CSE 546 Homework 2A

Jake Gonzales

November 10, 2023

Collaborators: n/a

1 Conceptual Questions

A1.

(a) To explain why the L1 norm is more likely to result in sparse solutions, I think about the figure we discussed in lecture where we had the level sets of the solution, the ellipses, and observed how they intersected with the L1 and L2 norm shapes. The intersection between the level set and the norm shape is the minimized error function in a feasible region. The intersection of the L1 norm penalty has a component of the diamond/square shape on the axis which results in more sparse solutions (rather than the curved shape on the L2 norm).

(b) The regularizer's upside is that it would promote more sparse solutions because of its "spiky" shape, but the downside is that it is non-convex which mean's the local minimum may not be the global.

(c) **True**: if the step-size is too large the gradient-descent could overshoot the minimum.

(d) Advantage: SGD is more memory-efficient because it takes smaller samples at a time. Disadvantage: It is not as smooth and noise has a higher impact on the stability of SGD than GD.

(e) Logistic regression does not have a closed-form solution due to the nonlinearity of the sigmoid function, therefore, we need to use GD. Linear-regression has a closed-form solution.

2 Convexity and Norms

A2.

(a) We will show that $|a+b| \leq |a| + |b|$ for all $a, b \in \mathbb{R}$:

$$\begin{aligned} a &\leq |a| \\ a+b &\leq |a| + |b| \\ (a+b)^2 &\leq (|a| + |b|)^2 \\ \therefore |a+b| &\leq |a| + |b| \end{aligned}$$

Now lets show that the L1 norm $f(x) = (\sum_{i=1}^{n} |x_i|)$ satisfies these properties:

- Non-negativity: $f(x) \geq 0$ since $|x_i| \forall x \in \mathbb{R}^n$. Also, $|x_i| = 0$ if $x_i = 0$, so $f(x) = 0$ if and only if $x = 0$.
- Absolute scalability: Let $\alpha, x \in \mathbb{R}^n$ then we have $f(a\alpha x) = \sum_{i=1}^{n} |\alpha x_i| = \sum_{i=1}^{n} |\alpha||x_i| = |\alpha| f(x)$ for all α, x.

- Triangle inequality: let $\alpha, \beta \in \mathbb{R}^n$ then $f(\alpha) + f(\beta) = \sum i = 1^n |\alpha_i| + \sum i = 1^n |\beta_i| = \sum i = 1^n |\alpha_i| + |\beta_i| \geq \sum i = 1^n |\alpha_i + \beta_i| = f(\alpha + \beta)$ where we use the definition of the triangle inequality that was previously proven.

Therefore we have shown that $f(x) = (\sum_{i=1}^{n} |x_i|)$ is a norm.

(b) Lets consider two points in $n = 2$ as (0,1) and (1,0). Then evaluating $g(x) = (\sum_{i=1}^{n} |x_i|^{\frac{1}{2}})^2$ at these points we get

$$g(x) + g(y) = 1^2 + 1^2 = 2$$
$$g(x + y) = (1 + 1)^2 = 4$$
$$g(x + y) > g(x) + g(y)$$

therefore the triangle inequality does not hold and $g(x)$ is not a norm.

A3.

(a) For figure **I** we can draw a straight line between points b and c which would exit the set and therefore we say it is not convex.

(b) For figure **II** we can draw a line between d and a and would exit the set therefore it is not convex.

A4.

(a) For the function in panel **I** we have a quadratic function in which we can pick any two points on the interval of [a,c] and draw a line and this line would lie above the graph. The positive quadratic, by the definition of convexity, is a convex function.

(b) For the function in panel **II** we can look at the interval [a,d] and graphically drawing a line between $f(a)$ and $f(d)$ and we can clearly see this line will not lie above the graph, thus breaking the definition of convexity, making this function on the interval [a,d] non-convex. Clearly, we can also see that the interval of [a,c] is a concave function.

3 Lasso on a Real Dataset

A5.

(a) Plot 1 is the number of non-zeros as a function of λ shown below.

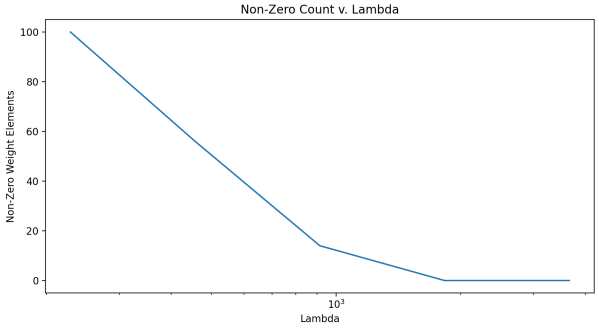

Figure 1: Plot 1 of the number of non-zeros as a function of λ.

