# Homework #2

Hyungseok Choi
Student No. : 2250166

May 11, 2022

Collaborated with Grace Chen

## Short Answer and "True or False" Conceptual questions

A1.

a. Not necessarily. The coefficient magnitudes don't indicate variable importance and there might be substitute or even a better features that makes better model.

b. Penalizing least squares with L1 and L2 norm is equal to minimize$_w \sum_{i=1}^{n} \left(w^T x_i - y_i\right)^2$ with w subject to $||w||_1 \leq \mu$ and $||w||_2^2 \leq \mu$. Since L1 norm is pointy and L2 norm is smooth, L1 norm is more likely to result in a sparse solutions.

c. upside: more likely to result in a sparse solutions. (pointier than L1)
downside: more likely to get a large errors.

d. True

e. Even though with a small portion of the data is considered, if we select data randomly, we can estimate the mean with some errors. Therefore, iterating more with random data makes SGD work.

f. advantage of SGD over GD: The proper slope can be obtained with much less calculation.
disadvantage of SGD relative to GD: The noise of SGD is large.

## Convexity and Norms

A2.

a. $(i)$ $f(x) = (\sum_{i=1}^{n} |x_i|) = |x_1| + |x_2| + \cdots + |x_n|$
Since, each $|x_i| \geq 0$ with equality iff $x_i = 0$, the sum of each term also greater than or equal to 0 with equality iff $x_1 = 0, x_2 = 0, \cdots, x_n = 0$.

$(ii)$ $f(ax) = (\sum_{i=1}^{n} |ax_i|) = |ax_1| + |ax_2| + \cdots + |ax_n| = |a||x_1| + |a||x_2| + \cdots + |a||x_n| = |a|f(x)$

$(iii)$ $0 \leq |a+b|^2 = (a+b)^2 = a^2 + 2ab + b^2 \leq |a|^2 + 2|a||b| + |b|^2 = (|a| + |b|)^2$
Therefore, we know $|a+b| \leq |a| + |b|$.
$f(x+y) = (\sum_{i=1}^{n} |x_i + y_i|) = |x_1 + y_1| + |x_2 + y_2| + \cdots + |x_n + y_n|$
$\leq |x_1| + |x_2| + \cdots + |x_n| + |y_1| + |y_2| + \cdots + |y_n| \leq f(x) + f(y)$

From $(i), (ii), (iii)$, $f(x)$ is a norm.

b. When n = 2, $X_1 = (1, \frac{1}{2}), X_2 = (\frac{1}{2}, 1)$,
$g(X_1 + X_2) = \left(\frac{3}{2}^{\frac{1}{2}} + \frac{3}{2}^{\frac{1}{2}}\right)^2 = 6$ whereas $g(X_1) + g(X_2) = \left(1^{\frac{1}{2}} + \frac{1}{2}^{\frac{1}{2}}\right)^2 + \left(\frac{1}{2}^{\frac{1}{2}} + 1^{\frac{1}{2}}\right)^2 = 5.82$
Since it does not satisfy triangle inequality, it is not a norm.

A3.
I: Not convex. Since the part of line segment bc is located outside Figure I, it goes against the convex condition.
II: Convex.
III: Not convex. Since the part of line segment ad is located outside Figure III, it goes against the convex condition.

A4.

a. Function in panel I on $[a, c]$: Convex.
b. Function in panel II on $[a, c]$: Not convex - on $[a, b]$, $f(\frac{a+b}{2}) > \frac{f(a)+f(b)}{2}$.
c. Function in panel III on $[a, d]$: Not convex - on $[a, c]$, $f(\frac{a+c}{2}) > \frac{f(a)+f(c)}{2}$.
d. Function in panel III on $[c, d]$: Convex.

# Lasso on a Real Dataset

A5.

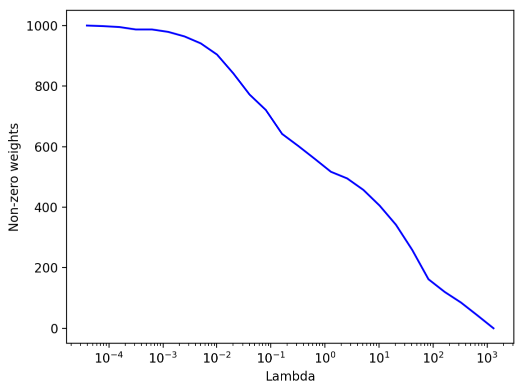


Figure 1: A number of non-zero weights with compare to lambda.

a.

