2c

In [65]:
```
# Softmax function
def softmax(X):
    exps = np.exp(X - np.max(X, axis=1, keepdims=True))
    return exps / np.sum(exps, axis=1, keepdims=True)

# Gradient descent
def gradient_descent(loss_func, X, y, learning_rate=0.5, epochs=100):
    n, d = X.shape
    k = y.shape[1]
    W = np.random.randn(d, k) / np.sqrt(d)
    for epoch in range(epochs):
        scores = np.dot(X, W)
        softmax_output = softmax(scores)
        grad = -np.dot(X.T, (y - softmax_output)) / n
        W -= learning_rate * grad
        loss = loss_func(W, X, y)
        if (epoch+1)%10==0:
            print(f"Epoch {epoch + 1}, Loss: {loss}")
    return W

# Calculate the loss for L(W)
def calculate_L_loss(W, X, y):
    scores = np.dot(X, W)
    softmax_output = softmax(scores)
    loss = -np.sum(y * np.log(softmax_output))
    return loss

def predict(W, X):
    scores = np.dot(X, W)
    return np.argmax(scores, axis=1)
```

Running Gradient Decent

In [66]:
```
# Run gradient descent for J(W)
W_J = gradient_descent(calculate_J_loss, X_train, y_train)
```

```
Epoch 10, Loss: 135.32837449812567
Epoch 20, Loss: 120.43910989043378
Epoch 30, Loss: 49.83535927525998
Epoch 40, Loss: 80.22838849472518
Epoch 50, Loss: 55.47235498585658
Epoch 60, Loss: 80.1572175880842
Epoch 70, Loss: 80.20783650954867
Epoch 80, Loss: 59.17931904961713
Epoch 90, Loss: 70.30831778137976
Epoch 100, Loss: 64.81101974623745
```

In [67]:
```
# Run gradient descent for L(W)
W_L = gradient_descent(calculate_L_loss, X_train, y_train)
```

```
Epoch 10, Loss: 380083.93463238265
Epoch 20, Loss: 96651.604971244
Epoch 30, Loss: 149879.92377353387
Epoch 40, Loss: 92413.8823929142
Epoch 50, Loss: 78148.35590485296
Epoch 60, Loss: 116359.84786588624
Epoch 70, Loss: 46970.920905421124
Epoch 80, Loss: 84640.215685404
Epoch 90, Loss: 91843.66003399061
Epoch 100, Loss: 112521.54924157326
```

Accuracy for J(W)

In [68]:
```
# Calculate accuracy for J(W)
y_train_pred_J = predict(W_J, X_train)
y_test_pred_J = predict(W_J, X_test)
```

In [69]:
```
accuracy_train_J = np.mean(y_train_pred_J == np.argmax(y_train, axis=1))
accuracy_test_J = np.mean(y_test_pred_J == np.argmax(y_test, axis=1))

print(f"Accuracy for J(W) - Training set: {accuracy_train_J * 100:.4f}%")
print(f"Accuracy for J(W) - Test set: {accuracy_test_J * 100:.4f}%")
```

```
Accuracy for J(W) - Training set: 74.7267%
Accuracy for J(W) - Test set: 73.6300%
```

Accuracy for L(W)

In [70]:
```
# Calculate accuracy for L(W)
y_train_pred_L = predict(W_L, X_train)
y_test_pred_L = predict(W_L, X_test)
```

