```
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt



class LogisticRegression:
  def __init__(self, lr = 0.01, batch_size = 16, iters = 10):
    #initialize the learning rate, batch size, iterations, and weights
    self.lr = lr
    self.batch_size = batch_size
    self.iters = iters
    self.weights = None

  def sigmoid(self, x):
    #create function from sigmoid
    return 1/(1+np.exp(-x))

  def loss(self, y, y_pred):
    #calculate loss of a data sample
    return (-1/y.shape[0])*np.sum(y*np.log(y_pred+1e-8) + (1-y)*np.log(1-y_pred+1e-8))

  def grad(self, X, y, y_pred):
    #calculate gradient of data sample with identity feature mapping
    return (-1/y.shape[0])*(X.T).dot(y-y_pred)

  def predict(self, X):
    #predict results based on weights
    return self.sigmoid(np.dot(X, self.weights))


  def train(self, X, y):
    samples, features = X.shape
    self.weights = np.random.normal(size = features)

    for i in range(self.iters):
      #shuffle indices
      inds = np.arange(samples)
      np.random.shuffle(inds)
      X_alt = X[inds]
      y_alt = y[inds]

      for j in range(samples//self.batch_size):
        #create batches
        X_batch = X_alt[j*self.batch_size:(j+1)*self.batch_size]
        y_batch = y_alt[j*self.batch_size:(j+1)*self.batch_size]

        #predict results of batch using current weights
        y_pred = self.predict(X_batch)

        #update gradient
        grad = self.grad(X_batch, y_batch, y_pred)
        self.weights -= self.lr*grad

      #calculate loss
      y_pred = self.predict(X)
      loss = self.loss(y, y_pred)

    #print("Final Loss: ", loss, " for ", self.lr, " ", self.batch_size, " ", self.iters)



data = load_breast_cancer()
X = data['data']
y = data['target']
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size = 0.2)
X_val, X_test, y_val, y_test = train_test_split(X_val, y_val, test_size = 0.5)

print("X train size: ", X_train.shape)
print("y train size: ", y_train.shape)
print("X validation size: ", X_val.shape)
print("y validation size: ", y_val.shape)
print("X test size: ", X_test.shape)
print("y test size: ", y_test.shape)
```

```
X train size:  (455, 30)
y train size:  (455,)
X validation size:  (57, 30)
y validation size:  (57,)
X test size:  (57, 30)
y test size:  (57,)
```

```
for lr in [1, 1e-2, 1e-3, 1e-4, 1e-5]:
  for batch_size in [4, 8, 16, 32, 64]:
    error = 0
    for i in range(10):
      LR = LogisticRegression(lr = lr, batch_size = batch_size)
      LR.train(X_train, y_train)

      y_pred = LR.predict(X_val)
      y_pred = np.where(y_pred > 0.5, 1, 0)
      error += np.sum(y_pred != y_val)/y_val.shape[0]

    print("Error: ", error/10, " for ", lr, " ", batch_size)
```

```
<ipython-input-109-472a90248188>:9: RuntimeWarning: overflow encountered in exp
  return 1/(1+np.exp(-x))
Error:  0.2192982456140351  for  1   4
Error:  0.2245614035087719  for  1   8
Error:  0.23684210526315788  for  1   16
Error:  0.28771929824561404  for  1   32
Error:  0.41052631578947363  for  1   64
Error:  0.19649122807017544  for  0.01   4
Error:  0.15438596491228068  for  0.01   8
Error:  0.30877192982456136  for  0.01   16
Error:  0.3280701754385965  for  0.01   32
Error:  0.43508771929824563  for  0.01   64
Error:  0.21052631578947367  for  0.001   4
Error:  0.18771929824561404  for  0.001   8
Error:  0.20526315789473681  for  0.001   16
Error:  0.34385964912280703  for  0.001   32
Error:  0.36140350877192984  for  0.001   64
Error:  0.18596491228070172  for  0.0001   4
Error:  0.25614035087719295  for  0.0001   8
Error:  0.2649122807017544  for  0.0001   16
Error:  0.4736842105263158  for  0.0001   32
Error:  0.44035087719298244  for  0.0001   64
Error:  0.3421052631578948  for  1e-05   4
Error:  0.33508771929824566  for  1e-05   8
Error:  0.41052631578947363  for  1e-05   16
Error:  0.4491228070175438  for  1e-05   32
Error:  0.5333333333333333  for  1e-05   64
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```
#choose lr of 0.01 and batch size of 8 beacause it gives lowest error rate
LR = LogisticRegression(lr = 0.01, batch_size = 8)
loss = LR.train(X_train, y_train)

y_pred = LR.predict(X_test)
y_pred = np.where(y_pred > 0.5, 1, 0)
```

```
print("Accuracy: ", accuracy_score(y_test, y_pred))
print("Precision: ", precision_score(y_test, y_pred))
print("Recall: ", recall_score(y_test, y_pred))
print("F1 Score: ", f1_score(y_test, y_pred))
```

```
Accuracy:  0.8421052631578947
Precision:  0.7857142857142857
Recall:  1.0
F1 Score:  0.88
<ipython-input-109-472a90248188>:9: RuntimeWarning: overflow encountered in exp
  return 1/(1+np.exp(-x))
```

The accuracy of the model is 0.84, which means that it will distinguish whether a patient has breast cancer or not 84% of the time. The precision is 0.78, which means there are a few false positive results. This will predict that some people have breast cancer when they do not actually have breast cancer The recall score is 1, which means there are no false negatives, so no one will be told they don't have breast cancer but they do. A high recall score is critical in the case of breast cancer since it can be deadly if breast cancer is missed. F1 score combines the precision and recall scores. In this case, the F1 score is 0.88.

Overall, the most important score is recall for this model. Since the recall is 1 for this model, it performs quite well.