HW 2

Alonso Buitano

Problem 1.

10 coins : 3 - 100% Heads , 3 - 100% Tails , 4 - 50/50 H/T

a) ~~3(1) + 3(0) + 4(0.5) / 10~~ $\frac{3}{10} \to$ 100% of H $\quad \frac{3}{10} \to 0\%$ $\quad \frac{4}{10} \to$ 50% of H $\Rightarrow \frac{4(0.5)}{10} = \frac{2}{10}$

$\Rightarrow \frac{3+2}{10} = 0.5 \Rightarrow$ 50% chance of getting a head.

b) $P(\text{Tails coin} \mid \text{Tails}) = \frac{P(\text{Tails} \mid \text{Tails coin}) \cdot P(\text{Tails coin})}{P(\text{Tails})}$

$= \frac{1\ (.3)}{0.5} = \frac{0.3}{0.5} = 0.6 \Rightarrow$ 60% of the coin having Tails on both sides

$P(\text{Fair} \mid \text{Tails}) = \frac{P(\text{Tails} \mid \text{Fair})\, P(\text{fair})}{P(\text{Tails})}$

$= \frac{0.5\,(0.4)}{0.5} = 0.4 \Rightarrow$ 40% of the coin being fair

c) First coin: 60% of being a Tails coin, 40% of being a fair coin

$\frac{2}{9} \to$ 100% of T	$\frac{3}{9} \to$ 100% of T
$\frac{3}{9} \to 0\%$	$\frac{3}{9} \to 0\%$
$\frac{4}{9} \to$ 50% of T	$\frac{3}{9} \to$ 50% of T
$\frac{2}{9} + \frac{2}{9} = \frac{4}{9} = 0.44$	$\frac{3}{9} + \frac{3/2}{9} = \frac{4.5}{9} = 50\%$

$(0.44)(0.6) + (.5)(0.4) = 0.464 \Rightarrow$ 46.4% of getting Tail again

Problem 3.

a) $f(x) = \theta \sigma^{\theta} x^{-\theta-1}$ for $x \geq \sigma = P(x_i | \theta, \sigma)$

~~b~~ $L(\theta) = \prod_{i=1}^{n} \theta \sigma^{\theta} x_i^{-\theta-1}$

$\ell(\theta) = \log L(\theta) = \sum_{i=1}^{n} \log(\theta \sigma^{\theta} x_i^{-\theta-1})$

$\ell(\theta) = \sum_{i=1}^{n} (\log\theta + \theta\log\sigma + (-\theta-1)\log x_i)$

$\frac{\partial \ell(\theta)}{\partial \theta} = \sum_{i=1}^{n} \left(\frac{1}{\theta} + \log\sigma - \log x_i\right)$

$\frac{\partial \ell(\theta)}{\partial \sigma} = \sum_{i=1}^{n} \frac{\theta}{\sigma}$

$\hat{\theta} : \frac{\partial \ell(\theta)}{\partial \theta} = 0 \quad \sum_{i=0}^{n}(\frac{1}{\theta} + \log\sigma - \log x_i) = 0 = \frac{n}{\theta} + n\log\sigma - \sum_{i=0}^{n} \log x_i$

$\hat{\theta} = \frac{n}{\sum_{i=1}^{n}(\log x_i - \log\sigma)}$

$\hat{\sigma} : \frac{\partial \ell(\theta)}{\partial \sigma} = 0$

$\hat{\sigma} : \sum_{i=1}^{n} \frac{\theta}{\sigma} = 0$

$\hat{\sigma} = \sum_{i=1}^{n} \theta = n\theta$

b) $x_i = 13, 2, 16, 5, 11, 16, 18, 5, 8, 15$

$p(\ln(x_i)) = N(\mu, \sigma^2)$

$\Rightarrow$

$$\hat{\mu} = \sum_{i=1}^{n} \frac{\ln(x_i)}{n}$$

$\lambda = \mu$

$\xi = \sigma$

$$\underbrace{\hat{\sigma}^2 = \sum_{i=1}^{n} \frac{(\ln(x_i) - \mu)^2}{n}}_{\text{from class notes}}$$

