# Logistic Regression

**1. Fit a logistic regression classifier to training data set. What is the accuracy on the test set? Explain why in standard logistic regression, without any type of regularization, the weights may not converge (even though the predicted label for each data point effectively does) if the input data is linearly separable.**

When the data is linearly separable, maximum likelihood function approaches infinity and hence it won't converge even though predicted labels are correct.

In [1]:
```
import pandas as pd
import numpy as np
import math
```

In [2]:
```
sonar_train = pd.read_csv('sonar_train.data', header=None)
sonar_test = pd.read_csv('sonar_test.data', header=None)
sonar_valid = pd.read_csv('sonar_valid.data', header=None)

sonar_train.loc[sonar_train[60] == 2, 60] = -1
sonar_test.loc[sonar_test[60] == 2, 60] = -1
sonar_valid.loc[sonar_valid[60] == 2, 60] = -1

def split_data(data):
    return data.iloc[:, :60].to_numpy(), data.iloc[:, 60:].to_numpy()

X_train, y_train = split_data(sonar_train)
X_validation, y_validation = split_data(sonar_valid)
X_test, y_test = split_data(sonar_test)
```

In [142]:
```
def sigmoid_function(x):
    return np.where(x >= 0,
                    1 / (1 + np.exp(-x)),
                    np.exp(x) / (1 + np.exp(x)))
```

In [143]:

```
def compute_loss(X, Y, w, b):
    m, n = X.shape
    loss = 0
    for i in range(m):
        x = X[i, :]
        y = Y[i, 0]
        z = np.dot(w.T, x) + b
        loss += ((y+1)/2)*z- np.log(1 + np.exp(z))
    return loss

def logistic_regression(X, Y, step_size):
    m, n = X.shape
    w = np.array([1/m] * n)
    b = 0
    ploss = compute_loss(X, Y, w, b)
    itrs = 0
    while True:
        itrs += 1
        gradient_w = 0
        gradient_b = 0
        for i in range(m):
            x = X[i, :]
            z = np.dot(w.T, x) + b
            a = sigmoid_function(z)
            gradient_w += x * ((Y[i, 0]+1)/2 - a)
            gradient_b += ((Y[i, 0]+1)/2 - a)
        w = w + step_size * gradient_w
        b = b + step_size * gradient_b
        loss = compute_loss(X, Y, w, b)
        if loss - ploss < 0.0005 and itrs > 1:
            break

        if itrs % 1000 == 0:
            print(itrs, ploss, loss, loss-ploss)
        ploss = loss
        weight = w
        bias = b
    print("Iterations", itrs)
    return w, b
```

In [144]:

```
def compute_accuracy(X, Y, w, b):
    m, n = X.shape
    cnt = 0
    for i in range(m):
        x = X[i, :]
        t = np.dot(w.T, x) + b
        if t > 0:
            if Y[i, 0] > 0:
                cnt += 1
        else:
            if Y[i, 0] < 0:
                cnt += 1
    return cnt/m * 100
```

In [145]:
```
w, b = logistic_regression(X_train, y_train, 0.001)
valid_acc = compute_accuracy(X_validation, y_validation, w, b)
print("Validation accuracy is",valid_acc)
```

```
1000 -47.534424157403976 -47.52804979097219 0.006374366431785461
2000 -43.4970861439145 -43.494404863840025 0.002681280074476433
3000 -41.39334828377261 -41.39166718627862 0.0016810974939929224
4000 -39.96185281222269 -39.960622811791396 0.0012300004312919555
5000 -38.869420544631986 -38.86844487377355 0.0009756708584376383
6000 -37.98000201518943 -37.97918804569227 0.0008139694971589506
7000 -37.22464485160145 -37.22394184042838 0.0007030111730657О4
8000 -36.56379684045631 -36.563174241515256 0.0006225989410566513
9000 -35.97288293743816 -35.972321128199816 0.0005618092383414819
10000 -35.43573858848996 -35.43522433480008 0.0005142536898787853
Iterations 10349
Validation accuracy is 80.76923076923077
```

In [7]:
```
test_acc = compute_accuracy(X_test, y_test, w, b)
print("Test accuracy",test_acc)
```

```
Test accuracy 82.6923076923077
```

2. Fit a logistic regression classifier with an l2 penalty on the weights to this data set using the validation set to select a good choice of the regularization constant. Report your selected constant, the learned weights and bias, and the accuracy on the test set.

