pred)}")
print(f"Recall: {metrics.recall_score(Diabetes_Predictor.Y_test, Y_pred)}")
print(f"F1 Score: {metrics.f1_score(Diabetes_Predictor.Y_test, Y_pred)}")
```

```
Problem 1: Diabetes Classifier Metrics

Accuracy: 0.8246753246753247
Precision: 0.7631578947368421
Recall: 0.6170212765957447
F1 Score: 0.6823529411764706
```

In [7]:
```
disp = metrics.ConfusionMatrixDisplay.from_predictions(
    y_true = Diabetes_Predictor.Y_test,
    y_pred = Y_pred,
    cmap = 'Greens'

)

disp.ax_.set_title("Problem 1: Diabetes Confusion Matrix")
#disp.plot()
```

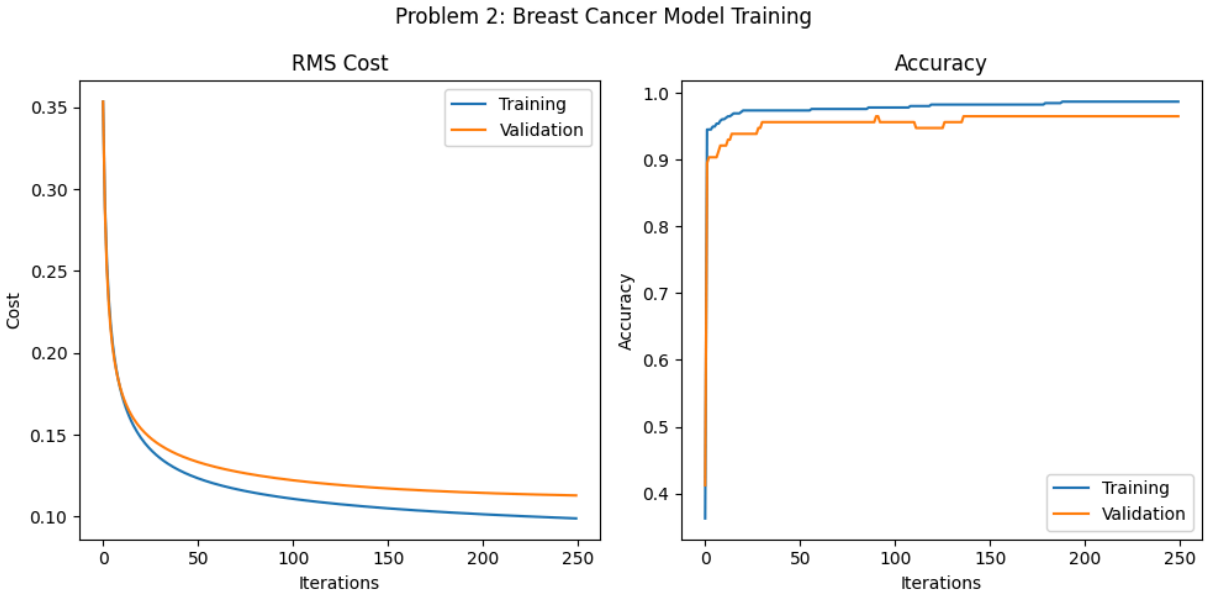
Out[7]:
```
Text(0.5, 1.0, 'Problem 1: Diabetes Confusion Matrix')
```



In [8]:
```
# from sklearn.linear_model import LogisticRegression

# sklearn_model = LogisticRegression(penalty='l1', solver = 'liblinear', max_iter=2
# sklearn_model.fit(X_train, Y_train)
# Y_pred = sklearn_model.predict(X_test)

