```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import cv2
import glob
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, recall_score, f1_score
```

Activity 1:

```
!unzip /content/drive/MyDrive/Data/Test_1.zip -d /content/sample_data
```

Show hidden output

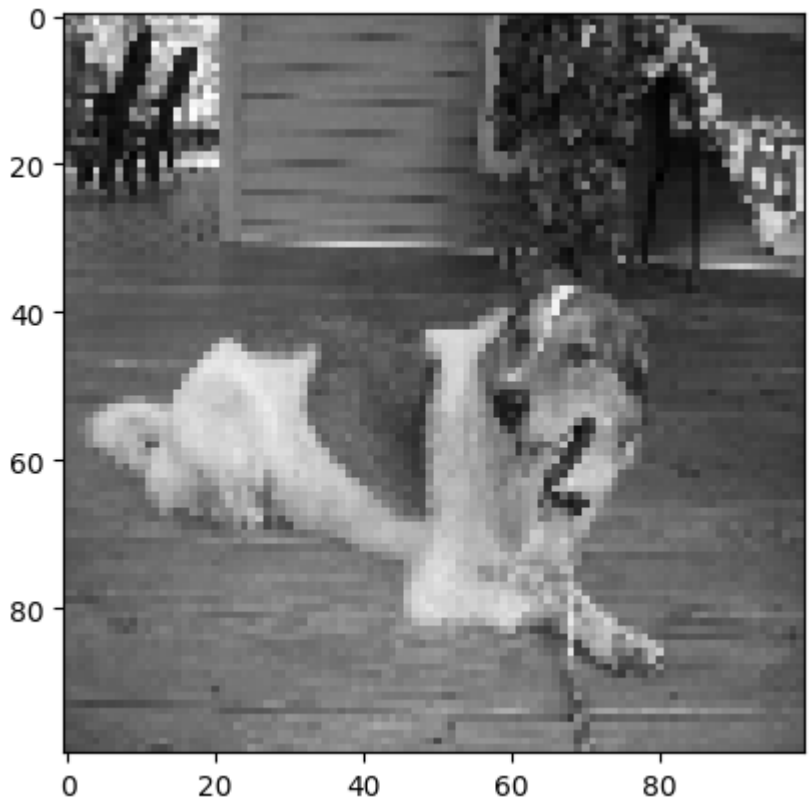
```
!unzip /content/drive/MyDrive/Data/Train_1.zip -d /content/sample_data
```

Show hidden output

```
path = "/content/sample_data/Train_1"

X = []
name_lable = []

for i in glob.glob(path + "/*"):
  img = cv2.imread(i)
  img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  img = cv2.resize(img, (100, 100))
  X.append(img)

  name = i.split("/")[-1]
  new_name = name.split(".")[0]
  if new_name == "cat":
    name_lable.append(0)
  else:
    name_lable.append(1)
```

```
print(name_lable[0])
plt.imshow(X[0], cmap="gray")
```

```
1
<matplotlib.image.AxesImage at 0x7b7711f97040>
```