$\lambda = 2.209$

$\xi = 0.00079$

c) $f(x;\theta) = \frac{x}{\theta^2} e^{-\frac{x}{\theta}}$ for $x > 0$

$$L(\theta) = \prod_{i=1}^{n} \frac{x_i}{\theta^2} e^{-\frac{x_i}{\theta}}$$

$$\ell(\theta) = \log L(\theta) = \sum_{i=1}^{n} \left(\log x_i - \log \theta^2 - \frac{x_i}{\theta} \right)$$

$$\hat{\Theta} \Rightarrow \frac{\partial \ell(\theta)}{\partial \theta} = 0$$

$$\frac{\partial \ell(\theta)}{\partial \theta} = \sum_{i=1}^{n} -\frac{1}{\theta^2} + \frac{x_i}{\theta^2} = \sum_{i=1}^{n} \frac{1}{\theta}(-1 + x_i) = 0$$

$$\hat{\Theta} = \sqrt{\sum_{i=1}^{n} x_i - 1}$$

In [1]:
```python
import numpy as np
import time

np.random.seed(24787)
a = np.random.randint(8, size=(3,4,4))
```

In [2]:
```python
print(a)
print('Shape of a: ',a.shape)
```

```
[[[2 6 4 1]
  [0 4 4 3]
  [6 6 1 2]
  [7 0 6 5]]

 [[1 3 3 7]
  [4 7 2 5]
  [0 4 6 7]
  [5 5 7 1]]

 [[7 2 4 5]
  [6 7 7 0]
  [6 2 0 4]
  [2 0 7 6]]]
Shape of a:  (3, 4, 4)
```

In [3]:
```python
fours = np.where(a==4)
#Print indices, first array is depth, second is row, third is column.
print(fours)
```

```
(array([0, 0, 0, 1, 1, 2, 2]), array([0, 1, 1, 1, 2, 0, 2]), array([2, 1,
2, 0, 1, 2, 3]))
```

In [4]:
```python
b = np.tile(a,(2,2))
b.shape
```

Out[4]:
```
(3, 8, 8)
```

In [5]:
```python
c = b.sum(axis=0)

print(c)
print('Shape of c: ',c.shape)
```

```
[[10 11 11 13 10 11 11 13]
 [10 18 13  8 10 18 13  8]
 [12 12  7 13 12 12  7 13]
 [14  5 20 12 14  5 20 12]
 [10 11 11 13 10 11 11 13]
 [10 18 13  8 10 18 13  8]
 [12 12  7 13 12 12  7 13]
 [14  5 20 12 14  5 20 12]]
Shape of c:  (8, 8)
```

In [6]:
```python
np.random.seed(24787)
a = np.random.randint(8, size=(1000,1000))
b = np.random.randint(8, size=(1000,1000))
```

In [7]:
```
def matmul(a,b):
    product = np.zeros(a.shape)
    for x in range(a.shape[0]):
        for i in range(a.shape[0]):
            for j in range(a.shape[1]):
                product[i,x] += a[i,j]*b[j,x]

    return product
```

In [8]:
```
start = time.time()
c = matmul(a,b)
print("Multiplication took: ", time.time()-start, " seconds.")
```

```
Multiplication took:  515.9055788516998  seconds.
```

In [9]:
```
start = time.time()
d = a@b
print("Multiplication took: ", time.time()-start, " seconds.")
```

```
Multiplication took:  0.7417197227478027  seconds.
```

In [10]:
```
c==d
```

Out[10]:
```
array([[ True,  True,  True, ...,  True,  True,  True],
       [ True,  True,  True, ...,  True,  True,  True],
       [ True,  True,  True, ...,  True,  True,  True],
       ...,
       [ True,  True,  True, ...,  True,  True,  True],
       [ True,  True,  True, ...,  True,  True,  True],
       [ True,  True,  True, ...,  True,  True,  True]])
```

Numpy is generally faster than python because it runs its functions with C and Fortran (static languages), because dynamic languages (like Python) are slow for loops anf callinf functions.

