In [1]:

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

In [ ]:

```
!unzip /content/gdrive/Shareddrives/dogs_cats/dogs-vs-cats.zip -d "/content/"
```

In [3]:

```
import os
import cv2
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import glob
```

In [4]:

```
name_list = glob.glob('/content/dogs-vs-cats/train/'+"cat*")
print(len(name_list))

X = []
name_label = []

for name in name_list:
  name_label.append(0)

  img = cv2.imread(name)
  img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  img = cv2.resize(img, (64,64))

  X.append((img))

len(X)
```

```
12500
```

Out[4]:

```
12500
```

In [5]:

```
name_list = glob.glob('/content/dogs-vs-cats/train/'+"dog*")
print(len(name_list))

for name in name_list:
  name_label.append(1)

  img = cv2.imread(name)
  img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  img = cv2.resize(img, (64,64))

  X.append((img))

len(X)
```

```
12500
```

Out[5]:

```
25000
```

In [6]:

```
X = np.array(X)
```

```
y = np.array(name_label)

print(X.shape)
print(y.shape)
```

```
(25000, 64, 64)
(25000,)
```

In [7]:

```
# Split the dataset into train set and test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=
42)
print(X_train.shape, X_test.shape)
print(y_train.shape, y_test.shape)
```

```
(20000, 64, 64) (5000, 64, 64)
(20000,) (5000,)
```

In [8]:

```
X_train_scaled = np.array([x.ravel()/255. for x in X_train])
X_test_scaled = np.array([x.ravel()/255. for x in X_test])

print(X_train_scaled.shape)
print(X_test_scaled.shape)
```

```
(20000, 4096)
(5000, 4096)
```

In [9]:

```
y_train_scaled = np.array([y for y in y_train])
y_test_scaled = np.array([y for y in y_test])
```

In [10]:

```
# sigmoid function
g = lambda z : np.exp(z) / (1+np.exp(z))

def predict_prob(X,w):
  z = np.dot(X,w)
  a = g(z)
  return a
def predict(X,w):
  y_hat = predict_prob(X,w)
  y_hat[y_hat >=0.5] =1
  y_hat[y_hat <0.5] =0
  return y_hat
# loss function
def loss(X,y,w):
  y_hat = predict_prob(X,w)
  l = y*np.log(y_hat) + (1-y)*np.log(1-y_hat)
  return -np.mean(l)
```

In [11]:

```
# gradient
def grad(X,y,w):
  y_hat = predict_prob(X,w)
  delta = y_hat - y
  dw = np.dot(X.T,delta)
  return dw
```

In [12]:

```
def gradient_descent(X,y,lr = 1e-7, epochs = 2000):
  w = np.zeros((X.shape[1],1))
  losses = []
  for i in range(epochs):
```

```
    dw = grad(X,y,w)
    w -= lr*dw
    a = loss(X,y,w)
    losses.append(a)
  return losses,w
```

In [13]:

```
print(X_train_scaled)
print(np.array([y_train_scaled]).T)
print(X_train_scaled.shape)
print(np.array([y_train_scaled]).T.shape)
```

