```python
import scipy.io
import matplotlib.pyplot as plt
import numpy as np
from tqdm import tqdm
from sklearn.metrics import accuracy_score


class LogisticRegression:

    def __init__(self, X_train, y_train, X_test, y_test) -> None:
        self.X_train = X_train
        self.y_train = y_train
        self.X_test = X_test
        self.y_test = y_test

        self.X_train_original = self.X_train
        self.X_test_original = self.X_test

        self.X_train, self.X_test = self.normalize(self.X_train, self.X_test)

        # If our set is already flat, we don't need the next line
        # self.X_train, self.X_test = self.flatten_pixels(
        #     self.X_train, self.X_test)

        self.W, self.b = self.initialization(self.X_train)

        self.trained = False

    @staticmethod
    def initialization(X):
        W = np.random.randn(X.shape[1])  # np.random.randn(X.shape[1], 1)
        b = np.random.randn(1)[0]  # np.random.randn(1)[0]
        return (W, b)

    @staticmethod
    def normalize(train_set, test_set):
        return (train_set / train_set.max(), test_set / test_set.max())

    @staticmethod
    def flatten_pixels(train_set, test_set):
        train_set = train_set.reshape(
            train_set.shape[0], train_set.shape[1]*train_set.shape[2])
        test_set = test_set.reshape(
            test_set.shape[0], test_set.shape[1]*test_set.shape[2])
        return (train_set, test_set)

    @staticmethod
    def model(X, W, b):
        Z = X.dot(W) + b
        A = 1 / (1 + np.exp(-Z))
        return A

    @staticmethod
    def log_loss(A, y, epsilon=1e-15):
        return 1 / len(y) * np.sum(-y * np.log(A + epsilon) - (1-y) * np.log(1 - A +
epsilon))

    @staticmethod
    def gradients(A, X, y):
        dW = 1 / len(y) * np.dot(X.T, A - y)
        db = 1 / len(y) * np.sum(A - y)
        return (dW, db)

    @staticmethod
    def update(dW, db, W, b, learning_rate):
        W = W - learning_rate * dW
        b = b - learning_rate * db
        return (W, b)

    def train(self, learning_rate=1.2, n_iter=300):
        if self.trained:
            print("The model has been already trained.")
            return

        Loss = []
        history = []

        for i in tqdm(range(n_iter)):
```

```
            A = self.model(self.X_train, self.W, self.b)
            Loss.append(self.log_loss(A, self.y_train))
            dW, db = self.gradients(A, self.X_train, self.y_train)
            self.W, self.b = self.update(dW, db, self.W, self.b, learning_rate)
            history.append([self.W, self.b, Loss[i]])

        plt.plot(Loss)
        plt.show()
        y_pred = self.predict(self.X_train, self.W, self.b)
        print("Accuracy score: ", accuracy_score(self.y_train, y_pred))

        self.trained = True

    def predict(self, X, W, b):
        A = self.model(X, W, b)
        return A >= 0.5

    def show_train_set(self):
        plt.figure(figsize=(16, 8))
        for i in range(1, 15):
            plt.subplot(4, 5, i)
            plt.imshow(self.X_train_original[i], cmap='gray')
            plt.title("chien" if self.y_train[i] == 1.0 else "chat")
            plt.tight_layout()
        plt.show()

    def show_test_set(self):
        y_predict = self.predict(self.X_train, self.W, self.b)
        for i in range(1, 15):
            plt.subplot(4, 5, i)
            plt.imshow(self.X_train_original[i], cmap='gray')
            plt.title("chien" if y_predict[i] == 1.0 else "chat")
            plt.tight_layout()
        plt.show()


def retrive_data():
    mat = scipy.io.loadmat('mnist_all.mat')

    x1 = mat.get('train1')
    x2 = mat.get('train2')
    X_train = np.concatenate((x1, x2), axis=0)

    y1 = np.zeros(len(x1), dtype=int)
    y2 = np.ones(len(x2), dtype=int)
    y_train = np.concatenate((y1, y2), axis=0)

    x1_test = mat.get('test1')
    x2_test = mat.get('test2')
    X_test = np.concatenate((x1_test, x2_test), axis=0)

    y1_test = np.zeros(len(x1_test), dtype=int)
    y2_test = np.ones(len(x1_test), dtype=int)
    y_test = np.concatenate((y1_test, y2_test), axis=0)

    return X_train, y_train, X_test, y_test


if __name__ == "__main__":

    X_train, y_train, X_test, y_test = retrive_data()
    myModel = LogisticRegression(X_train, y_train, X_test, y_test)
    myModel.train()

```