Question 4: Logistic Regression

In [1]:
```
#Import all the required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Load the data

In [2]:
```
# load the data
X1 = pd.read_csv('class0-input.csv')
X2 = pd.read_csv('class1-input.csv')

labels = pd.read_csv('labels.csv')
# Perform important operations on the data
X = pd.concat([X1,X2]).to_numpy()
Y = labels.label.to_numpy()
```

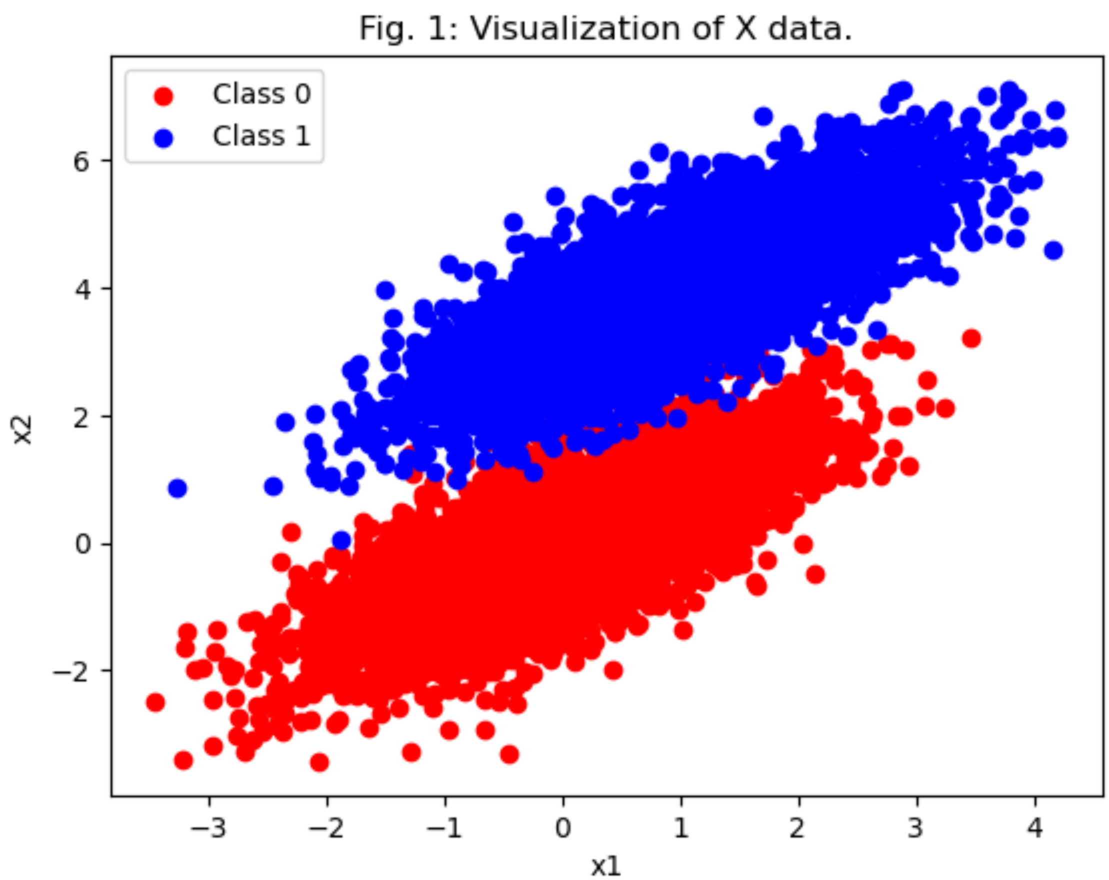
Check the shape

In [3]:
```
# Shape of X
print(X.shape)
# Shape of Y
print(Y.shape)
```