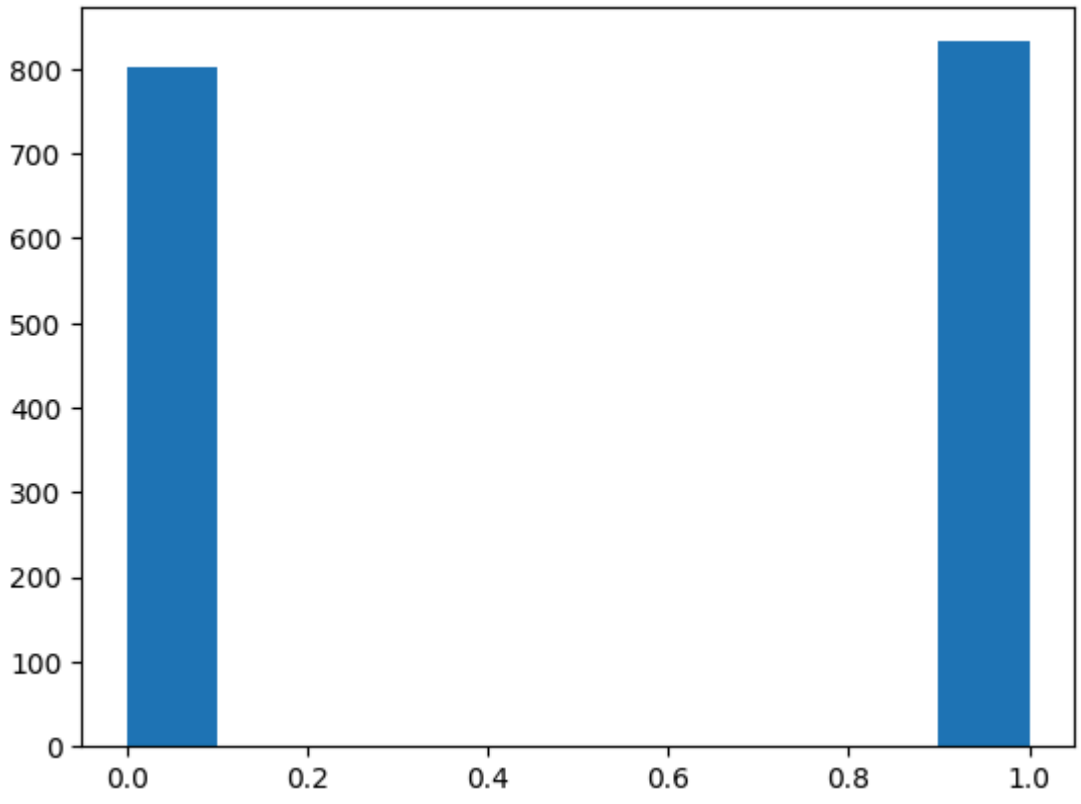



```
print(np.unique(name_lable))
```

```
[0 1]
```

```
plt.hist(name_lable)
```

```
(array([803.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0., 832.]),
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
 <BarContainer object of 10 artists>)
```




```
X = np.array(X)
y = np.array(name_lable)
print(X.shape)
print(y.shape)
```

```
(1635, 100, 100)
(1635,)
```

```
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)
```

```
(1308, 100, 100) (327, 100, 100)
(1308,) (327,)
```

```
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])
```

```
X_train_scaled.shape, X_test_scaled.shape
```

```
((1308, 10000), (327, 10000))
```