# print("Sklearn's Logistic Regression")
# print(f"Accuracy: {metrics.accuracy_score(Y_test, Y_pred)}")
# print(f"Precision: {metrics.precision_score(Y_test, Y_pred)}")
# print(f"Recall: {metrics.recall_score(Y_test, Y_pred)}")
# print(f"F1 Score: {metrics.f1_score(Y_test, Y_pred)}")
# #confusion matrix
# print(metrics.confusion_matrix(Y_test, Y_pred))
```

In [9]:
```
from sklearn.datasets import load_breast_cancer


breast = load_breast_cancer()
Cancer_Predictor = LogisticRegression(breast.data, breast.target, StandardScaler(),
Cancer_Predictor.train(lmbda=0.0, alpha=0.1, epochs=250)
#No lambda could be found that improves metrics


fig, ax = plt.subplots(1,2,sharex=True, squeeze=True)
fig.suptitle("Problem 2: Breast Cancer Model Training")
Cancer_Predictor.plot_cost('RMS Cost', ax[0])
Cancer_Predictor.plot_accuracy('Accuracy', ax[1])
```

```
plt.tight_layout()

Y_pred_no_penalty = np.round(Cancer_Predictor.predict(Cancer_Predictor.X_test))
f1_score_no_penalty = metrics.f1_score(Cancer_Predictor.Y_test, Y_pred_no_penalty)
```



In [10]:
```
Cancer_Predictor.train(lmbda=0.05, alpha=0.1, epochs=250)
Y_pred_penalty = np.round(Cancer_Predictor.predict(Cancer_Predictor.X_test))
f1_score_penalty = metrics.f1_score(Cancer_Predictor.Y_test, Y_pred_penalty)

cancer_metrics = np.array([
    [metrics.accuracy_score(Cancer_Predictor.Y_test, Y_pred_no_penalty), metrics.ac
    [metrics.precision_score(Cancer_Predictor.Y_test, Y_pred_no_penalty), metrics.p
    [metrics.recall_score(Cancer_Predictor.Y_test, Y_pred_no_penalty), metrics.reca
    [f1_score_no_penalty, f1_score_penalty]
])