```
(10000, 2)
(10000,)
```

Visualize the data

In [4]:
```
# Use different colors for each class
# Use plt.scatter
# Dont forget to add axes titles, graph title, legend
plt.scatter(X[Y == 0][:,0], X[Y == 0][:,1], c = 'red')
plt.scatter(X[Y == 1][:,0], X[Y == 1][:,1], c = 'blue')
# plt.scatter(X[5000:,0], X[5000:,1], c = 'blue')
plt.legend(['Class 0','Class 1'])
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Fig. 1: Visualization of X data.')
plt.show()
```



Define the required functions

In [5]:
```
# Pass in the required arguments
# Implement the sigmoid function
def sigmoid(weights, X):
    e = weights[0]
    for i in range(1,len(weights)):
        e += weights[i]*X[:,i-1]
    e = np.exp(e)
    sig = e / (1 + e)
    return sig
```

In [6]:
```
# Pass in the required arguments
# The function should return the gradients
def calculate_gradients(pred, X, Y):
    grads = np.array(np.mean(Y - pred))
    for i in range(X.shape[1]):
        grads = np.append(grads,np.mean((Y - pred)*X[:,i]))
    return grads
```

In [7]:
```python
# Update the weights using gradients calculated using above function and lea
# The function should return the updated weights to be used in the next step
def update_weights(prev_weights, current_grads, learning_rate):

    for i in range(len(prev_weights)):
        prev_weights[i] = prev_weights[i] + learning_rate*current_grads[i]
    return prev_weights
```

In [8]:
```python
# Use the implemented functions in the main function
# 'main' function should return weights after all the iterations
# Dont forget to divide by the number of datapoints wherever necessary!
# Initialize the intial weights randomly

def main(X, Y, learning_rate = 0.00005, num_steps = 50000):
    np.random.seed(24787)
    weights = np.random.rand(3)
    x1 = X
    for i in range(num_steps):
        sig = sigmoid(weights, X)
        grads = calculate_gradients(sig, X, Y)
        weights = update_weights(weights, grads, learning_rate)
    sig = sigmoid(weights, X)
    grads = calculate_gradients(sig, X, Y)
    weights = update_weights(weights, grads, learning_rate)
    return weights
```

In [9]:
```python
# Pass in the required arguments (final weights and input)
# The function should return the predictions obtained using sigmoid function
def predict(weights, X):
    Y_hat = sigmoid(weights, X)
    return Y_hat
```

In [10]:
```python
# Use the final weights to perform prediction using predict function
# Convert the predictions to '0' or '1'
# Calculate the accuracy using predictions and labels