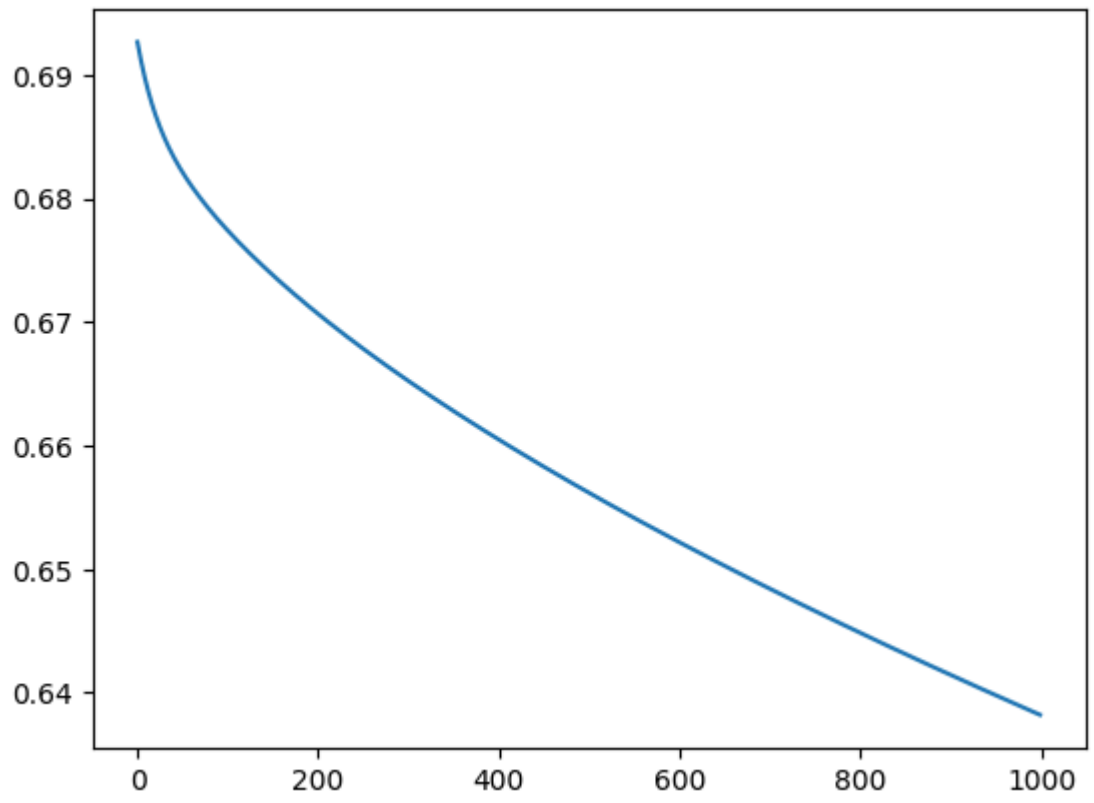
```
g = lambda z : np.exp(z) / (1 + np.exp(z))
```

```
def predict_prob(X, w):
    z = np.dot(X, w)
    return g(z)

def predict(X, w):
    y_hat = predict_prob(X, w)
    y_hat = np.where(y_hat >= 0.5, 1, 0)
    return y_hat

def loss(X, y, w):
    y_hat = predict_prob(X, w)
    # Giới hạn y_hat để tránh log(0) và log(1-1)
    y_hat = np.clip(y_hat, 1e-15, 1 - 1e-15)
    l = y * np.log(y_hat) + (1 - y) * np.log(1 - y_hat)
    return -np.mean(l)

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

def gradient_descent(X, y, lr=0.02, epochs=1000):
    w = np.zeros((X.shape[1], 1))
    losses = []
```

```
    for i in range(epochs):
        dw = grad(X, y, w)
        w -= lr * dw
        losses.append(loss(X, y, w))
    return losses, w
```

```
l, w = gradient_descent(X_train_scaled, y_train.reshape(-1, 1), lr=0.000001, epochs=1000)
plt.plot(l)
```

```
[<matplotlib.lines.Line2D at 0x7b770a8e3fa0>]
```



Bài tập 1

```
y_pred = predict(X_train_scaled, w)

accuracy = accuracy_score(y_train, y_pred)
recall = recall_score(y_train, y_pred)
f1 = f1_score(y_train, y_pred)

print("Train:")
print(f"Accuracy: {accuracy}")
print(f"Recall: {recall}")
print(f"F1-score: {f1}")

y_pred = predict(X_test_scaled, w)

accuracy = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("\nTest:")
print(f"Accuracy: {accuracy}")
print(f"Recall: {recall}")
print(f"F1-score: {f1}")
```

```
Train:
Accuracy: 0.6659021406727829
Recall: 0.6641337386018237
F1-score: 0.6666666666666666

Test:
Accuracy: 0.5290519877675841
Recall: 0.5402298850574713
F1-score: 0.5497076023391813
```

Bài tập 2

```
def batch_generator(X, y, batch_size=32):
  num_samples = len(X)
  for i in range(0, num_samples, batch_size):
    yield X[i:i + batch_size], y[i:i + batch_size]
```

Bài tập 3

```
class binary1:
    def __init__(self, lr=0.000003, epochs=1000, reg_lambda=0.01, batch_size=32, tol=1e-3):
        self.reg_lambda = reg_lambda
        self.batch_size = batch_size
        self.epochs = epochs
        self.tolerance = tol
        self.losses = []
        self.w = None
        self.lr = lr

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

    def predict_prob(self, X):
        z = np.dot(X, self.w)
        return self.g(z)

    def predict(self, X):
        y_hat = self.predict_prob(X)
        y_hat = np.where(y_hat >= 0.5, 1, 0)
        return y_hat
```

```
    def loss(self, X, y):
        y_hat = self.predict_prob(X)
        data_loss = -(y * np.log(y_hat) + (1 - y) * np.log(1 - y_hat))
        reg_loss = (self.reg_lambda / 2) * np.sum(self.w ** 2)
        return np.mean(data_loss) + reg_loss

    def grad(self, X, y):
        y_hat = self.predict_prob(X)
        delta = y_hat - y
        dw = np.dot(X.T, delta) + self.reg_lambda * self.w
        return dw

    def fit(self, X, y):
        X_bias = np.c_[X, np.ones(X.shape[0])]
        self.w = np.zeros((X_bias.shape[1], 1))
        for i in range(self.epochs):
            for X_batch, y_batch in batch_generator(X_bias, y, self.batch_size):
                dw = self.grad(X_batch, y_batch)
                self.w -= self.lr * dw
                self.losses.append(self.loss(X_batch, y_batch))

    def evaluate(self, X, y):
        X_bias = np.c_[X, np.ones(X.shape[0])]
        y_pred = self.predict(X_bias)
        accuracy = accuracy_score(y, y_pred)
        recall = recall_score(y, y_pred)
        f1 = f1_score(y, y_pred)
        print(f"Accuracy: {accuracy}")
        print(f"Recall: {recall}")
        print(f"F1-score: {f1}")
```