cancer_percent_improvements = ((cancer_metrics[:,1] - cancer_metrics[:,0]) / cancer
print(cancer_percent_improvements)
Cancer_Predictor.train(lmbda=0, alpha=0.1, epochs=250)
```

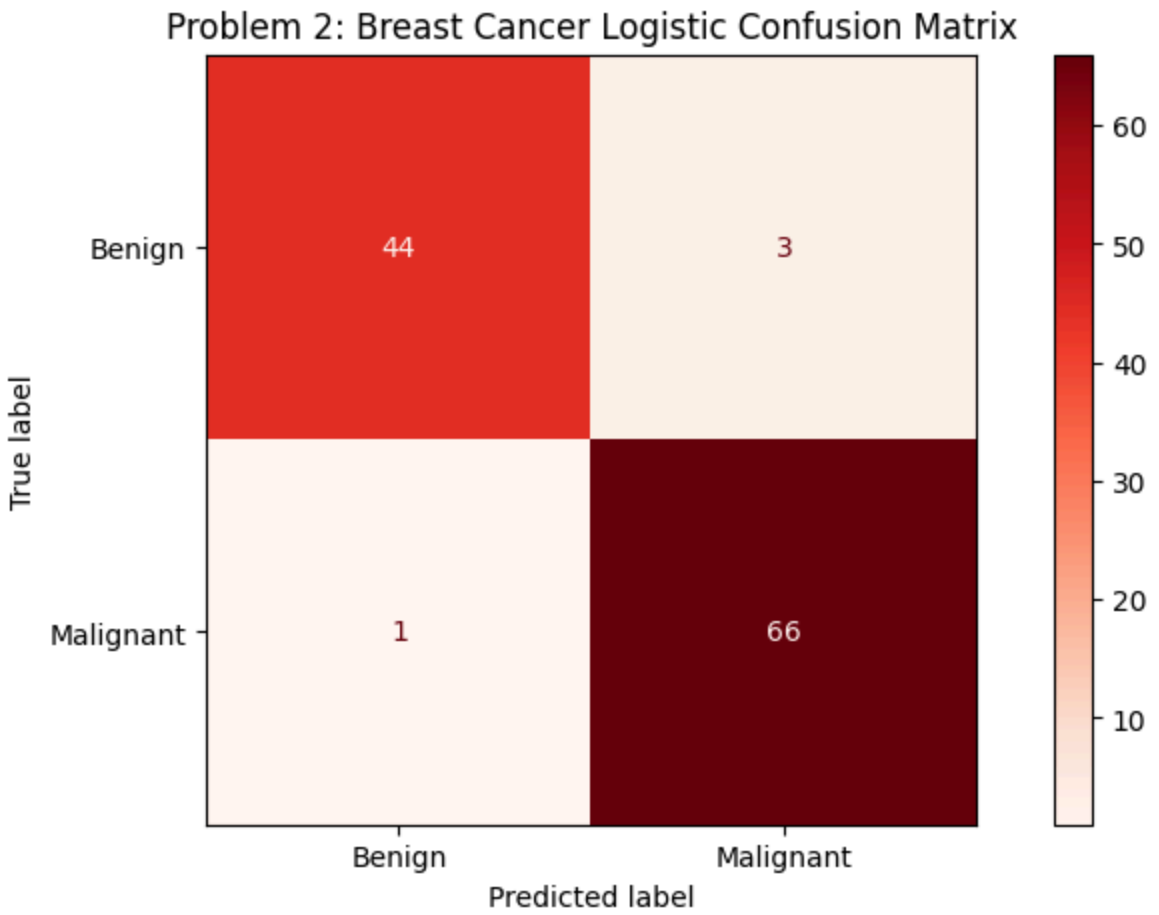
```
[-0.90909091 -1.42857143  0.         -0.72992701]
```

In [11]:
```
Y_test, Y_pred = Cancer_Predictor.validate()
p2metrics = np.array([
    metrics.accuracy_score(Y_test, Y_pred),
    metrics.precision_score(Y_test, Y_pred),
    metrics.recall_score(Y_test, Y_pred),
    metrics.f1_score(Y_test, Y_pred)
])
print("Problem 2: Breast Cancer Classifier Metrics\n")
print(metrics.classification_report(Y_test, Y_pred))
# print(f"Accuracy: {metrics.accuracy_score(Cancer_Predictor.Y_test, Y_pred)}")
# print(f"Precision: {metrics.precision_score(Cancer_Predictor.Y_test, Y_pred)}")
# print(f"Recall: {metrics.recall_score(Cancer_Predictor.Y_test, Y_pred)}")
# print(f"F1 Score: {metrics.f1_score(Cancer_Predictor.Y_test, Y_pred)}")
```

```
Problem 2: Breast Cancer Classifier Metrics

              precision    recall  f1-score   support

           0       0.98      0.94      0.96        47
           1       0.96      0.99      0.97        67

    accuracy                           0.96       114
   macro avg       0.97      0.96      0.96       114
weighted avg       0.97      0.96      0.96       114
```

In [12]:
```
#Replace 0 with benign 1 with malignant
#print(y_true[:10])