(b) Plot 2 has the false discovery rate (FDR) and true positive rate (TPR) for non-zeros in w for each λ tried.

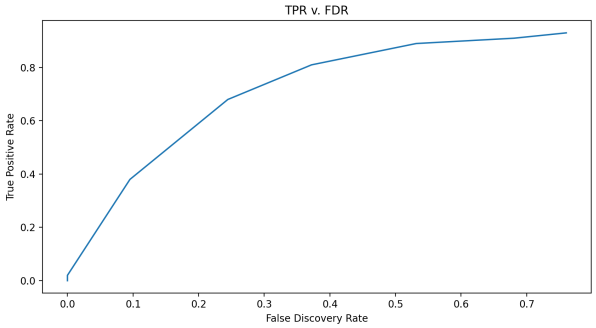

Figure 2: TPR v. FDR.

(c) In Lasso a larger λ means we will have more sparsity, thus more zeros in our weight vector and when we have a smaller λ we will have more non-zero elements so the model uses more features. The second plot clearly shows that if λ is too small we will have a higher FDR and for a overly large λ we have poor model performance. We would desire a λ in the upper left corner of the plot that would lead to sparse solutions with good performance.

```
from typing import Optional, Tuple

import matplotlib.pyplot as plt
import numpy as np

from utils import problem


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

    a = 2*np.sum(X*X, axis=0)

    n, d = X.shape
    b = np.mean(y-np.dot(X, weight))

    for k in range(d):
        x_k = X[:,k]
        weight[k] = 0
        c_k = 2*np.dot(x_k, y - (b + np.dot(X, weight)))
        if c_k < -_lambda:
            weight[k] = (c_k + _lambda) / a[k]
        elif c_k > _lambda:
            weight[k] = (c_k - _lambda)/ a[k]

    return (weight, b)

@problem.tag("hw2-A")
def loss(
    X: np.ndarray, y: np.ndarray, weight: np.ndarray, bias: float, _lambda: float
) -> float:
    #L-1 (Lasso) regularized MSE loss.
    loss_fn = np.linalg.norm(X @ weight + bias - y)**2 + _lambda * np.linalg.norm(weight, 1)

    return loss_fn

@problem.tag("hw2-A", start_line=5)
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]:

    a = 2*np.sum(X*X, axis=0)

    x, d = X.shape


    if start_weight is None:
        start_weight = np.zeros(d)
        start_bias = 0
    old_w = start_weight + np.inf


    while convergence_criterion(start_weight, old_w, None, None, convergence_delta) is False:
        old_w = np.copy(start_weight)
        w = start_weight
        train = step(X, y, w, None, _lambda)

    return train

@problem.tag("hw2-A")
def convergence_criterion(
    weight: np.ndarray, old_w: np.ndarray, bias: float, old_b: float, convergence_delta: float
) -> bool:
    max_absolute_change = np.linalg.norm(weight - old_w, ord=np.inf)

    if max_absolute_change > convergence_delta:
        return False


@problem.tag("hw2-A")
def main():

    sigma = 1
    n = 500
    d = 1000
    k = 100
    iter = 5
    # Gaussian noise for epsilon
    eps_noise = np.random.normal(0, sigma**2, size=n)
    x = np.random.normal(size = (n,d))
    w = np.zeros((d, ))

    for j in range(1, k+1):
        w[j-1] = j/k

    y = x @ w + eps_noise

    lambda_max = np.max(2 * np.sum(x.T * (y - np.mean(y)), axis=0))
    lambda_iter = [lambda_max/(2**i) for i in range (iter)]

    fdr, tpr = [], []
    delta = 1e-5

    # keep nonnzeros count
    non_zeros = []

    for lambda_reg in lambda_iter:
        weight = train(x, y, lambda_reg, delta)[0]
        not_Zeros = np.count_nonzero(weight)
        non_zeros.append(not_Zeros)
        correct_nonZeros  = np.count_nonzero(weight[:k])
        incorrect_nonZeros = np.count_nonzero(weight[k+1:])

        try:
```

```
            fdr.append(incorrect_nonZeros / not_Zeros)
        except ZeroDivisionError:
            fdr.append(0)