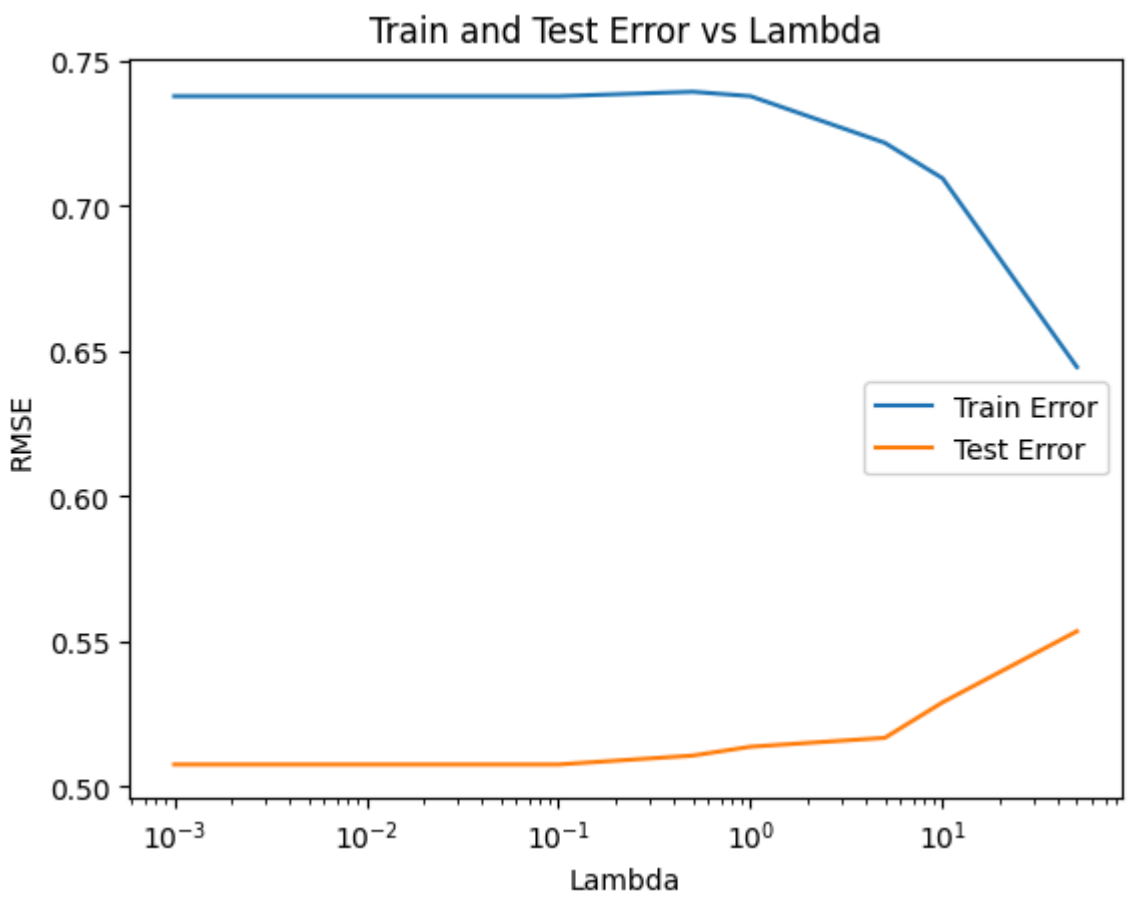
```
model = binary1(reg_lambda=0.05)
model.fit(X_train_scaled, y_train.reshape(-1, 1))
```

```
print("Train:")
model.evaluate(X_train_scaled, y_train)
print("\nTest:")
model.evaluate(X_test_scaled, y_test)
```

```
Train:
Accuracy: 0.7377675840978594
Recall: 0.7340425531914894
F1-score: 0.7379679144385026

Test:
Accuracy: 0.5076452599388379
Recall: 0.5229885057471264
F1-score: 0.5306122448979592
```

```
lambdas = [0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50]
train_errors = []
test_errors = []
```

```
for reg_lambda in lambdas:
    model = binary1(reg_lambda=reg_lambda)
    model.fit(X_train_scaled, y_train.reshape(-1, 1))
    y_pred_train = model.predict(np.c_[X_train_scaled, np.ones(X_train_scaled.shape[0])])
    y_pred_test = model.predict(np.c_[X_test_scaled, np.ones(X_test_scaled.shape[0])])

    train_error = accuracy_score(y_train, y_pred_train)
    test_error = accuracy_score(y_test, y_pred_test)

    train_errors.append(train_error)
    test_errors.append(test_error)

plt.plot(lambdas, train_errors, label='Train Error')
plt.plot(lambdas, test_errors, label='Test Error')
plt.xlabel('Lambda')
plt.ylabel('Accuracy')
plt.title('Train and Test Error vs Lambda')
plt.legend()
plt.xscale('log')
plt.show()
```




```
model = binary1(reg_lambda=0.1)
model.fit(X_train_scaled, y_train.reshape(-1, 1))

print("Train:")
model.evaluate(X_train_scaled, y_train)
print("\nTest:")
model.evaluate(X_test_scaled, y_test)
```

```
Train:
Accuracy: 0.7377675840978594
Recall: 0.7340425531914894
F1-score: 0.7379679144385026

Test:
Accuracy: 0.5076452599388379
Recall: 0.5229885057471264
F1-score: 0.5306122448979592
```