disp = metrics.ConfusionMatrixDisplay.from_predictions(
    y_true = Cancer_Predictor.Y_test,
    y_pred = Y_pred,
    cmap = 'Reds',
    display_labels = ['Benign', 'Malignant']
)
disp.ax_.set_title("Problem 2: Breast Cancer Logistic Confusion Matrix")
```

Out[12]: Text(0.5, 1.0, 'Problem 2: Breast Cancer Logistic Confusion Matrix')



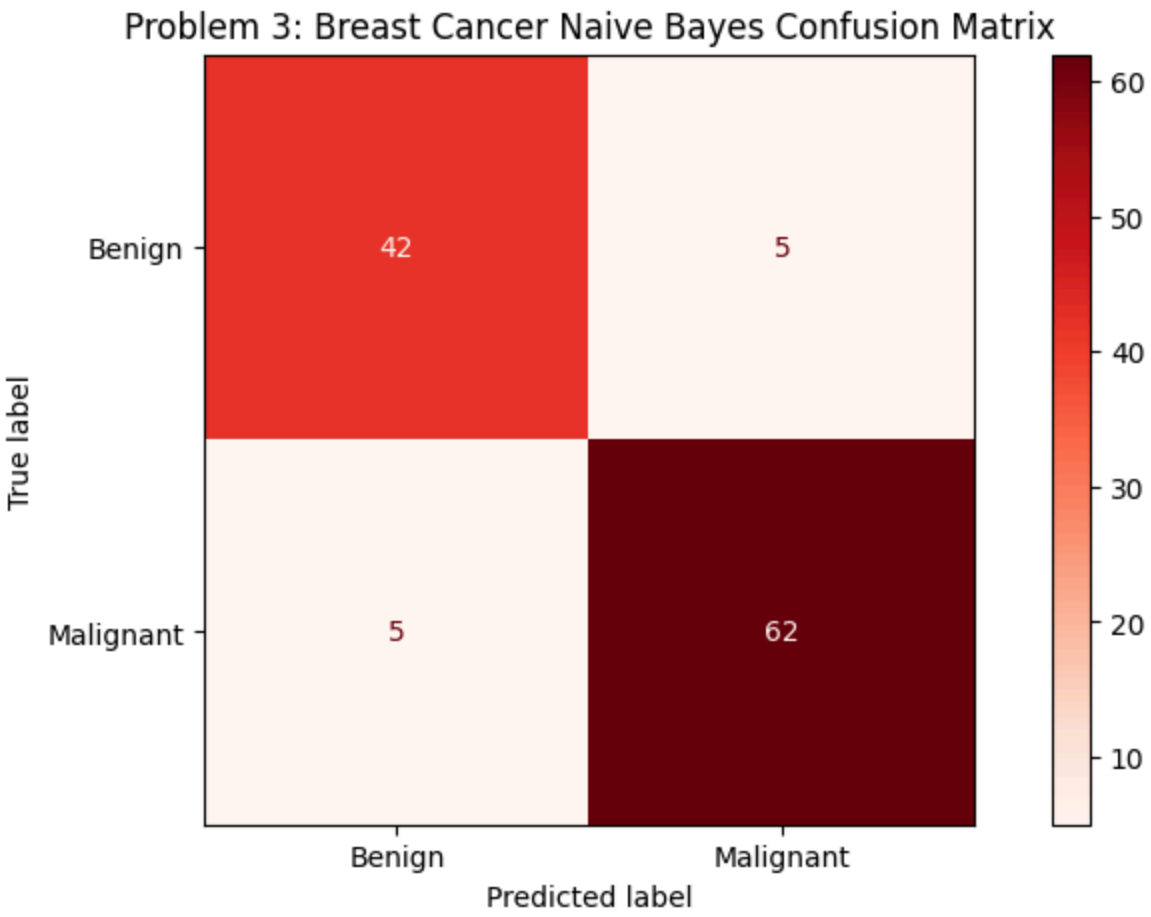
In [13]:
```
class NGBClassifier(Classifier):
    def __init__(self, *args, **kwargs):
        super(NGBClassifier, self).__init__(*args, **kwargs)
```