B = main(X,Y)
```

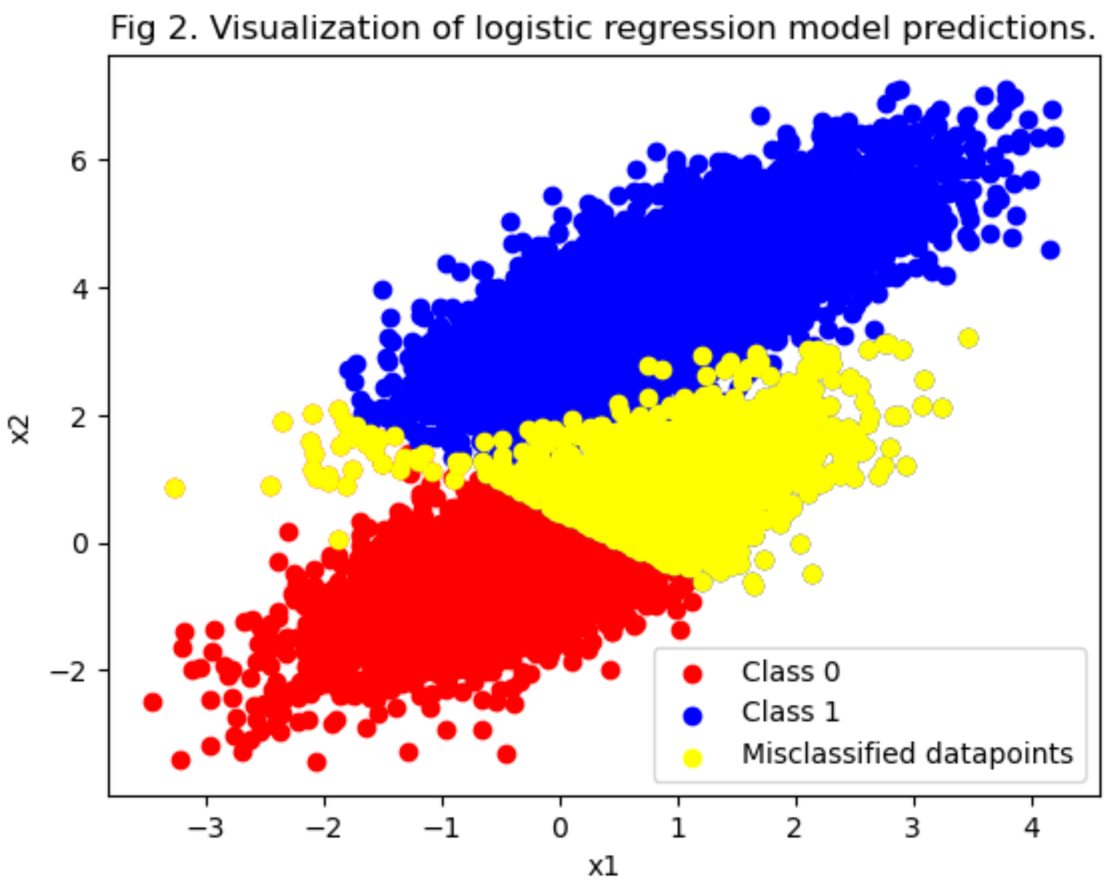
In [11]:
```python
final_prediction = predict(B,X)

for i in range(len(final_prediction)):
    if final_prediction[i] >= 0.5:
        final_prediction[i] = 1
    else : final_prediction[i] = 0
```

In [12]:
```python
accuracy = (final_prediction == Y).mean()
print("Accuracy of logistic regression model is: ", accuracy)
```