        tpr.append(correct_nonZeros/k)


    # first plot
    plt.figure(figsize=(10,5))
    plt.plot(lambda_iter, non_zeros)
    plt.xscale('log')
    plt.title('Non-Zero Count v. Lambda')
    plt.xlabel('Lambda')
    plt.ylabel('Non-Zero Weight Elements')
    plt.show()

    # second plot
    plt.figure(figsize = (10,5))
    plt.plot(fdr, tpr)
    plt.title('TPR v. FDR')
    plt.xlabel('False Discovery Rate')
    plt.ylabel('True Positive Rate')
    plt.show()



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

A6.

(a) The rent prices in certain areas are largely affected by the US housing policies. Police per population relates directly to a police departments budget and resources which is determined by spending laws at the local, state, and federal levels. War on drugs. US federal government sets initiative for drug policies to discourage production, distribution, and consumption of drugs. This led to race issues, mass incarceration (of peoples parents and family members) and other controversial topics all that were directly related to policy decisions made in 1970's still affecting communities today.

(b) Any of the features that are related to requesting officers per some amount of population. The specific features would be LemasTotReqPerPop, PolicReqPerOfficg, LemasTotalReq. There is some crazy statistic that 90% or more of calls to 911 do not involve violent crime. Many people may think if a community has more calls to 911 than another then that means there is more crime there but this is not true. However, the 911 calls can be a result of violent crime.

(c) Figure 3: Plot for the number of nonzero weights of each solution as a function of λ.

(d) Figure 4: Plot for the regularization paths for the coefficients for the input variables that were listed.

(e) Figure 5: Plot for the mean squared error on the training and test data as a function of λ.

(f) The largest of *all* input variable when setting $\lambda = 30$ is PctIlleg, which is the percentage of children born to never married, which had a value of 0.069 as shown in Figure 6. The most negative of *all* the input variables when setting $\lambda = 30$ is Pct2Kids2Par, which is the percentage of children in family housing with two parents, that had a value of -0.69. The high-level outcome of these results show that it is important for children to grow up with two parents in the household which may bring a safe environment for the children to be raised in.

(g) Correlation does not mean causation. if the input variable agePct65up had a large negative weight then there is a high correlation between people over 65 and violent crime. This is probably not

the case, a community could already have high crime rates and old people may move in for certain reasons or move out for other reasons.It does not make initiative sense (nor is it a clever discovery) that there's a relation between old people and violent crime.

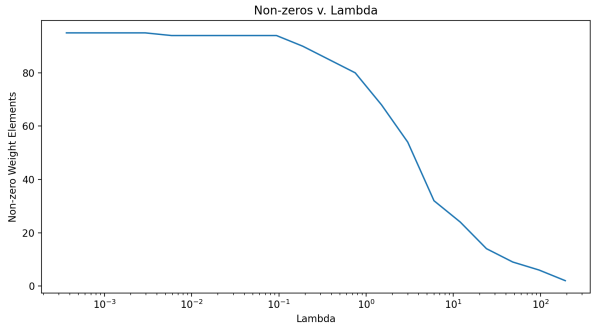

Figure 3: Crime Data: Non-zero Weights as a function of λ

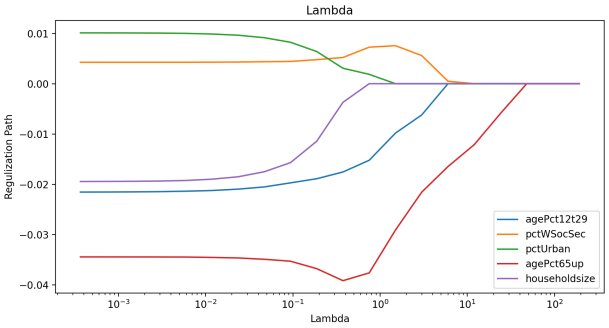

Figure 4: Regularization Paths

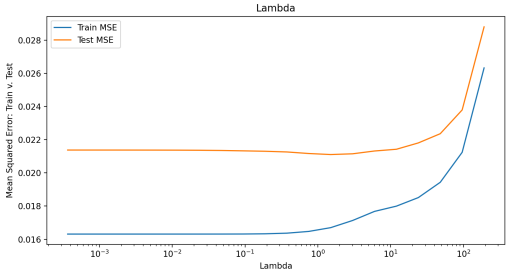

Figure 5: Train v. Test MSE

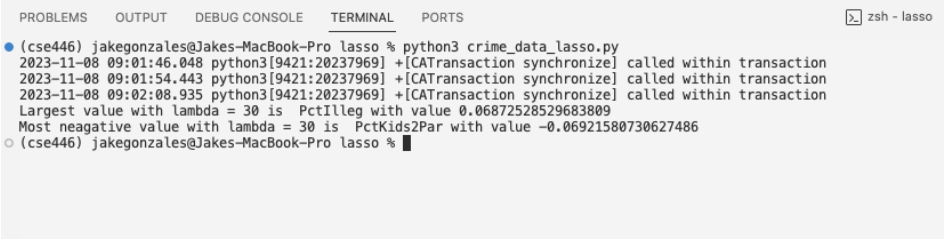

Figure 6: $\lambda = 30$.