Activity 2:

```
data = pd.read_csv("smsspamcollection.tsv", sep='\t')
data.head()
```

	label	message	length	punct
0	ham	Go until jurong point, crazy.. Available only ...	111	9
1	ham	Ok lar... Joking wif u oni...	29	6
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	155	6
3	ham	U dun say so early hor... U c already then say...	49	6
4	ham	Nah I don't think he goes to usf, he lives aro...	61	2

Next steps: Generate code with `data` | View recommended plots | New interactive sheet

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   label    5572 non-null   object
 1   message  5572 non-null   object
 2   length   5572 non-null   int64 
 3   punct    5572 non-null   int64 
dtypes: int64(2), object(2)
memory usage: 174.2+ KB
```

```
np.unique(data["label"])
```

```
array(['ham', 'spam'], dtype=object)
```

```
data['message'] = data['message'].apply(lambda x: x.lower())
data['message'] = data['message'].str.replace('[.,?*!#@]', '', regex=True)
```

```
data_ham = data[data["label"] == "ham"]
test_ham = data_ham.sample(frac=0.2, random_state=42)
train_ham = data_ham.drop(test_ham.index)

data_spam = data[data["label"] == "spam"]
test_spam = data_spam.sample(frac=0.2, random_state=42)
train_spam = data_spam.drop(test_spam.index)
```

```
print(test_ham.shape, test_spam.shape)
print(train_ham.shape, train_spam.shape)
```

```
(965, 4) (149, 4)
(3860, 4) (598, 4)
```

```
train_data = pd.concat([train_ham, train_spam])
test_data = pd.concat([test_ham, test_spam])
```

```
print(train_data.shape, test_data.shape)
```

```
(4458, 4) (1114, 4)
```

```
vocab = []

for message in train_data.message.values:
    for word in message.split():
        if word not in vocab:
            vocab.append(word)

len(vocab)
```

```
9095
```

```
vocab_size = len(vocab)
word2idx = {word: idx for idx, word in enumerate(vocab)}
X_train = np.zeros((len(train_data), vocab_size))

for i, message in enumerate(train_data.message.values):
    for word in message.split():
        if word in vocab:
            X_train[i, word2idx[word]] += 1
```