```
        self.J = self.X.shape[1]
        self.K = max(self.Y)+1 #max = 3, 4 total classes
        self.double_variances = np.zeros((self.J,self.K))
        self.means = np.zeros((self.J,self.K))
        self.invsqrs = np.zeros((self.J,self.K))
        self.log_priors = np.zeros(self.K)


    def train(self):
        training_data = np.hstack((self.X_train, self.Y_train))

        for k in range(0,self.K):
            xk = training_data[training_data[:,-1] == k]
            self.log_priors[k] = np.log( xk.shape[0] / training_data.shape[0] )

            for j in range(0,self.J):
                xjk = xk[:,j]
                self.double_variances[j,k]=2*np.var(xjk)
                if (self.double_variances[j,k]==0):
                    self.double_variances[j,k] = 0.0000000000000000000001
                self.means[j,k] = np.mean(xjk)
                self.invsqrs[j,k] = 1/np.sqrt(np.pi*self.double_variances[j,k])
    def predict(self, X: np.ndarray):
        Y = np.zeros(X.shape[0])

        for i, x in enumerate(X):
            pvck = np.zeros(self.K)
            for k in range(0,self.K):
                cur_sum = self.log_priors[k]
                for j in range(0,self.J):
                    cur_sum += np.log( self.invsqrs[j,k] * np.exp(-np.square(x[j]-s
                pvck[k] = cur_sum
            Y[i] = int(np.argmax(pvck))
        return Y.astype(int)


ngb = NGBClassifier(breast.data, breast.target, StandardScaler())
ngb.train()
Y_test, Y_pred = ngb.validate()


print(metrics.classification_report(Y_test, Y_pred))
p3metrics = np.array([
    metrics.accuracy_score(Y_test, Y_pred),
    metrics.precision_score(Y_test, Y_pred),
    metrics.recall_score(Y_test, Y_pred),
    metrics.f1_score(Y_test, Y_pred)
])
disp = metrics.ConfusionMatrixDisplay.from_predictions(
    y_true = ngb.Y_test,
    y_pred = Y_pred,
    cmap = 'Reds',
    display_labels = ['Benign', 'Malignant']
)
```

```
disp.ax_.set_title("Problem 3: Breast Cancer Naive Bayes Confusion Matrix")
```

```
              precision    recall  f1-score   support

           0       0.89      0.89      0.89        47
           1       0.93      0.93      0.93        67

    accuracy                           0.91       114
   macro avg       0.91      0.91      0.91       114
weighted avg       0.91      0.91      0.91       114
```

Out[13]: Text(0.5, 1.0, 'Problem 3: Breast Cancer Naive Bayes Confusion Matrix')



In [14]:
```
p3_2_improvements = (1/p2metrics)*(p3metrics-p2metrics)*100
print(p3_2_improvements)
```

```
[-5.45454545 -3.25644505 -6.06060606 -4.65852555]
```

In [ ]:

In [ ]:

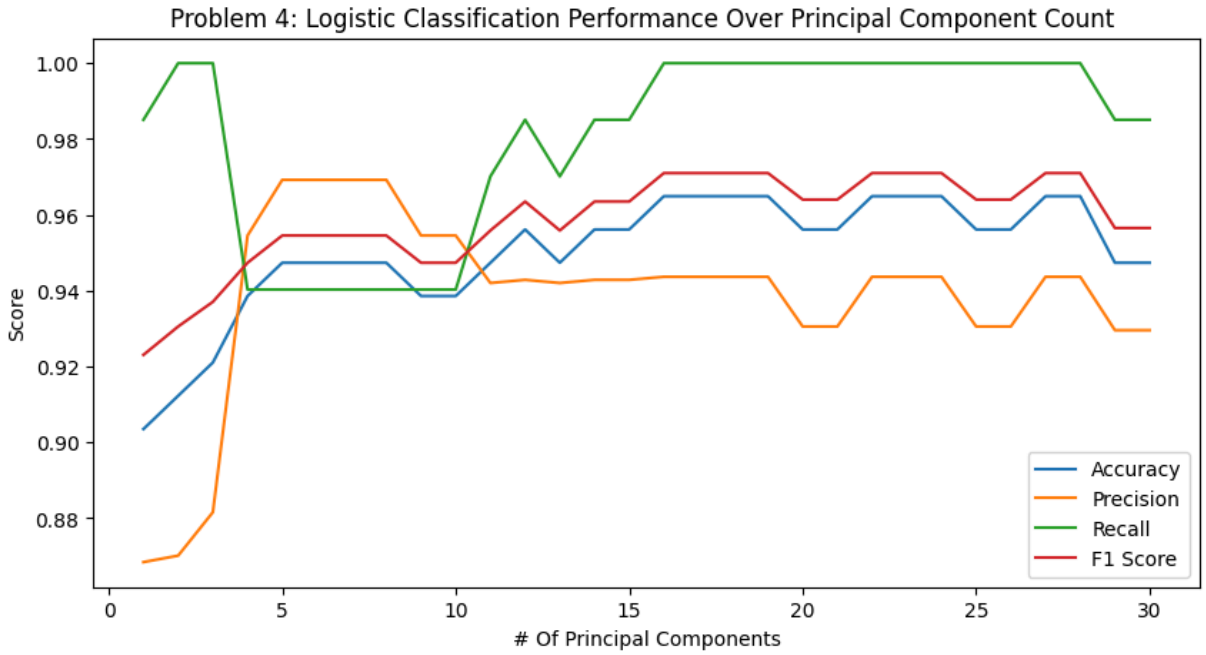
In [15]:
```
from sklearn.naive_bayes import GaussianNB
ngb_sk = GaussianNB()
Y_pred = ngb_sk.fit(ngb.X_train, np.ravel(ngb.Y_train)).predict(ngb.X_test)
print(metrics.classification_report(ngb.Y_test, Y_pred))
```