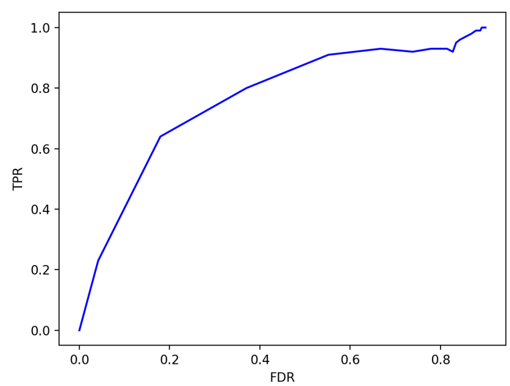


Figure 2: True positive rate vs False discovery rate

b.

c. $\lambda$ regularize the weights in lasso regression. What it means, is the more large $\lambda$, the more weights go to zero.

A6.

a. 1. perCapInc - changeable in accordance with minimum wage, inflation, and employment laws.
   2. PctImmigRecent - sensitive to recent immigrant and immigration policy
   3. PolicOperBudg - responded in accordance with the Police operational budget and Budget policies.

b. PolicPerPop - More police may have been deployed in areas with high crime levels.
   PolicReqPerOffic - The high total number of requests is an indicator that a crime has occurred. Therefore, it cannot be an indicator that causes the crime rate.
   PolicAveOTWorked - Same as the above.

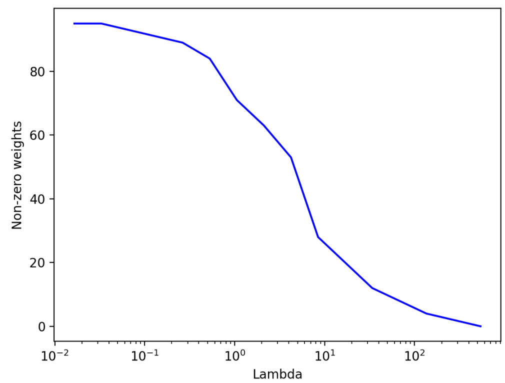


Figure 3: A number of non-zero weights with compare to lambda.

c.

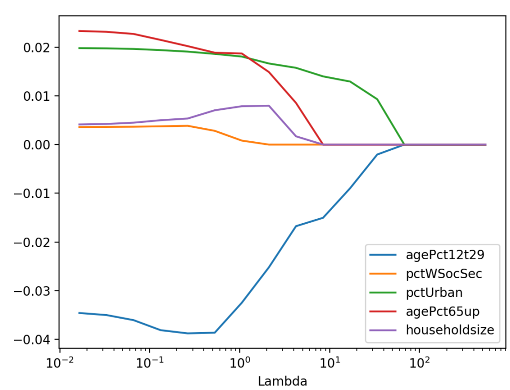


Figure 4: The regularization paths for the 5 variables

d.

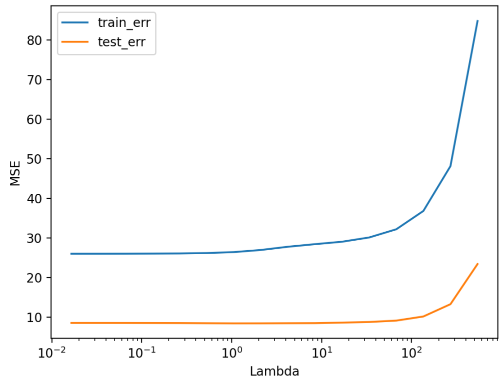


Figure 5: Squared error on the training and test data)

e.

f. The largest coefficient: PctIlleg: 0.06872
The most negative coefficient: PctKids2Par -0.06921