```
X_train.shape
```

```
(4458, 9095)
```

```
X_test = np.zeros((len(test_data), vocab_size))

for i, message in enumerate(test_data.message.values):
    for word in message.split():
        if word in vocab:
            X_test[i, word2idx[word]] += 1

X_test.shape
```

```
(1114, 9095)
```

```
X_min = X_train.min(axis=0, keepdims=True)
X_max = X_train.max(axis=0, keepdims=True)
X_train_scaled = (X_train - X_min) / (X_max - X_min)
X_test_scaled = (X_test - X_min) / (X_max - X_min)
```

```
y_train = train_data.loc[:, ["label"]].values
y_test = test_data.loc[:, ["label"]].values
print(y_train.shape, y_test.shape)
```

```
(4458, 1) (1114, 1)
```

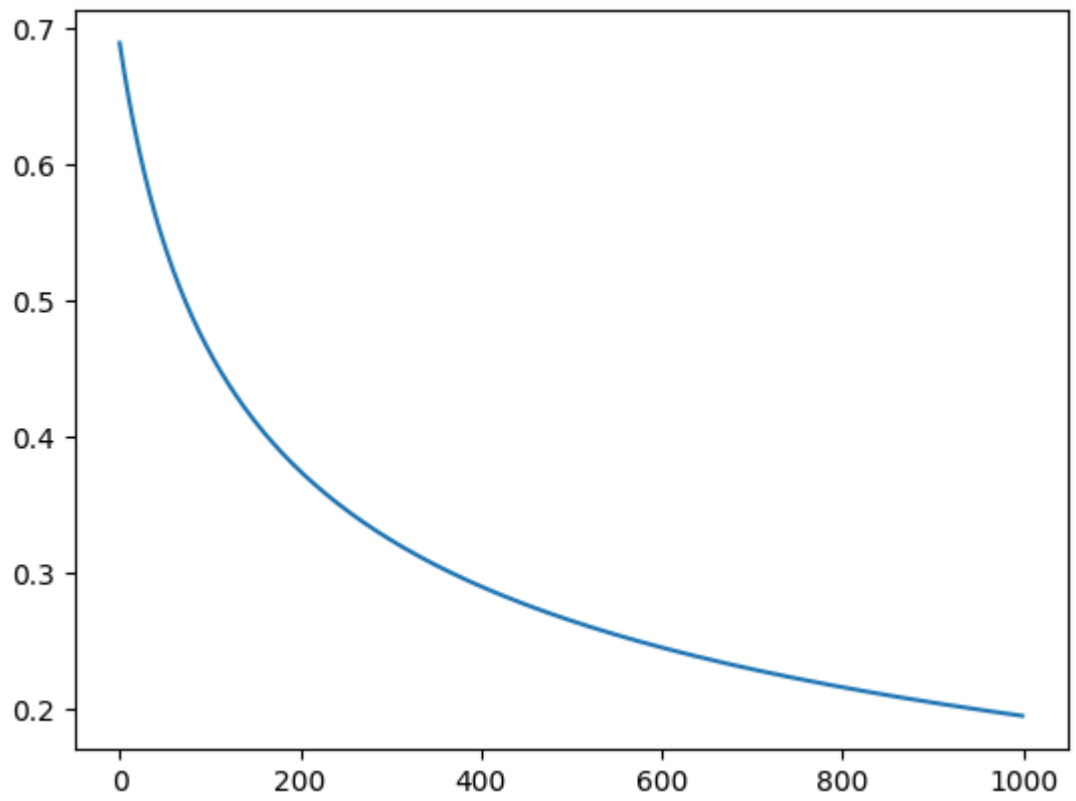
```
y_train_new = np.where(y_train == "ham", 0, 1)
y_test_new = np.where(y_test == "ham", 0, 1)
```

model

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

def predict_prob(X, w):
    z = np.dot(X, w)
    return g(z)

def predict(X, w):
    y_hat = predict_prob(X, w)
    y_hat = np.where(y_hat >= 0.5, 1, 0)
    return y_hat

def loss(X, y, w):
    y_hat = predict_prob(X, w)
    y_hat = np.clip(y_hat, 1e-15, 1 - 1e-15)
    l = y * np.log(y_hat) + (1 - y) * np.log(1 - y_hat)
    return -np.mean(l)

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

def gradient_descent(X, y, lr=0.02, epochs=1000):
    w = np.zeros((X.shape[1], 1))
    losses = []
    for i in range(epochs):
        dw = grad(X, y, w)
        w -= lr * dw
        losses.append(loss(X, y, w))
    return losses, w
```

```
losses, weights = gradient_descent(X_train_scaled, y_train_new.reshape(-1, 1), lr=0.0001, epochs=1000)
plt.plot(losses)
```

```
[<matplotlib.lines.Line2D at 0x796df79d7910>]
```