```
print(metrics.confusion_matrix(ngb.Y_test, Y_pred))
# My bayes classifier is a tiny bit better for some reason
```

```
              precision    recall  f1-score   support

           0       0.88      0.89      0.88        47
           1       0.92      0.91      0.92        67

    accuracy                           0.90       114
   macro avg       0.90      0.90      0.90       114
weighted avg       0.90      0.90      0.90       114

[[42  5]
 [ 6 61]]
```

In [16]:
```
from sklearn.decomposition import PCA
K=30
accuracy_hist = np.zeros(K)
precision_hist = np.zeros(K)
recall_hist = np.zeros(K)
f1_hist = np.zeros(K)

for k in range(1,K+1):
    extraction = PCA(n_components=k)
    X_data = extraction.fit_transform(breast.data)
    Y_data = breast.target

    model = LogisticRegression(X_data, Y_data, StandardScaler(), test_size=0.2)
    model.train(lmbda=0.0, alpha=0.1, epochs=250)

    Y_test, Y_pred = model.validate()
    accuracy_hist[k-1] = metrics.accuracy_score(Y_test, Y_pred)
    precision_hist[k-1] = metrics.precision_score(Y_test, Y_pred)
    recall_hist[k-1] = metrics.recall_score(Y_test, Y_pred)
    f1_hist[k-1] = metrics.f1_score(Y_test, Y_pred)
```

In [17]:
```
K = range(1,k+1)
plt.title("Problem 4: Logistic Classification Performance Over Principal Component
plt.xlabel("# Of Principal Components")
plt.ylabel("Score")
plt.plot(K,accuracy_hist, label='Accuracy')
plt.plot(K,precision_hist, label = 'Precision')
plt.plot(K,recall_hist, label = 'Recall')
plt.plot(K,f1_hist, label = 'F1 Score')

plt.legend()
plt.show()
print(f'Max precision at K = {np.argmax(precision_hist)+1}; {max(precision_hist)}')
print(f"Max accuracy at K = {np.argmax(accuracy_hist)+1}; {max(accuracy_hist)}")
print(f"Max recall at K = {np.argmax(recall_hist)+1}; {max(recall_hist)}")
print(f"Max F1 Score at K = {np.argmax(f1_hist)+1}; {max(f1_hist)}")
#Ideal k=16
```