In [8]:
```
def logistic_regression_l2(X, Y, step_size, l2):
    m, n = X.shape
    w = np.array([1/m] * n)
    b = 0
    ploss = compute_loss(X, Y, w, b)
    itrs = 0
    while True:
        itrs += 1
        gradient_w = 0
        gradient_b = 0
        for i in range(m):
            x = X[i, :]
            z = np.dot(w.T, x) + b
            a = sigmoid_function(z)
            gradient_w += x * ((Y[i, 0]+1)/2 - a)
            gradient_b += ((Y[i, 0]+1)/2 - a)
        gradient_w = gradient_w - (l2 * w)
        w = w + step_size * gradient_w
        b = b + step_size * gradient_b
        loss = compute_loss(X, Y, w, b) - l2 * (np.linalg.norm(w) ** 2)
        if loss - ploss < 0.0005 and itrs > 1:
            break
        ploss = loss
    print(f"L2 {l2} - {itrs} iterations")
    return w, b
```

In [9]:
```
l2_cons = [1e-4, 1e-3, 1e-2, 1e-1, 1, 1e1, 1e3]
best_l2_val_acc = 0
best_w2 = 0
best_l2 = 0
best_b2 = 0
for l2 in l2_cons:
    w2, b2 = logistic_regression_l2(X_train, y_train, 0.001, l2)
    l2_valid_acc = compute_accuracy(X_validation, y_validation, w2, b2)
    print(f"{l2} - Validation Accuracy {l2_valid_acc}")
    if l2_valid_acc > best_l2_val_acc:
        best_l2_val_acc = l2_valid_acc
        best_w2 = w2
        best_l2 = l2
        best_b2 = b2
```

```
L2 0.0001 - 10301 iterations
0.0001 - Validation Accuracy 80.76923076923077
L2 0.001 - 9894 iterations
0.001 - Validation Accuracy 80.76923076923077
L2 0.01 - 7382 iterations
0.01 - Validation Accuracy 78.84615384615384
L2 0.1 - 2681 iterations
0.1 - Validation Accuracy 75.0
L2 1 - 480 iterations
1 - Validation Accuracy 78.84615384615384
L2 10.0 - 64 iterations
10.0 - Validation Accuracy 75.0
L2 1000.0 - 3 iterations
1000.0 - Validation Accuracy 63.46153846153846
```

In [10]:
```
print("Best L2 Constant", best_l2)
print("Best Validation Accuracy", best_l2_val_acc)
print("Weights", best_w2)
print("Bias", best_b2)
```

```
Best L2 Constant 0.0001
Best Validation Accuracy 80.76923076923077
Weights [-1.38156587 -1.16521762 -0.52008009 -1.61313899  0.03852722  1.0396578
7
  2.4291557   2.21942791 -3.16563393 -1.86213178 -3.26060765 -3.05090886
 -1.97745255 -0.25376856 -0.21681329  2.00693198  2.62121749 -0.02500562
 -0.98419332 -1.53627997  0.092207   -0.48988661 -1.79897843 -1.25966664
  0.95369775  3.19533978  0.21978754 -1.3083397   0.34256399 -0.53823538
  2.5767834  -2.7451562   0.04695152  2.43472489  0.44680527  1.9555875
  0.74679886 -1.07261112  0.19224791  2.91063996  0.4655868   0.36342821
 -3.67342713 -3.11643217 -1.94723882 -0.94111064 -2.17006824 -3.15729646
 -1.83960432  0.08501944 -0.53618483 -0.51618584 -0.22545448 -0.2740607
  0.17352099 -0.2270359   0.03304952 -0.30383766 -0.29071737 -0.20942555]
Bias 1.80159729345166
```

In [11]:
```
test_l2_acc = compute_accuracy(X_test, y_test, best_w2, best_b2)
print("Test accuracy",test_l2_acc)
```

```
Test accuracy 82.6923076923077
```

3. Fit a logistic regression classifier with an l1 penalty on the weights to this data set using the validation set to select a good choice of the regularization constant. Report your selected constant, the learned weights and bias, and the accuracy on the test set.