g. The causation between results and features cannot be explained through regression analysis.

```python
#coordinate_descent_algo.py

from typing import Optional, Tuple

import matplotlib.pyplot as plt
import numpy as np

from utils import problem

@problem.tag("hw2-A")
def precalculate_a(X: np.ndarray) -> np.ndarray:
    return pow(np.linalg.norm(X, axis=0), 2)*2


@problem.tag("hw2-A")
def step(
    X: np.ndarray, y: np.ndarray, weight: np.ndarray, a: np.ndarray, _lambda: float
) -> Tuple[np.ndarray, float]:
    n, d = X.shape
    bias = (y-X@weight).mean()

    for k in range(d):
        a_k = a[k]
        c_k = 0

        weight_copy = weight.copy()
        weight_copy[k] = 0

        for i in range(n):
            error = (y[i] - bias - weight_copy.T@X[i,:])
            c_k += X[i][k]*error
        c_k *= 2

        if (c_k < -_lambda):
            weight[k] = (c_k+_lambda)/a_k
        elif(c_k > _lambda):
            weight[k] = (c_k-_lambda)/a_k
        else:
            weight[k] = 0

    return (weight, bias)


@problem.tag("hw2-A")
def loss(
    X: np.ndarray, y: np.ndarray, weight: np.ndarray, bias: float, _lambda: float
) -> float:
    B = bias*np.ones(len(y))
    W = _lambda * np.linalg.norm(weight, ord=1)
    val = np.linalg.norm(X@weight+B-y)

    return pow(val, 2)+W


@problem.tag("hw2-A", start_line=4)
```

```python
def train(
    X: np.ndarray,
    y: np.ndarray,
    _lambda: float = 0.01,
    convergence_delta: float = 1e-4,
    start_weight: np.ndarray = None,
) -> Tuple[np.ndarray, float]:
    if start_weight is None:
        start_weight = np.zeros(X.shape[1])
    a = precalculate_a(X)
    old_w: Optional[np.ndarray] = None
    new_weight = start_weight

    while(True):
        old_w = np.copy(new_weight)
        new_weight, bias = step(X, y, new_weight, a, _lambda)
        if(convergence_criterion(new_weight, old_w, convergence_delta)):
            break

    return (new_weight, bias)


@problem.tag("hw2-A")
def convergence_criterion(
    weight: np.ndarray, old_w: np.ndarray, convergence_delta: float
) -> bool:
    return np.abs(weight-old_w).max() <= convergence_delta


@problem.tag("hw2-A")
def main():
    # set run_for var among "a" and "b"
    run_for = "b"

    X = np.random.normal(0,1, (500, 1000))
    weight = np.pad(np.linspace(0.01, 1, 100), (0,900))
    y = X@weight + np.random.normal(size=500)

    y_mean = y.mean()
    l_max = np.max(np.abs((X.T@(y-y_mean))))*2

    lambda_ = []
    non_zeros = []
    FDR = []
    TPR = []

    l = l_max
    w = np.zeros(len(weight))

    while (np.count_nonzero(w) < 1000):
        print(l)
        w, b = train(X, y, l, 0.0001, w)
        lambda_.append(l)

        icnz = np.count_nonzero(w[100:])
```

```python
        cnz = np.count_nonzero(w[:100])

        non_zeros.append(icnz + cnz)
        FDR.append(icnz/(icnz+cnz) if icnz != 0 else 0)
        TPR.append(cnz/100)
        l /= 2

    # plot the number of non-zero weights vs lambda curve
    if run_for == "a":
        plt.plot(lambda_, non_zeros, "b-")
        plt.xlabel("Lambda")
        plt.ylabel("Non-zero weights")
        plt.xscale('log')
        plt.show()

    # plot FDR vs TPR curve
    if run_for == "b":
        plt.figure()
        plt.plot(FDR, TPR, "b-")
        plt.xlabel("FDR")
        plt.ylabel("TPR")
        plt.show()


if __name__ == "__main__":
    main()
```

```python
#crime_data_lasso.py

if __name__ == "__main__":
    from coordinate_descent_algo import train  # type: ignore
else:
    from .coordinate_descent_algo import train

from re import A
import matplotlib.pyplot as plt
import numpy as np

from utils import load_dataset, problem


@problem.tag("hw2-A", start_line=3)
def main():
    # set run_for var among "c", "d", "e" and "f".
    run_for = "f"

    df_train, df_test = load_dataset("crime")

    train_y = df_train.iloc[:, 0].to_numpy()
    df_train = df_train.iloc[:,1:]

    columns = df_train.columns
    train_x = df_train.to_numpy()

    n, d = train_x.shape
    weight = np.zeros(d)

    y_mean = train_y.mean()
    l_max = np.max(np.abs((train_x.T@(train_y-y_mean))))*2

    lambda_ = []
    l = l_max

    if run_for == "c":
        non_zeros = []

    if run_for == "d":
        cols = ["agePct12t29", "pctWSocSec", "pctUrban", "agePct65up", "householdsize"]
        idx = [columns.get_loc(i) for i in cols]
        coefs = np.zeros((5,1))

    if run_for == "e":
        test_y = df_test.iloc[:, 0].to_numpy()
        test_x = df_test.iloc[:, 1:].to_numpy()
        mse = np.zeros((2,1))

    if run_for in ["c", "d", "e"]:
        while(l > 0.01):
            print(l)
            weight, b = train(train_x, train_y, l, 0.0001, weight)
            lambda_.append(l)
```

```
            # Plot nonzero weights (6-c)
            if run_for == "c":
                nz = np.count_nonzero(weight)
                non_zeros.append(nz)

            # Plot the regularization path (6-d)
            if run_for == "d":
                vals = np.asarray([weight[i] for i in idx]).reshape((-1,1))
                coefs = np.append(coefs, vals, 1)

            # Plot the squared error (6-e)
            if run_for == "e":
                B = b*np.ones(len(train_y))
                train_err = pow(np.linalg.norm(train_x@weight+B-train_y), 2)
                B = b*np.ones(len(test_y))
                test_err = pow(np.linalg.norm(test_x@weight+B-test_y), 2)
                val = np.asarray([train_err, test_err]).reshape((-1,1))
                mse = np.append(mse, val, 1)

            l /= 2

        # Plot the nonzero weights(6-c)
        if run_for == "c":
            plt.plot(lambda_, non_zeros, "-")
            plt.ylabel("Non-zero weights")

        # Plot the regularization path (6-d)
        if run_for == "d":
            plt.plot(lambda_, coefs[:,1:].T, "-")
            plt.legend(cols)
            plt.ylabel("Coefficient values")

        # Plot the squared error (6-e)
        if run_for == "e":
            plt.plot(lambda_, mse[:,1:].T, "-")
            plt.legend(["train_err", "test_err"])
            plt.ylabel("MSE")

        plt.xlabel("Lambda")
        plt.xscale('log')
        plt.show()
        return


    weight, b = train(train_x, train_y, 30, 0.0001)
    max_idx = np.argmax(weight)
    min_idx = np.argmin(weight)
    print(columns[max_idx], weight[max_idx], columns[min_idx], weight[min_idx])

if __name__ == "__main__":
    main()
```