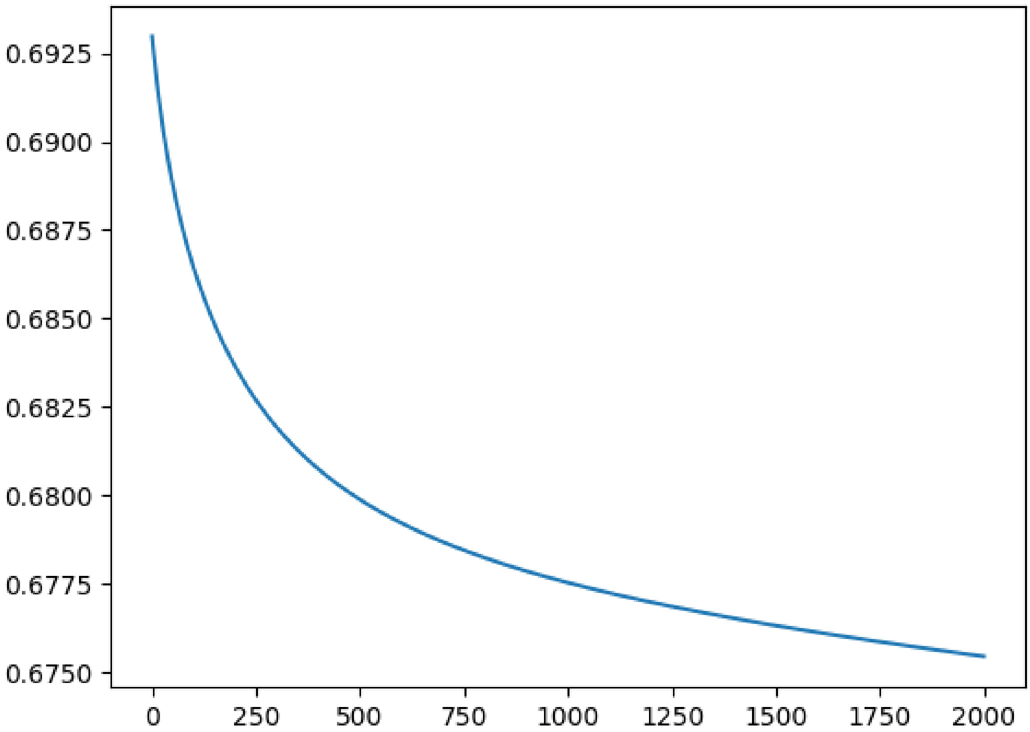
```
[[0.2627451  0.31372549 0.31372549 ... 0.49803922 0.44313725 0.42352941]
 [0.11764706 0.1372549  0.09411765 ... 0.26666667 0.25098039 0.22745098]
 [0.41568627 0.34509804 0.49803922 ... 0.75294118 0.73333333 0.7372549 ]
 ...
 [0.03921569 0.04313725 0.19215686 ... 0.56862745 0.52941176 0.49411765]
 [0.67843137 0.69019608 0.69411765 ... 0.48627451 0.4745098  0.4745098 ]
 [0.34509804 0.33333333 0.3254902  ... 0.11764706 0.38431373 0.45098039]]
[[1]
 [1]
 [0]
 ...
 [0]
 [1]
 [1]]
(20000, 4096)
(20000, 1)
```

In [14]:

```
loss,w = gradient_descent(X_train_scaled,np.array([y_train_scaled]).T)
plt.plot(loss)
```

Out[14]:

```
[<matplotlib.lines.Line2D at 0x7fa95f195ee0>]
```



In [15]:

```
import numpy as np

class LogisticRegression1:
    def __init__(self, learning_rate=0.001, num_iterations=2000, fit_intercept=True):
        self.learning_rate = learning_rate
```

```
        self.num_iterations = num_iterations
        self.fit_intercept = fit_intercept

    def __add_intercept(self, X):
        intercept = np.ones((X.shape[0], 1))
        return np.concatenate((intercept, X), axis=1)

    def __sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def __loss(self, h, y):
        return (-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()

    def fit(self, X, y):
        if self.fit_intercept:
            X = self.__add_intercept(X)

        # Khởi tạo các tham số
        self.theta = np.zeros(X.shape[1])

        for i in range(self.num_iterations):
            z = np.dot(X, self.theta)
            h = self.__sigmoid(z)
            gradient = np.dot(X.T, (h - y)) / y.size
            self.theta -= self.learning_rate * gradient

            if(i % 10000 == 0):
                z = np.dot(X, self.theta)
                h = self.__sigmoid(z)
                print(f'loss: {self.__loss(h, y)} \t')

    def predict_prob(self, X):
        if self.fit_intercept:
            X = self.__add_intercept(X)

        return self.__sigmoid(np.dot(X, self.theta))

    def predict(self, X, threshold = 0.5):
        y_pred = self.predict_prob(X)
        y_pred[y_pred >=threshold] =1
        y_pred[y_pred <threshold] =0
        return y_pred
```

In [16]:

```
# model = LogisticRegression1(fit_intercept=False)
# model.fit(X_train_scaled, y_train)
y_hat = predict(X_test_scaled,w)
```