```
Max precision at K = 5; 0.9692307692307692
Max accuracy at K = 16; 0.9649122807017544
Max recall at K = 2; 1.0
Max F1 Score at K = 16; 0.9710144927536231
```

In [18]:
```
pca = PCA(n_components=16)
Y_data = breast.target
X_data = pca.fit_transform(breast.data)
Y_data = breast.target

model = LogisticRegression(X_data, Y_data, StandardScaler(), test_size=0.2)
model.train(lmbda=0.0, alpha=0.1, epochs=250)

Y_test, Y_pred = model.validate()
p4metrics = np.array([
    metrics.accuracy_score(Y_test, Y_pred),
    metrics.precision_score(Y_test, Y_pred),
    metrics.recall_score(Y_test, Y_pred),
    metrics.f1_score(Y_test, Y_pred)
])
print(metrics.classification_report(Y_test, Y_pred))
```

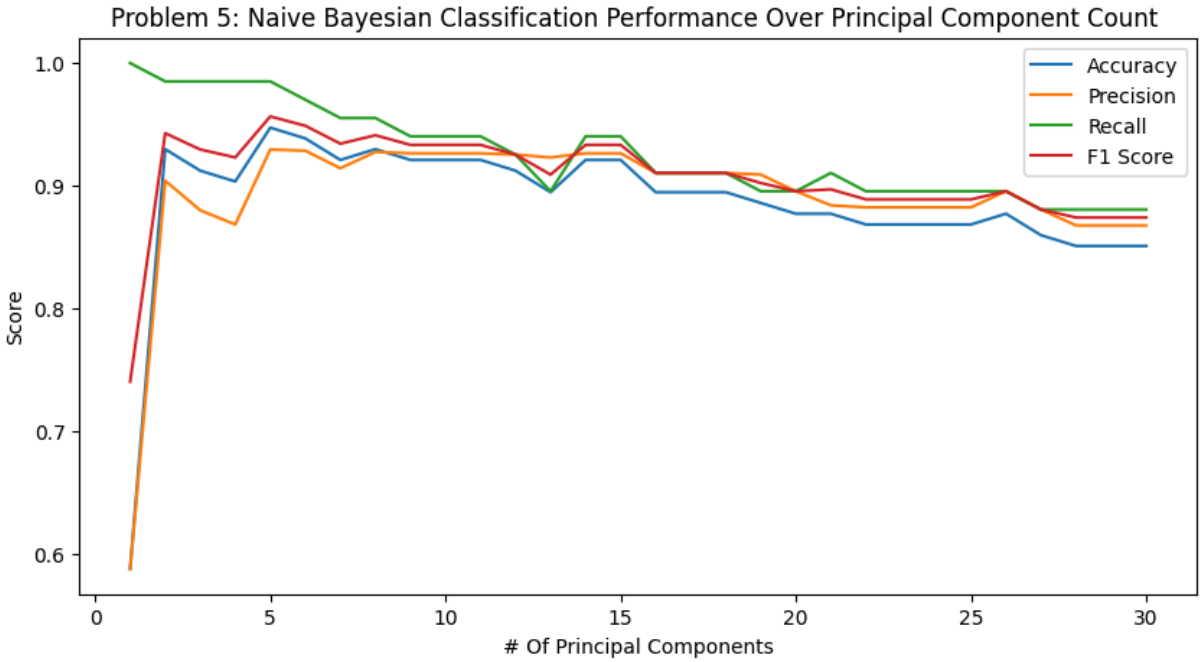
```
              precision    recall  f1-score   support

           0       1.00      0.91      0.96        47
           1       0.94      1.00      0.97        67

    accuracy                           0.96       114
   macro avg       0.97      0.96      0.96       114
weighted avg       0.97      0.96      0.96       114
```

In [26]:
```
p4_2_improvements = (1/p2metrics)*(p4metrics-p2metrics)*100
print(p4_2_improvements)
p4_3_improvements = (1/p3metrics)*(p4metrics-p3metrics)*100
print(p4_3_improvements)
```