# Logistic Regression

## Binary Logistic Regression

A7.

a.

$$\begin{aligned}
\nabla_b J(w,b) &= \frac{1}{n}\sum_{i=1}^{n} \frac{-y_i \exp -y_i(-y_i(b+x_i^T w))}{1+\exp -y_i(b+x_i^T w} \\
&= \frac{1}{n}\sum_{i=1}^{n} -y_i \left(\frac{1}{\mu(w,b)} - 1\right)\mu(w,b) \\
&= \frac{1}{n}\sum_{i=1}^{n} -y_i \left(1-\mu(w,b)\right)
\end{aligned}$$

$$\begin{aligned}
\nabla_{w_j} J(w,b) &= \frac{1}{n}\sum_{i=1}^{n} \left(\frac{-y_i x_{ij} \exp\left(-y_i(b+x_{ij}^T w_j)\right)}{1+\exp\left(-y_i(b+x_{ij}^T w_j)\right)}\right) + 2\lambda w_j \\
\nabla_w J(w,b) &= \frac{1}{n}\sum_{i=1}^{n} \left(-y_i x_i^T \left(\frac{1}{\mu(w,b)} - 1\right)\mu(w,b)\right) + 2\lambda w \\
&= \frac{1}{n}\sum_{i=1}^{n} \left(-y_i x_i^T \left(1-\mu(w,b)\right)\right) + 2\lambda w
\end{aligned}$$

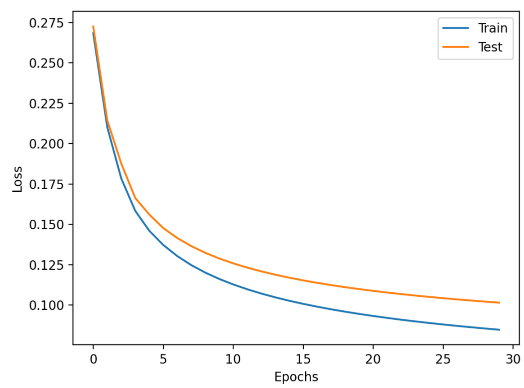


Figure 6: GD's $J(w,b)$ as a function of the iteration number(lr=0.4)

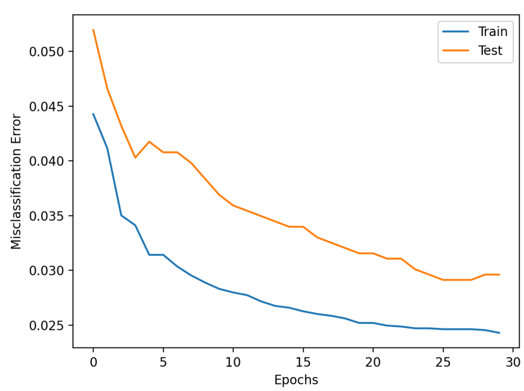


Figure 7: GD's $J(w,b)$ as a function of the iteration number(lr=0.4)

b.