```
Accuracy of logistic regression model is:  0.7966
```

Visualize the misclassification

In [13]:
```
# Use different colors for class 0, class 1 and misclassified datapoints
# Use plt.scatter
# Dont forget to add axes titles, graph title, legend
plt.scatter(X[final_prediction == 0][:,0], X[final_prediction == 0][:,1], c
plt.scatter(X[final_prediction == 1][:,0], X[final_prediction == 1][:,1], c
plt.scatter(X[(final_prediction - Y)  != 0][:,0], X[(final_prediction - Y) !
plt.legend(['Class 0', 'Class 1', 'Misclassified datapoints'])
plt.xlabel('x1')
plt.ylabel('x2')
plt.title("Fig 2. Visualization of logistic regression model predictions.")
plt.show()
```



Compare the results with sklearn's Logistic Regression

In [14]:
```
# import sklearn and necessary libraries
from sklearn.linear_model import LogisticRegression
```

In [15]:
```
# Print the accuracy obtained by sklearn and your model
skmodel = LogisticRegression().fit(X,Y)
```

In [16]:
```
skmodel.score(X,Y)
```

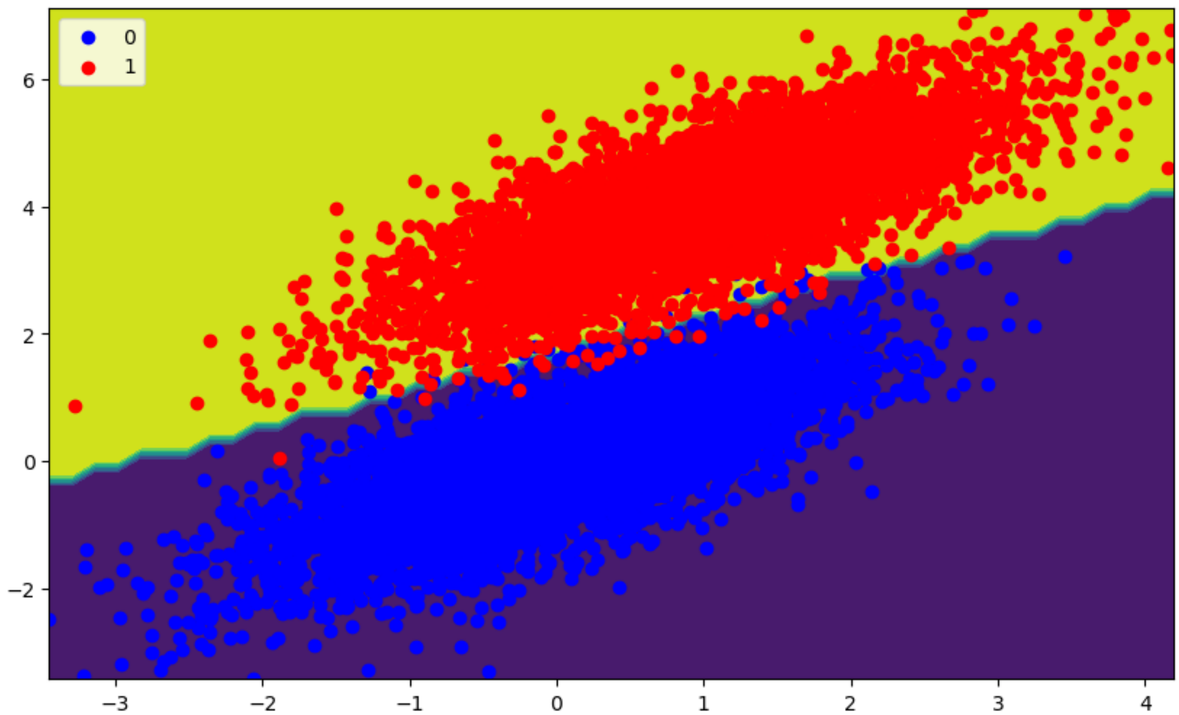
Out[16]: 0.9948

In [17]:

```
plt.figure(figsize=(10, 6))
plt.scatter(X[Y == 0][:, 0], X[Y == 0][:, 1], color='b', label='0')
plt.scatter(X[Y == 1][:, 0], X[Y == 1][:, 1], color='r', label='1')
plt.legend()
x1_min, x1_max = X[:,0].min(), X[:,0].max(),
x2_min, x2_max = X[:,1].min(), X[:,1].max(),
xx1, xx2 = np.meshgrid(np.linspace(x1_min, x1_max), np.linspace(x2_min, x2_m
grid = np.c_[xx1.ravel(), xx2.ravel()]
probs = skmodel.predict(grid).reshape(xx1.shape)
plt.contourf(xx1, xx2, probs);
plt.scatter(X[Y == 0][:, 0], X[Y == 0][:, 1], color='b', label='0')
plt.scatter(X[Y == 1][:, 0], X[Y == 1][:, 1], color='r', label='1')
```

Out[17]:

```
<matplotlib.collections.PathCollection at 0x1683e9b10>
```



With the given hyperparameters, the sklearn model has much better accuracy than my logistic regression model, at 20% higher accuracy score.

Worth noting that, changing the learning rate of my model (removing a 0), its accuracy becomes much closer to the one given by sklearn.