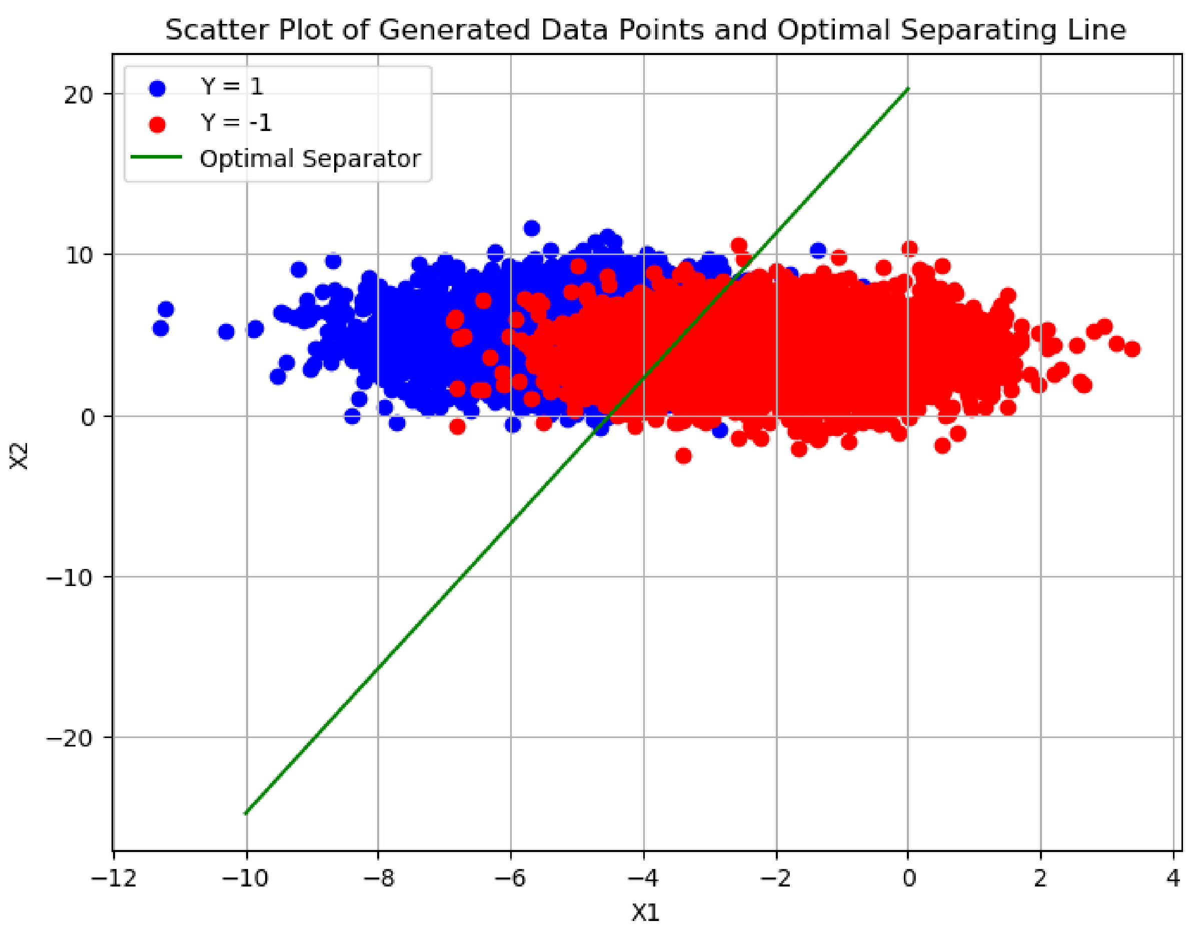
In [71]:
```
accuracy_train_L = np.mean(y_train_pred_L == np.argmax(y_train, axis=1))
accuracy_test_L = np.mean(y_test_pred_L == np.argmax(y_test, axis=1))

print(f"Accuracy for L(W) - Training set: {accuracy_train_L * 100:.4f}%")
print(f"Accuracy for L(W) - Test set: {accuracy_test_L * 100:.4f}%")
```

```
Accuracy for L(W) - Training set: 76.4183%
Accuracy for L(W) - Test set: 75.5800%
```

3b

In [2]:
```
# Given parameters
n = 5000
mu_1 = np.array([-5, 5])
mu_neg1 = np.array([-2, 4])
sigma = np.array([[2, 0], [0, 3]])
w_star = np.array([-1.5, 1/3])
b_star = -6.75
```

In [3]:
```
# Generate data points
X_1 = np.random.multivariate_normal(mu_1, sigma, n)
X_minus1 = np.random.multivariate_normal(mu_neg1, sigma, n)

# Scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(X_1[:, 0], X_1[:, 1], c='blue', label='Y = 1')
plt.scatter(X_minus1[:, 0], X_minus1[:, 1], c='red', label='Y = -1')

# Plot the optimal separating line
x_vals = np.linspace(-10, 0, 100)
y_vals = (-w_star[0] * x_vals - b_star) / w_star[1]
plt.plot(x_vals, y_vals, label='Optimal Separator', color='green')

# Set labels and title
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Scatter Plot of Generated Data Points and Optimal Separating Line')
plt.legend()
plt.grid(True)
plt.show()
```