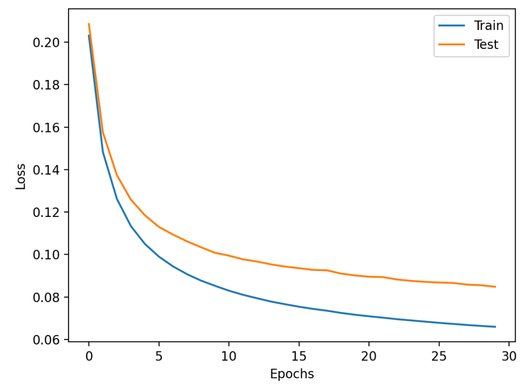


Figure 8: Batch size 1 SGD's $J(w, b)$ as a function of the iteration number(lr=0.0001

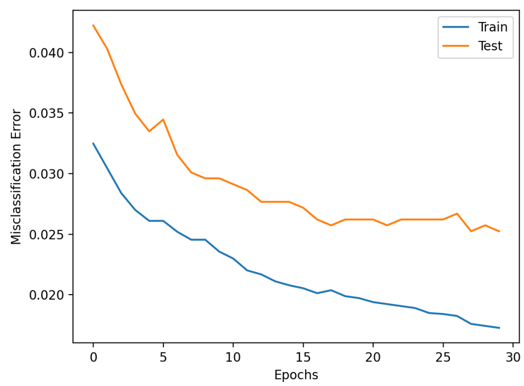


Figure 9: Batch size 1 SGD's $J(w, b)$ as a function of the iteration number(lr=0.0001)

c.

d.

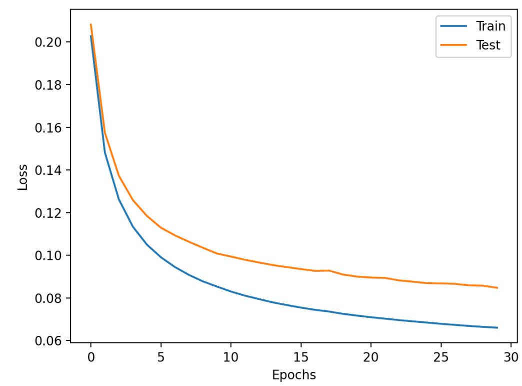


Figure 10: Batch size 100 SGD's $J(w, b)$ as a function of the iteration number(lr=0.01)