Đánh giá mô hình: accuracy, recall, f1_score

In [17]:

```
def evaluation(y_true, y_pred):
    # y_true và y_pred là hai danh sách chứa nhãn thực tế và nhãn dự đoán của các mẫu
    # Giả sử nhãn Positive là 1 và nhãn Negative là 0
    tp = tn = fp = fn = 0 # Khởi tạo các biến đếm
    for i in range(len(y_true)): # Duyệt qua từng cặp nhãn
        if y_true[i] == 1 and y_pred[i] == 1: # Nếu cả hai nhãn đều là Positive
            tp += 1 # Tăng biến tp lên 1
        elif y_true[i] == 0 and y_pred[i] == 0: # Nếu cả hai nhãn đều là Negative
            tn += 1 # Tăng biến tn lên 1
        elif y_true[i] == 0 and y_pred[i] == 1: # Nếu nhãn thực tế là Negative nhưng nhã
n dự đoán lại là Positive
            fp += 1 # Tăng biến fp lên 1
        elif y_true[i] == 1 and y_pred[i] == 0: # Trường hợp còn lại, tức là nhãn thực t
ế là Positive nhưng nhãn dự đoán lại là Negative
            fn += 1 # Tăng biến fn lên 1

    accuracy = (tp+tn)/(tp+tn+fp+fn)
```

```
    precision = (tp)/(tp+fp)
    recall = (tp)/(tp+fn)
    f1_score = 2*precision*recall/(precision+recall)
    print(f"Accuracy = {accuracy}")
    print(f"Precision = {precision}")
    print(f"Recall = {recall}")
    print(f"F1 Score = {f1_score}")
```

In [18]:

```
evaluation(y_test, y_hat)
```

```
Accuracy = 0.5694
Precision = 0.5639445300462249
Recall = 0.5891348088531188
F1 Score = 0.5762645148592797
```

Khởi tạo hàm batch_generator để chia nhỏ dữ liệu khi train trên từng epochs.

In [19]:

```
import numpy as np

def batch_generator(X, y, batch_size):
    """
    Hàm chia dữ liệu thành các batch để train trên từng epoch

    Arguments:
    X -- ma trận đầu vào (m, n)
    y -- ma trận đầu ra (m, 1)
    batch_size -- kích thước mỗi batch

    Returns:
    batches -- list chứa các batch được chia từ dữ liệu đầu vào
    """

    m = X.shape[0]   # số lượng mẫu dữ liệu
    batches = []  # khởi tạo list chứa các batch

    # chia dữ liệu thành các batch
    for i in range(0, m, batch_size):
        X_batch = X[i:i+batch_size, :]
        y_batch = y[i:i+batch_size]
        batch = (X_batch, y_batch)
        batches.append(batch)

    return batches
```

Khởi tạo 1 class có tên là binary1, sử dụng các hàm ở trên và tích hợp trong class để huấn luyện mô hình. Đồng thời thêm hệ số bias. Đánh giá mô hình: accuracy, recall, f1_score

In [20]:

```
bias = np.ones((X_train.shape[0],1))
X_train_scaled = np.hstack((bias,X_train_scaled));
bias = np.ones((X_test.shape[0],1))
X_test_scaled = np.hstack((bias,X_test_scaled));
```

In [33]:

```
class binary1:
    # def __init__(self, learning_rate=1e-7, n_iters=2000):
    #     self.lr = learning_rate
    #     self.n_iters = n_iters
    #     self.weights = None
```

```
    #     self.bias = None

    # def sigmoid(self, z):
    #     return 1 / (1 + np.exp(-z))

    # def fit(self, X, y):
    #     n_samples, n_features = X.shape

    #     # init parameters
    #     self.weights = np.zeros(n_features)
    #     self.bias = 0

    #     # gradient descent
    #     for _ in range(self.n_iters):
    #         linear_model = np.dot(X, self.weights) + self.bias
    #         y_predicted = self.sigmoid(linear_model)

    #         # compute gradients
    #         dw = (1 / n_samples) * np.dot(X.T, (y_predicted - y))
    #         db = (1 / n_samples) * np.sum(y_predicted - y)

    #         # update parameters
    #         self.weights -= self.lr * dw
    #         self.bias -= self.lr * db

    # def predict(self, X):
    #     linear_model = np.dot(X, self.weights) + self.bias
    #     y_predicted = self.sigmoid(linear_model)
    #     y_predicted_cls = [1 if i > 0.5 else 0 for i in y_predicted]
    #     return y_predicted_cls
    # sigmoid function
    def __init__(self, lr = 1e-7, epochs = 2000):
        self.lr = lr
        self.epochs = epochs
    g = lambda z : np.exp(z) / (1+np.exp(z))

    def predict_prob(self,X,w):
      z = np.dot(X,w)
      a = g(z)
      return a
    def predict(self,X,w):
      y_hat = self.predict_prob(X,w)
      y_hat[y_hat >=0.5] =1
      y_hat[y_hat <0.5] =0
      return y_hat
    # loss function
    def loss(self,X,y,w):
      y_hat = self.predict_prob(X,w)
      l = y*np.log(y_hat) + (1-y)*np.log(1-y_hat)
      return -np.mean(l)
      # gradient
    def grad(self,X,y,w):
      y_hat = self.predict_prob(X,w)
      delta = y_hat - y
      dw = np.dot(X.T,delta)
      return dw
    def gradient(self,X,y,lr = 1e-7, epochs = 2000):
      w = np.zeros((X.shape[1],1))
      losses = []
      for i in range(epochs):
        dw = self.grad(X,y,w)
        w -= lr*dw
        a = self.loss(X,y,w)
        losses.append(a)
      return losses,w
```