```python
if __name__ == "__main__":
    from ISTA import train  # type: ignore
else:
    from .ISTA import train

import matplotlib.pyplot as plt
import numpy as np

from utils import load_dataset, problem

def mse(x, y, weight, bias):

    predict = x.dot(weight) + bias

    return np.mean((predict - y)**2)


@problem.tag("hw2-A", start_line=3)
def main():
    # df_train and df_test are pandas dataframes.
    # Make sure you split them into observations and targets
    df_train, df_test = load_dataset("crime")

    x_df = df_train.drop('ViolentCrimesPerPop', axis=1)
    y_df = df_train['ViolentCrimesPerPop']

    x = np.asarray(df_train.drop('ViolentCrimesPerPop', axis=1))
    y = np.asarray(df_train['ViolentCrimesPerPop'])

    n, d = x.shape

    x_test = np.asarray(df_test.drop('ViolentCrimesPerPop', axis=1))
    y_test = np.asarray(df_test['ViolentCrimesPerPop'])

    lambda_max = np.max(2 * np.sum(x.T * (y - np.mean(y)), axis=0))
    weight = np.zeros((d, ))

    iter = 20
    lambda_iter = [lambda_max/(2**i) for i in range(iter)]
    convergence_delta = 1e-4

    num_zeros = []
    mse_train, mse_test = [], []

    features = ['agePct12t29', 'pctWSocSec', 'pctUrban', 'agePct65up', 'householdsize']
    features_indices = [x_df.columns.get_loc(j) for j in features]

    agePct12t29 = []
    pctWSocSec = []
    pctUrban = []
    agePct65up = []
    householdsize = []
```

```
    for lambda_reg in lambda_iter:
        weight = train(x, y, lambda_reg, convergence_delta, None)[0]
        bias = train(x, y, lambda_reg, convergence_delta, None)[1]

        not_zeros = np.count_nonzero(weight)
        num_zeros.append(not_zeros)

        #y_predict = x.dot(weight) + bias
        #y_test_predict = x_test.dot(weight) + bias
        # compute Mean Squared Error (MSE)
        mse_train_iter = mse(x, y, weight, bias)
        mse_train.append(mse_train_iter)
        mse_test_iter = mse(x_test, y_test, weight, bias)
        mse_test.append(mse_test_iter)

        agePct12t29.append(weight[0])
        pctWSocSec.append(weight[1])
        pctUrban.append(weight[2])
        agePct65up.append(weight[3])
        householdsize.append(weight[4])

    # First plot
    plt.figure(figsize=(10,5))
    plt.plot(lambda_iter, num_zeros)
    plt.xscale('log')
    plt.title('Non-zeros v. Lambda')
    plt.xlabel('Lambda')
    plt.ylabel('Non-zero Weight Elements')
    plt.show()

    # second plot
    plt.figure(figsize=(10,5))
    plt.plot(lambda_iter, agePct12t29, label= 'agePct12t29')
    plt.plot(lambda_iter, pctWSocSec, label= 'pctWSocSec')
    plt.plot(lambda_iter, pctUrban, label= 'pctUrban')
    plt.plot(lambda_iter, agePct65up, label= 'agePct65up' )
    plt.plot(lambda_iter, householdsize, label= 'householdsize')
    plt.xscale('log')
    plt.title('Lambda')
    plt.xlabel('Lambda')
    plt.ylabel('Regulization Path')
    plt.legend()
    plt.show()

    # third plot
    plt.figure(figsize=(10,5))
    plt.plot(lambda_iter, mse_train, label="Train MSE")
    plt.plot(lambda_iter, mse_test, label='Test MSE')
    plt.xscale('log')
    plt.title('Lambda')
    plt.xlabel('Lambda')
    plt.ylabel('Mean Squared Error: Train v. Test')
    plt.legend()
    plt.show()

    # Question F
    new_lambda = 30
    weight = train(x, y, new_lambda, 1e-4, None)[0]

    print('Largest value with lambda = 30 is ', x_df.columns[np.argmax(weight)], 'with value', np.max(weight) )
    print('Most neagative value with lambda = 30 is ', x_df.columns[np.argmin(weight)], 'with value', np.min(weight) )


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

4 Logistic Regression

A7.