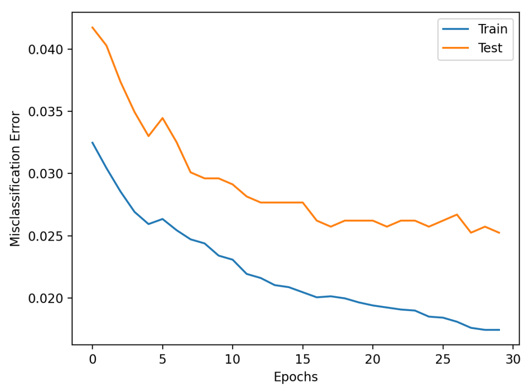


Figure 11: Batch size 100 SGD's $J(w, b)$ as a function of the iteration number(lr=0.01)

```python
# binary_log_regression.py

from textwrap import indent
from typing import Dict, List, Tuple

import matplotlib.pyplot as plt
import numpy as np

from utils import load_dataset, problem

# When choosing your batches / Shuffling your data you should use this RNG variable, and no
RNG = np.random.RandomState(seed=446)
Dataset = Tuple[Tuple[np.ndarray, np.ndarray], Tuple[np.ndarray, np.ndarray]]

def load_2_7_mnist() -> Dataset:
    (x_train, y_train), (x_test, y_test) = load_dataset("mnist")
    train_idxs = np.logical_or(y_train == 2, y_train == 7)
    test_idxs = np.logical_or(y_test == 2, y_test == 7)

    y_train_2_7 = y_train[train_idxs]
    y_train_2_7 = np.where(y_train_2_7 == 7, 1, -1)

    y_test_2_7 = y_test[test_idxs]
    y_test_2_7 = np.where(y_test_2_7 == 7, 1, -1)

    return (x_train[train_idxs], y_train_2_7), (x_test[test_idxs], y_test_2_7)


class BinaryLogReg:
    @problem.tag("hw2-A", start_line=4)
    def __init__(self, _lambda: float = 1e-3):
        self._lambda: float = _lambda
        # Fill in with matrix with the correct shape
        self.weight: np.ndarray = np.zeros(784)  # type: ignore
        self.bias: float = 0.0

    @problem.tag("hw2-A")
    def mu(self, X: np.ndarray, y: np.ndarray) -> np.ndarray:

        return 1/(1+np.exp(-y*(self.bias+X.dot(self.weight))))

    @problem.tag("hw2-A")
    def loss(self, X: np.ndarray, y: np.ndarray) -> float:

        return np.mean(np.log(1+np.exp(-y*(self.bias+X.dot(self.weight))))) + self._lambda *

    @problem.tag("hw2-A")
    def gradient_J_weight(self, X: np.ndarray, y: np.ndarray) -> np.ndarray:

        return np.mean(-y*(X.T)*(1-self.mu(X,y)), axis=1) + 2*self._lambda*self.weight


    @problem.tag("hw2-A")
    def gradient_J_bias(self, X: np.ndarray, y: np.ndarray) -> float:
```

```
        return np.mean(-y*(1-self.mu(X,y)))

    @problem.tag("hw2-A")
    def predict(self, X: np.ndarray) -> np.ndarray:

        ret = X.dot(self.weight)+self.bias
        ret = np.where(ret < 0, -1, ret)
        ret = np.where(ret >= 0, 1, ret)
        return ret

    @problem.tag("hw2-A")
    def misclassification_error(self, X: np.ndarray, y: np.ndarray) -> float:

        return 1-(np.sum(np.where(self.predict(X)!=y, 0, 1))/len(y))


    @problem.tag("hw2-A")
    def step(self, X: np.ndarray, y: np.ndarray, learning_rate: float = 1e-4):

        self.bias -= learning_rate * self.gradient_J_bias(X, y)
        self.weight -= learning_rate * self.gradient_J_weight(X, y)


    @problem.tag("hw2-A", start_line=7)
    def train(
        self,
        X_train: np.ndarray,
        y_train: np.ndarray,
        X_test: np.ndarray,
        y_test: np.ndarray,
        learning_rate: float = 1e-2,
        epochs: int = 30,
        batch_size: int = 100,
    ) -> Dict[str, List[float]]:

        num_batches = int(np.ceil(len(X_train) // batch_size))
        result: Dict[str, List[float]] = {
            "train_losses": [],  # You should append to these lists
            "train_errors": [],
            "test_losses": [],
            "test_errors": [],
        }

        for i in range(epochs):
            for j in range(num_batches):
                indices = set()
                while len(indices) < batch_size:
                    indices.add(RNG.choice(len(X_train)))

                indices = tuple(indices)
                X_t = X_train[[indices]]
                y_t = y_train[[indices]]
                self.step(X_t, y_t, learning_rate)

            result["train_losses"].append(self.loss(X_train, y_train))
```

```
            result["train_errors"].append(self.misclassification_error(X_train, y_train))
            result["test_losses"].append(self.loss(X_test, y_test))
            result["test_errors"].append(self.misclassification_error(X_test, y_test))

        return result

if __name__ == "__main__":
    model = BinaryLogReg()
    (x_train, y_train), (x_test, y_test) = load_2_7_mnist()

    # set run_for var among "b", "c" and "d"
    run_for = "d"

    lr_ = 0.0
    bs_ = 1

    if run_for == "b":
        lr_ = 0.4
        bs_ = len(x_train)
    elif run_for == "c":
        lr_ = 0.0001
        bs_ = 1
    else:
        lr_ = 0.01
        bs_ = 100

    history = model.train(x_train, y_train, x_test, y_test, learning_rate=lr_, batch_size=bs

    # Plot losses
    plt.plot(history["train_losses"], label="Train")
    plt.plot(history["test_losses"], label="Test")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.legend()
    plt.show()

    # Plot error
    plt.plot(history["train_errors"], label="Train")
    plt.plot(history["test_errors"], label="Test")
    plt.xlabel("Epochs")
    plt.ylabel("Misclassification Error")
    plt.legend()
    plt.show()
```

# Administrative

A8.

a. 20 hours