In [23]:

```
batch_generator(X, y, 12500)
```

Out[23]:

```
[(array([[[  8,  10,  11, ..., 177, 174, 166],
          [  9,  11,  12, ..., 177, 173, 167],
          [  9,  11,  12, ..., 179, 172, 173],
          ...,
          [ 54,  45,  55, ..., 191, 184, 177],
          [ 49,  46,  51, ..., 188, 179, 174],
          [ 52,  50,  45, ..., 182, 171, 171]],

         [[174, 176, 176, ...,  36,  33,  53],
          [172, 172, 172, ..., 125, 135, 127],
          [168, 166, 167, ..., 126, 131, 117],
          ...,
          [175, 186, 183, ..., 128, 112, 119],
          [179, 188, 198, ..., 134, 133, 138],
          [192, 187, 187, ..., 151, 153, 147]],

         [[155, 166, 179, ..., 126, 119, 114],
          [161, 168, 172, ..., 126, 125, 117],
          [167, 174, 185, ..., 131, 123, 123],
          ...,
          [237, 241, 240, ..., 148, 149, 178],
          [230, 239, 241, ..., 155, 186, 193],
          [227, 237, 242, ..., 192, 195, 195]],

         ...,

         [[132, 131, 137, ..., 138, 131, 134],
          [135, 132, 127, ..., 137, 131, 135],
          [131, 127, 130, ..., 136, 125, 138],
          ...,
          [139, 141, 146, ...,  86,  87,  89],
          [132, 131, 138, ...,  85,  93,  97],
          [133, 133, 136, ..., 138, 137, 144]],

         [[166, 168, 168, ..., 151, 153, 156],
          [165, 168, 168, ..., 153, 152, 155],
          [165, 168, 168, ..., 155, 158, 158],
          ...,
          [ 48,  43,  22, ..., 224, 217, 181],
          [ 51,  38,  29, ..., 221, 210, 204],
          [ 66,  51,  31, ..., 214, 191, 209]],

         [[ 42,  41,  48, ...,  28,  28,  31],
          [ 40,  38,  45, ...,  28,  28,  31],
          [ 39,  41,  42, ...,  27,  28,  30],
          ...,
          [ 19,  19,  21, ..., 114, 105,  90],
          [ 19,  19,  21, ..., 111,  97,  90],
          [ 17,  18,  20, ...,  99,  94,  95]]], dtype=uint8),
  array([0, 0, 0, ..., 0, 0, 0])),
 (array([[[193, 187, 129, ..., 185, 183, 184],
          [196, 190, 126, ..., 187, 185, 184],
          [192, 189, 131, ..., 189, 186, 186],
          ...,
          [168, 208, 221, ...,  49,  45,  41],
          [197, 198, 227, ...,  29,  30,  38],
          [145, 195, 239, ...,  28,  30,  39]],

         [[119, 121, 118, ...,  62,  54,  51],
          [120, 119, 118, ...,  52,  62,  65],
          [120, 117, 120, ...,  58,  63,  68],
          ...,
          [ 67,  69,  70, ...,  88,  67,  64],
          [ 69,  66,  59, ...,  88,  97,  79],
          [ 69,  72,  59, ...,  78,  98,  89]],

         [[ 99, 123,  88, ..., 180, 195, 190],
          [ 64,  91,  88, ..., 201, 141, 161],
          [108, 124, 131, ..., 155, 154, 149],
          ...,
          [110, 112, 112, ..., 227, 229, 221],
          [107, 110, 112, ..., 242, 238, 229],
```

```
         [ 99, 103, 105, ..., 225, 221, 222]],

        ...,

        [[ 62,  60, 211, ...,  96, 174, 170],
         [ 57,  60, 207, ..., 173, 173, 174],
         [ 55,  55, 210, ..., 175, 183, 175],
         ...,
         [148, 139, 133, ..., 117, 109, 107],
         [139, 141, 136, ..., 108, 120, 112],
         [137, 139, 139, ..., 123, 120, 111]],

        [[ 70,  63, 236, ..., 177, 196, 227],
         [ 64,  56, 247, ..., 173, 224, 232],
         [ 60,  63, 243, ..., 188, 180, 172],
         ...,
         [ 87,  95, 145, ..., 149, 151, 150],
         [116,  86, 133, ..., 174, 212, 134],
         [113, 131, 101, ..., 196, 118, 142]],

        [[ 65,  76,  83, ...,  21,  10,  10],
         [ 68,  73,  82, ...,  21,  18,  18],
         [ 69,  80,  85, ...,  23,  25,  17],
         ...,
         [  7,  20,  13, ...,  78,  62,  61],
         [ 27,  29,  11, ...,  72,  69,  59],
         [ 21,   5,  36, ...,  42,  61,  63]]], dtype=uint8),
  array([1, 1, 1, ..., 1, 1, 1]))]
```