(a) Let's derive $\nabla_w J(w,b)$:

$$\begin{aligned}\nabla_w J(w,b) &= \nabla_w \left[\frac{1}{n}\sum_{i=1}^{n}\log(1+\exp(-y_i(b+x_i^T w))) + \lambda\|w\|_2^2\right] \\ &= \frac{1}{n}\sum_{i=1}^{n}\frac{-y_i x_i^T \exp(-y_i(b+x_i^T w))}{1+\exp(-y_i(b+x_i^T w))} + 2\lambda w \\ &= \frac{1}{n}\sum_{i=1}^{n} -y_i x_i^T (1-\mu_i(w,b)) + 2\lambda w\end{aligned}$$

Then to derive $\nabla_b J(w,b)$:

$$\begin{aligned}\nabla_b J(w,b) &= \nabla_b \left[\frac{1}{n}\sum_{i=1}^{n}\log(1+\exp(-y_i(b+x_i^T w))) + \lambda\|w\|_2^2\right] \\ &= \frac{1}{n}\sum_{i=1}^{n}\frac{-y_i \exp(-y_i(b+x_i^T w))}{1+\exp(-y_i(b+x_i^T w))} \\ &= \frac{1}{n}\sum_{i=1}^{n} -y_i (1-\mu_i(w,b))\end{aligned}$$

(b) Figure 7 is a subplot that has the losses for train and test sets as well as the missclassification error for train and test set with gradient descent implemented.

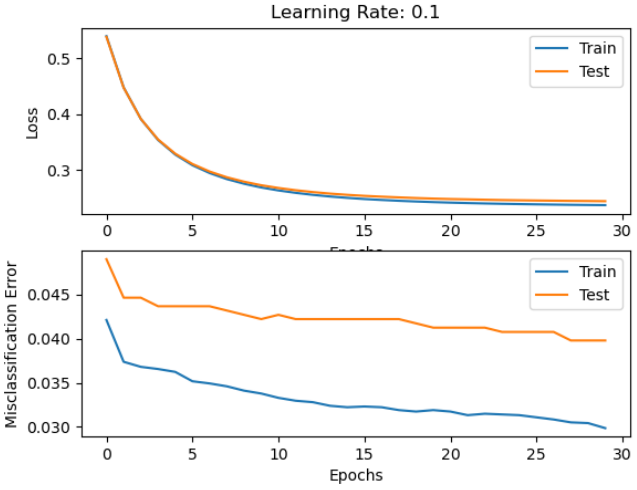

Figure 7: Gradient Descent: Losses and Missclassification Error

(c) Figure 8 is a subplot of repeating part B using stochastic gradient descent (SGD) with a batch size of 1.

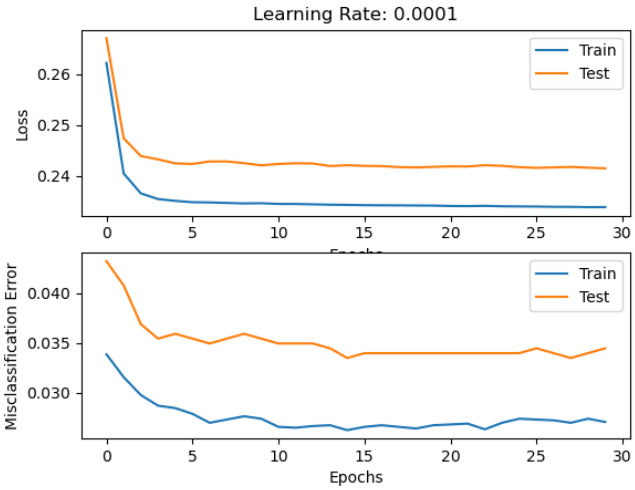

Figure 8: Stochastic Gradient Descent with batch size 1: Losses and Missclassification Error

(d) Figure 9 is a subplot of repeating part B using stochastic gradient descent (SGD) with a batch size of 100.