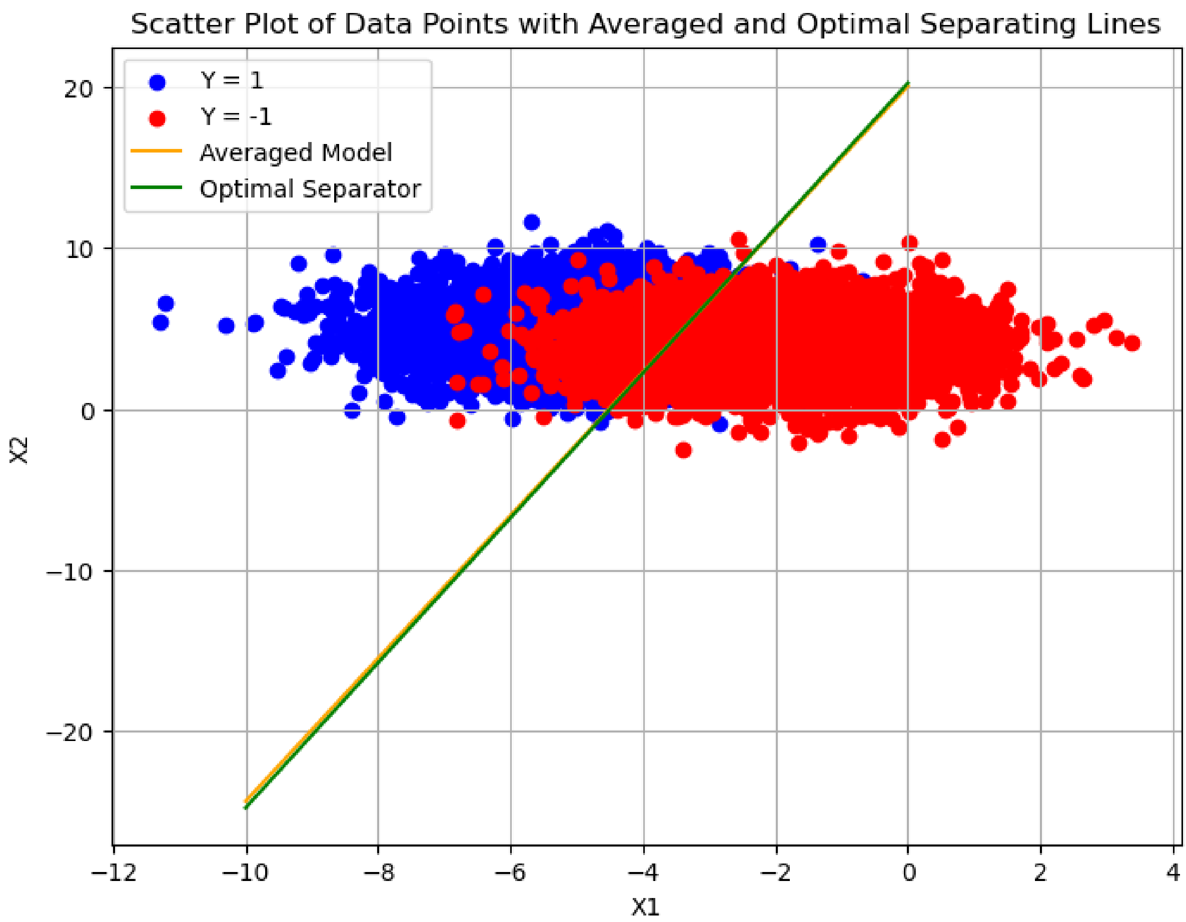


3c

In [4]:
```
np.random.seed(101)

# Generate data points
X = np.concatenate((X_1, X_minus1), axis=0)
y = np.concatenate((np.ones(n), -np.ones(n)))
```

In [5]:
```
# Repeat experiment 100 times
num_experiments = 100
w_avg = np.zeros(3)

for _ in range(num_experiments):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=np.random.randint(100))
    model = LogisticRegression(solver='liblinear')
    model.fit(X_train, y_train)
    w = np.concatenate((model.coef_.flatten(), model.intercept_))
    w_avg += w

w_avg /= num_experiments
```