In [116]:
```
def logistic_regression_l1(X, Y, step_size, l1):
    m, n = X.shape
    w = np.array([0] * n)
    b = 0
    ploss = compute_loss(X, Y, w, b)
    itrs = 0
    while True:
        itrs += 1
        gradient_w = 0
        gradient_b = 0
        for i in range(m):
            x = X[i, :]
            z = np.dot(w.T, x) + b
            a = sigmoid_function(z)
            gradient_w += x * ((Y[i, 0]+1)/2 - a)
            gradient_b += ((Y[i, 0]+1)/2 - a)
        gradient_w = gradient_w - (l1 * np.sign(w))
        w = w + step_size * gradient_w
        b = b + step_size * gradient_b
        loss = compute_loss(X, Y, w, b) - (l1 * np.linalg.norm(w))
        if loss - ploss < 0.0005 and itrs > 1:
            break
        ploss = loss
    print(f"L1 {l1} - {itrs} iterations")
    return w, b
```

In [138]:
```
best_l1_val_acc = 0
best_w1 = 0
best_l1 = 0
best_b1 = 0
for l1 in l2_cons:
    w1, b1 = logistic_regression_l1(X_train, y_train, 0.001, l1)
    l1_valid_acc = compute_accuracy(X_validation, y_validation, w1, b1)
    print(f"{l1} - Validation Accuracy {l1_valid_acc}")
    if l1_valid_acc > best_l1_val_acc:
        best_l1_val_acc = l1_valid_acc
        best_w1 = w1
        best_l1 = l1
        best_b1 = b1
```

```
L1 0.0001 - 10343 iterations
0.0001 - Validation Accuracy 80.76923076923077
L1 0.001 - 10299 iterations
0.001 - Validation Accuracy 80.76923076923077
L1 0.01 - 9882 iterations
0.01 - Validation Accuracy 80.76923076923077
L1 0.1 - 6869 iterations
0.1 - Validation Accuracy 80.76923076923077
L1 1 - 707 iterations
1 - Validation Accuracy 76.92307692307693
L1 10.0 - 2 iterations
10.0 - Validation Accuracy 36.53846153846153
L1 1000.0 - 2 iterations
1000.0 - Validation Accuracy 36.53846153846153
```

In [139]:
```
print("Best L1 Constant", best_l1)
print("Best Validation Accuracy", best_l1_val_acc)
print("Weights", best_w1)
print("Bias", best_b1)
```

```
Best L1 Constant 0.0001
Best Validation Accuracy 80.76923076923077
Weights [-1.39388289 -1.17492763 -0.52561677 -1.62238042  0.03777807  1.0411118
6
  2.4357166   2.22920315 -3.17415389 -1.86118016 -3.26253394 -3.05396299
 -1.97727662 -0.25134166 -0.21926155  2.0088797   2.62710308 -0.03005202
 -0.98453536 -1.53898234  0.09283601 -0.49045029 -1.80151384 -1.26154444
  0.95294217  3.19990681  0.21819296 -1.30931496  0.34147677 -0.54206743
  2.58607567 -2.75265136  0.0452214   2.43945234  0.44350477  1.9595406
  0.74635965 -1.07488427  0.19202989  2.91708211  0.46400849  0.36742084
 -3.68112414 -3.11952767 -1.94507024 -0.9394914  -2.17828965 -3.17145992
 -1.85118572  0.07689944 -0.54558893 -0.52600336 -0.23453652 -0.28327269
  0.16391332 -0.23624945  0.022867   -0.31325695 -0.30028642 -0.2188688 ]
Bias 1.8103762218194575
```

In [140]:
```
test_l1_acc = compute_accuracy(X_test, y_test, best_w1, best_b1)
print("Test accuracy",test_l1_acc)
```

```
Test accuracy 82.6923076923077
```

**4. Does l1 or l2 tend to produce sparser weight vectors?**

l1 regularizations tend to produce sparser weight vectors than l2 because L1 uses abs() function which is not differentiable at all points and this forces weights towards zero.