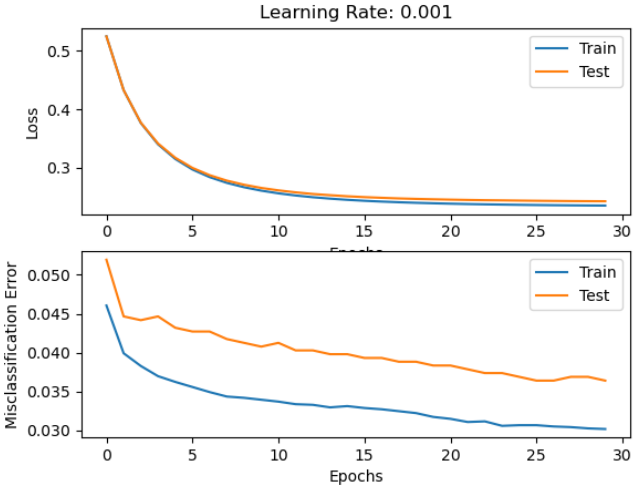

Figure 9: Stochastic Gradient Descent with batch size 100: Losses and Missclassification Error

```python
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 not `np.random.choice` etc.
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 = None  # type: ignore
        self.bias: float = 0.0
        # raise NotImplementedError("Your Code Goes Here")

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

        mu = 1/(1 + np.exp(-y * (self.bias + np.matmul(X, self.weight))))

        return mu

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

        n = y.size
        J = (1/n) * np.sum(np.log(1 + np.exp(-y*(self.bias + np.matmul(X, self.weight))))) + self._lambda * np.sum(self

        return J


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

```
        n = y.size
        mu_vec = self.mu(X, y)

        J_weight = (1/n) * np.matmul(np.multiply((mu_vec-1), y), X) + (2 * self._lambda * self.weight)

        return J_weight


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

        n = y.size
        mu_i = self.mu(X, y)
        J_bias = (1/n) * (np.dot(mu_i, y) - np.sum(y))

        return J_bias

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

        predict = np.sign(self.bias + np.matmul(X, self.weight))

        return predict

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

        n = y.size
        missclassified = np.where(y != self.predict(X))[0].size
        missclassified = missclassified/n

        return missclassified

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

        self.weight = self.weight - learning_rate * self.gradient_J_weight(X, y)
        self.bias = self.bias - learning_rate * self.gradient_J_bias(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": [],
        }
        n, d = X_train.shape
        self.weight = np.zeros(d)
        for epoch in range(epochs):
            for batch in range(num_batches):
                batch_i = RNG.choice(np.arange(n), size=batch_size)
                self.step(X_train[batch_i, :], y_train[batch_i], learning_rate=learning_rate)

            # Compute the losses and errors
            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

def plot(history, learning_rate):

    # plot function for errors and losses

    # losses
    fig, (ax1, ax2) = plt.subplots(nrows=2)
    ax1.plot(history["train_losses"], label="Train")
    ax1.plot(history["test_losses"], label="Test")
    ax1.set_xlabel("Epochs")
    ax1.set_ylabel("Loss")
    ax1.set_title(f'Learning Rate: {learning_rate}')
    ax1.legend()

    # error
    ax2.plot(history["train_errors"], label="Train")
    ax2.plot(history["test_errors"], label="Test")
    ax2.set_xlabel("Epochs")
    ax2.set_ylabel("Misclassification Error")
    ax2.legend()


    plt.show()

if __name__ == "__main__":
    model = BinaryLogReg(_lambda=0.1)
    (x_train, y_train), (x_test, y_test) = load_2_7_mnist()

    # GD
    GD_lr = 0.1
    history_gd = model.train(x_train, y_train, x_test, y_test, learning_rate=GD_lr , epochs=30, batch_size=x_train.shap
    plot(history_gd, learning_rate=GD_lr )

    # SGD Batch Size 1
    SGD_lr_b1 = 0.0001
    history_sgd_batch1 = model.train(x_train, y_train, x_test, y_test, learning_rate=SGD_lr_b1 , epochs=30, batch_size=
    plot(history_sgd_batch1, learning_rate=SGD_lr_b1 )

     # SGD Batch Size 100
```

```
SGD_lr_b100 = 0.001
history_sgd_batch100 = model.train(x_train, y_train, x_test, y_test, learning_rate=SGD_lr_b100, epochs=30, batch_si
plot(history_sgd_batch100, learning_rate=SGD_lr_b100)
```

A8.

(a) I spent like 20 hours on this homework.