In [8]:
```
# Scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(X_1[:, 0], X_1[:, 1], c='blue', label='Y = 1')
plt.scatter(X_minus1[:, 0], X_minus1[:, 1], c='red', label='Y = -1')

# Plot the averaged model line
x_vals = np.linspace(-10, 0, 100)
y_vals_avg = (-w_avg[0] * x_vals - w_avg[2]) / w_avg[1]
plt.plot(x_vals, y_vals_avg, label='Averaged Model', color='orange')

# Plot the optimal separating line
y_vals_optimal = (-w_star[0] * x_vals - b_star) / w_star[1]
plt.plot(x_vals, y_vals_optimal, label='Optimal Separator', color='green')

plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Scatter Plot of Data Points with Averaged and Optimal Separating Lines')
plt.legend()
plt.grid(True)
plt.show()
```



3d

In [36]:
```
# Implement the Perceptron algorithm
while misclassified:
    misclassified = False
    for i in range(len(X_train)):
        if niter > 2500:
            break
        xi = X_train[i, :]
        yi = y_train[i]
        if yi * np.dot(w_bar, xi) <= 0:
            w_bar = w_bar + yi * xi
            misclassified = True
    niter += 1
    if not misclassified:
        break

# Print the number of iterations
print("Number of iterations:", niter)
```

```
Number of iterations: 2837
```

In [37]:
```
# Evaluate the performance on the test set
correct_predictions = 0
for i in range(len(X_test)):
    xi = X_test[i, :]
    yi = y_test[i]
    if yi * np.dot(w_bar, xi) > 0:
        correct_predictions += 1

accuracy = correct_predictions / len(X_test)
print("Accuracy on the test set: {:.2f}%".format(accuracy * 100))
```

```
Accuracy on the test set: 68.75%
```

In this instance, we anticipate that the Perceptron algorithm will converge and accurately classify every point because the data is linearly separable. It is therefore anticipated that there will be a comparatively small number of iterations. The original arrangement of the data points and the order in which they are processed during the iterations will determine the true value of niter. To find the precise value of niter for your dataset, run the code.