In [24]:

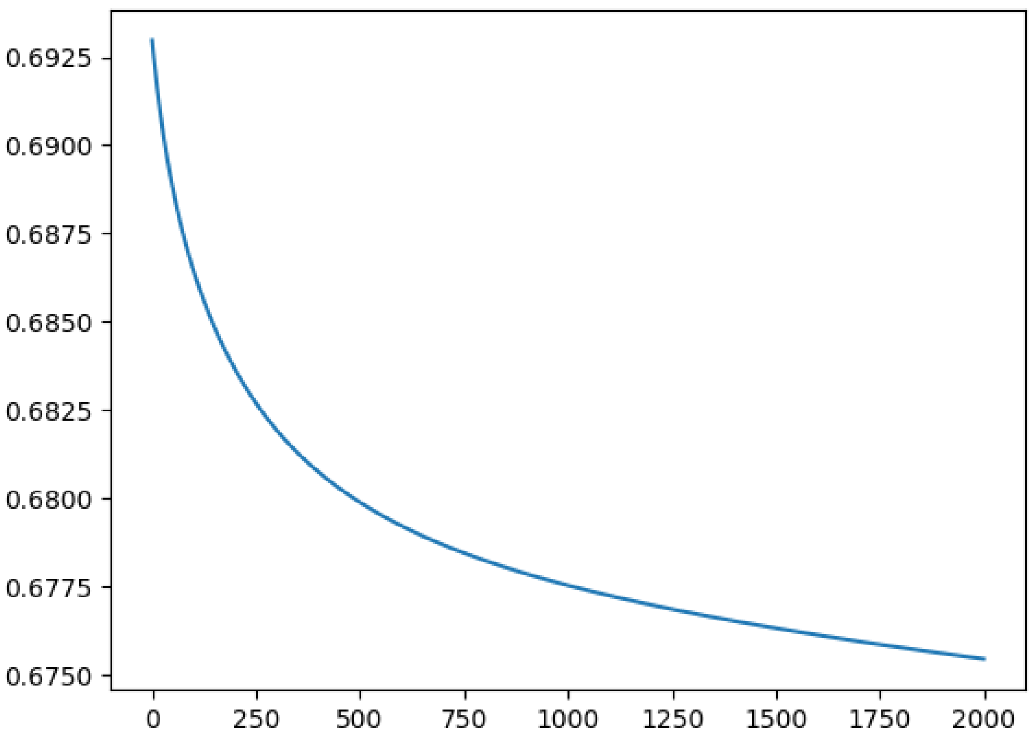
```
# model = binary1()
# model.fit(X_train_scaled, y_train)
```

In [34]:

```
model = binary1()
loss,w = model.gradient(X_train_scaled,np.array([y_train_scaled]).T)
plt.plot(loss)
```

Out[34]:

```
[<matplotlib.lines.Line2D at 0x7fa965b9eaf0>]
```



In [35]:

```
y_hat = model.predict(X_test_scaled,w)
```

In [36]:

```
evaluation(y_test, y_hat)
```

```
Accuracy = 0.5696
Precision = 0.5641124374278013
Recall = 0.5895372233400402
F1 Score = 0.5765446674537583
```