Bài 1

```
y_pred = predict(X_train_scaled, weights)

accuracy = accuracy_score(y_train_new, y_pred)
recall = recall_score(y_train_new, y_pred)
f1 = f1_score(y_train_new, y_pred)

print("Train:")
print(f"Accuracy: {accuracy}")
print(f"Recall: {recall}")
print(f"F1-score: {f1}")

y_pred = predict(X_test_scaled, weights)

accuracy = accuracy_score(y_test_new, y_pred)
recall = recall_score(y_test_new, y_pred)
f1 = f1_score(y_test_new, y_pred)

print("\nTest:")
print(f"Accuracy: {accuracy}")
print(f"Recall: {recall}")
print(f"F1-score: {f1}")
```

```
Train:
Accuracy: 0.9831763122476447
Recall: 0.8879598662207357
F1-score: 0.9340369393139841

Test:
Accuracy: 0.9640933572710951
Recall: 0.8120805369127517
F1-score: 0.8581560283687943
```

Bài 2:

```
def batch_generator(X, y, batch_size=32):
  num_samples = len(X)
  for i in range(0, num_samples, batch_size):
    yield X[i:i + batch_size], y[i:i + batch_size]
```

Bài 3:

```
class binary1:
    def __init__(self, lr=0.000003, epochs=1000, reg_lambda=0.01, batch_size=32, tol=1e-3)
        self.reg_lambda = reg_lambda
        self.batch_size = batch_size
        self.epochs = epochs
```

```
        self.tolerance = tol
        self.losses = []
        self.w = None
        self.lr = lr

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

    def predict_prob(self, X):
        z = np.dot(X, self.w)
        return self.g(z)

    def predict(self, X):
        y_hat = self.predict_prob(X)
        y_hat = np.where(y_hat >= 0.5, 1, 0)
        return y_hat

    def loss(self, X, y):
        y_hat = self.predict_prob(X)
        data_loss = -(y * np.log(y_hat) + (1 - y) * np.log(1 - y_hat))
        reg_loss = (self.reg_lambda / 2) * np.sum(self.w ** 2)
        return np.mean(data_loss) + reg_loss

    def grad(self, X, y):
        y_hat = self.predict_prob(X)
        delta = y_hat - y
        dw = np.dot(X.T, delta) + self.reg_lambda * self.w
        return dw

    def fit(self, X, y):
        X_bias = np.c_[X, np.ones(X.shape[0])]
        self.w = np.zeros((X_bias.shape[1], 1))
        for i in range(self.epochs):
            for X_batch, y_batch in batch_generator(X_bias, y, self.batch_size):
                dw = self.grad(X_batch, y_batch)
                self.w -= self.lr * dw
                self.losses.append(self.loss(X_batch, y_batch))

    def evaluate(self, X, y):
        X_bias = np.c_[X, np.ones(X.shape[0])]
        y_pred = self.predict(X_bias)
        accuracy = accuracy_score(y, y_pred)
        recall = recall_score(y, y_pred)
        f1 = f1_score(y, y_pred)
        print(f"Accuracy: {accuracy}")
        print(f"Recall: {recall}")
        print(f"F1-score: {f1}")
```

```
model = binary1(lr=0.0005, reg_lambda=0.05)
model.fit(X_train_scaled, y_train_new.reshape(-1, 1))
```

```
print("Train:")
model.evaluate(X_train_scaled, y_train_new)
print("\nTest:")
model.evaluate(X_test_scaled, y_test_new)
```

```
Train:
Accuracy: 0.9784656796769852
Recall: 0.8394648829431438
F1-score: 0.9127272727272727

Test:
Accuracy: 0.9649910233393177
Recall: 0.738255033557047
F1-score: 0.8494208494208494
```