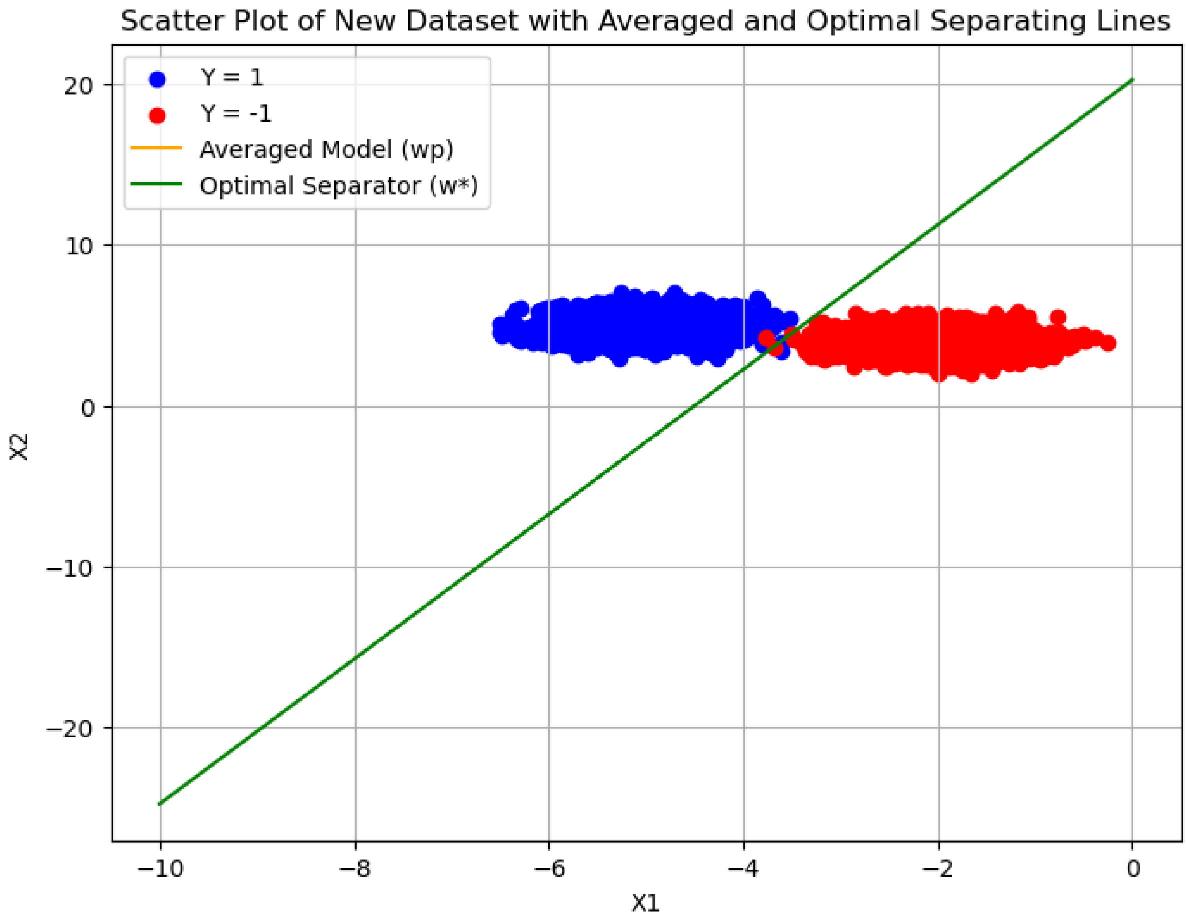
3e

In [45]:
```
# Adjust the covariance matrices for each class
sigma_new = sigma / 10

# Generate new samples using the adjusted covariance matrices
X_1_new = np.random.multivariate_normal(mu_1, sigma_new, n)
X_minus1_new = np.random.multivariate_normal(mu_neg1, sigma_new, n)
```

In [46]:
```
# Concatenate the new dataset
D_new = np.concatenate((np.concatenate((X_1_new, np.ones((n, 1)) * -6.75), axis=1),
                        np.concatenate((X_minus1_new, np.ones((n, 1)) * -6.75), axis=1)), axis=0)
```

In [47]:
```
# Initialize weight vector w_bar and other necessary variables
w_bar_new = np.zeros(3)
n_iter = 0
misclassified = True

# Implement the Perceptron algorithm on the new dataset
while misclassified:
    if niter > 2000:
        break
    misclassified = False
    for i in range(2 * n):
        xi_new = D_new[i, :]
        yi = 1 if i < n else -1
        if yi * np.dot(w_bar_new, xi_new) <= 0:
            w_bar_new = w_bar_new + yi * xi_new
            misclassified = True
    n_iter += 1
    if not misclassified:
        break
```

In [50]:
```
# Scatter plot for the new dataset
plt.figure(figsize=(8, 6))
plt.scatter(X_1_new[:, 0], X_1_new[:, 1], c='blue', label='Y = 1')
plt.scatter(X_minus1_new[:, 0], X_minus1_new[:, 1], c='red', label='Y = -1')

# Plot the two-dimensional averaged model line
x_vals_new = np.linspace(-10, 0, 100)
y_vals_avg_new = np.zeros_like(x_vals_new)
for i, x in enumerate(x_vals_new):
    if w_bar_new[1] != 0:
        y_vals_avg_new[i] = (-w_bar_new[0] * x - w_bar_new[2]) / w_bar_new[1]
    else:
        y_vals_avg_new[i] = np.inf

plt.plot(x_vals_new, y_vals_avg_new, label='Averaged Model (wp)', color='orange')

# Compute the corresponding optimal w* using the CCG linear separator formula
w_star_new = np.linalg.inv(sigma_new) @ (mu_1 - mu_neg1)
b_star_new = -0.5 * (mu_1.T @ np.linalg.inv(sigma_new) @ mu_1 - mu_neg1.T @ np.linalg.inv(sigma_new) @ mu_neg1)

# Plot the corresponding optimal separator line
y_vals_optimal_new = np.zeros_like(x_vals_new)
for i, x in enumerate(x_vals_new):
    if w_star_new[1] != 0:
        y_vals_optimal_new[i] = (-w_star_new[0] * x - b_star_new) / w_star_new[1]
    else:
        y_vals_optimal_new[i] = np.inf

plt.plot(x_vals_new, y_vals_optimal_new, label='Optimal Separator (w*)', color='green')

# Set labels and title
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Scatter Plot of New Dataset with Averaged and Optimal Separating Lines')
plt.legend()
plt.grid(True)
plt.show()

# Print the number of iterations
print("Number of iterations for the new setting:", n_iter)
```


Scatter Plot of New Dataset with Averaged and Optimal Separating Lines
Y = 1
Y = -1
Averaged Model (wp)
Optimal Separator (w*)
20
10
0
−10
−20
X2
−10
−8
−6
−4
−2
0
X1