```
[ 0.         -1.34443022  1.51515152  0.04391744]
[5.76923077 1.97637438 8.06451613 4.93221131]
```

In [20]:
```
K=30
accuracy_hist = np.zeros(K)
precision_hist = np.zeros(K)
recall_hist = np.zeros(K)
f1_hist = np.zeros(K)

for k in range(1,K+1):
    extraction = PCA(n_components=k)
    X_data = extraction.fit_transform(breast.data)
    Y_data = breast.target

    model = NGBClassifier(X_data, Y_data, StandardScaler(), test_size=0.2)
    model.train()

    Y_test, Y_pred = model.validate()
    accuracy_hist[k-1] = metrics.accuracy_score(Y_test, Y_pred)
    precision_hist[k-1] = metrics.precision_score(Y_test, Y_pred)
    recall_hist[k-1] = metrics.recall_score(Y_test, Y_pred)
    f1_hist[k-1] = metrics.f1_score(Y_test, Y_pred)
```

In [21]:
```
K = range(1,k+1)
plt.title("Problem 5: Naive Bayesian Classification Performance Over Principal Comp
plt.xlabel("# Of Principal Components")
plt.ylabel("Score")
plt.plot(K,accuracy_hist, label='Accuracy')
plt.plot(K,precision_hist, label = 'Precision')
plt.plot(K,recall_hist, label = 'Recall')
plt.plot(K,f1_hist, label = 'F1 Score')
plt.legend()
plt.show()
#Ideal k=5
print(f'Max precision at K = {np.argmax(precision_hist)+1}; {max(precision_hist)}')
print(f"Max accuracy at K = {np.argmax(accuracy_hist)+1}; {max(accuracy_hist)}")
print(f"Max recall at K = {np.argmax(recall_hist)+1}; {max(recall_hist)}")
print(f"Max F1 Score at K = {np.argmax(f1_hist)+1}; {max(f1_hist)}")
```




```
Max precision at K = 5; 0.9295774647887324
Max accuracy at K = 5; 0.9473684210526315
Max recall at K = 1; 1.0
Max F1 Score at K = 5; 0.9565217391304348
```

In [29]:
```
pca = PCA(n_components=5)
Y_data = breast.target
X_data = pca.fit_transform(breast.data)
Y_data = breast.target

model = NGBClassifier(X_data, Y_data, StandardScaler(), test_size=0.2)
model.train()

Y_test, Y_pred = model.validate()
p5metrics = np.array([
    metrics.accuracy_score(Y_test, Y_pred),
    metrics.precision_score(Y_test, Y_pred),
    metrics.recall_score(Y_test, Y_pred),
    metrics.f1_score(Y_test, Y_pred)
])
print(metrics.classification_report(Y_test, Y_pred))
p5_3_improvements = (1/p3metrics)*(p5metrics-p3metrics)*100
print(p5_3_improvements)
```

```
              precision    recall  f1-score   support

           0       0.98      0.89      0.93        47
           1       0.93      0.99      0.96        67

    accuracy                           0.95       114
   macro avg       0.95      0.94      0.94       114
weighted avg       0.95      0.95      0.95       114

[3.84615385 0.45433894 6.4516129  3.36605891]
```

In [30]:
```
p5_2_improvements = (1/p2metrics)*(p5metrics-p2metrics)*100
print(p5_2_improvements)
p5_3_improvements = (1/p3metrics)*(p5metrics-p3metrics)*100
print(p5_3_improvements)
p5_4_improvements = (1/p4metrics)*(p5metrics-p4metrics)*100
print(p5_4_improvements)
```

```
[-1.81818182 -2.81690141  0.         -1.44927536]
[3.84615385 0.45433894 6.4516129  3.36605891]
[-1.81818182 -1.49253731 -1.49253731 -1.49253731]
```