CSE546 - Homework # 2 - Solutions

Cheng-Yen Yang

May 7, 2021

A.0

A.0.a

If the feature is an independent feature totally uncorrelated with other feature, then the new predictions will be strictly worse than before. However, in this case we are predicting house prices which the the feature 'number of bathrooms' will probably be related to other features e.g. 'number of bedrooms', then the new predictions may not be affected due to the other perfectly correlated features.

A.0.b

L1 and L2 norm add constraints to the optimization problem, where the blue regions are the constraints. The optimized solution is the point where the $\hat{\beta}$ meets the constraints. L1 norm is more likely to produce an intersection that has one component of the solution is zero (i.e. the sparse model) due to the the geometric properties of ellipses, disks, and diamonds. (Ref: https://web.stanford.edu/ hastie/Papers/ESLII.pdf)

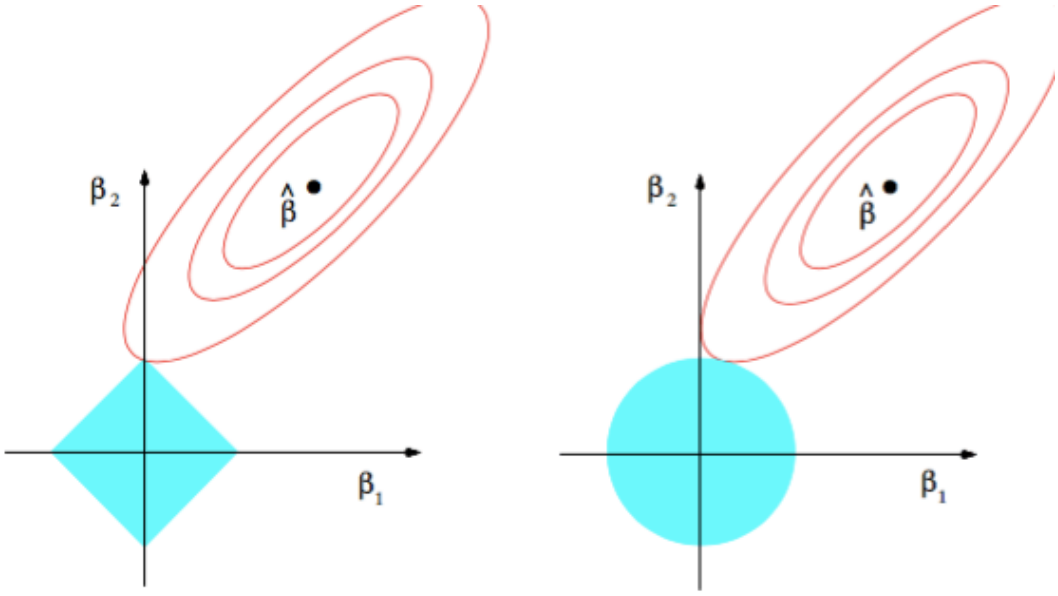

Figure 1: Estimation Picture of L1 and L2 Norm

A.0.c

Regularizer: $\left(\sum_i |w_i|^{0.5}\right)$ will gives an even sparser solution than L1 but it is non-convex.

A.0.d

True. If the step-size for gradient descent is too large, it may not converge.

A.0.e

Instead of calculating the cost of all data points, SGD calculates the cost of one single data point and the corresponding gradient, and then update the weights. If the true function is relatively monotonic functions, SGD works by analyzing only one example at a time and following its slope can reach a point that is very close to the actual minimum.

A.0.f

In the case of SGD, the computation is faster than GD as we take one data point at a time. But it takes much more time to converge. On the other hand, GD takes fewer steps to converge to the minimum than SGD.

A.1

We first begin by showing that $|a+b| \leq |a| + |b|$ for all $a, b \in \mathbb{R}$:

$$|a+b|^2 = a^2 + 2ab + b^2 \leq |a|^2 + 2|a||b| + |b|^2 = (|a| + |b|)^2 \Leftrightarrow |a+b| \leq |a| + |b|. \tag{1}$$

A.1.a

To show that $f(x) = (\sum_{i=1}^{n} |x_i|)$ is a norm, we will checking if $f(x)$ holds the by the properties: non-negativity, absolute scalability and triangle inequality:

- Non-negativity: Since $|x_i| \geq 0$ for all i, therefore $f(x) = (\sum_{i=1}^{n} |x_i|) \geq 0$ for all $x \in \mathbb{R}^n$ with $f(x) = 0$ if and only if $x = 0$.
- Absolute scalability: Since $f(ax) = (\sum_{i=1}^{n} |ax_i|) = |a| (\sum_{i=1}^{n} |ax_i|) = |a| f(x)$ for all $a \in \mathbb{R}$ and $x \in \mathbb{R}^n$.
- Triangle inequality: Since $f(x)+f(y) = (\sum_{i=1}^{n} |x_i|)+(\sum_{i=1}^{n} |y_i|) = (\sum_{i=1}^{n} |x_i| + |y_i|) \geq (\sum_{i=1}^{n} |x_i + y_i|) = f(x+y)$ for all $x, y \in \mathbb{R}^n$.

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

A.1.b

To show that $g(x) = \left(\sum_{i=1}^{n} |x_i|^{1/2}\right)^2$ is a norm, we will checking if $g(x)$ holds the by the properties: non-negativity, absolute scalability and triangle inequality:

- Triangle inequality: Consider two arbitrary points $x = [0, k]^T$ and $y = [k, 0]^T$ where $k > 0$, then we have $g(x) + g(y) = 2k < g(x+y) = 4k$ which does not follow triangle inequality $\|x+y\| \leq \|x\| + \|y\|$ for all $x, y \in \mathbb{R}^n$ when $n = 2$.

Therefore $g(x) = \left(\sum_{i=1}^{n} |x_i|^{1/2}\right)^2$ **is not a norm**.

A.2

A set $A \subseteq \mathbb{R}^n$ is *convex* if $\lambda x + (1-\lambda)y \in A$ for all $x, y \in A$ and $\lambda \in [0,1]$. So for the grey-shaded sets I, II and III to be convex, all of the line segments between any of the two points must be within the sets as $\lambda \in [0,1]$.

- **I is not convex**, $\lambda x + (1-\lambda)y \in A$ does not always holds when $(x, y) = (b, c)$.
- **II is convex**.
- **III is not convex**, $\lambda x + (1-\lambda)y \in A$ does not always holds when $(x, y) = (a, d)$.

A.3

A function $f : \mathbb{R}^d \to \mathbb{R}$ is convex on a set A if $f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y)$ for all $x, y \in A$ and $\lambda \in [0,1]$.

- **I is convex** on $[a, c]$.
- **II is not convex** on $[a, c]$, $f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y)$ does not hold when $(x, y) = (b, c)$.
- **III is not convex** on $[a, d]$, $f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y)$ does not hold when $(x, y) = (a, c)$.
- **III is convex** on $[c, d]$.

A.4

A.4.a

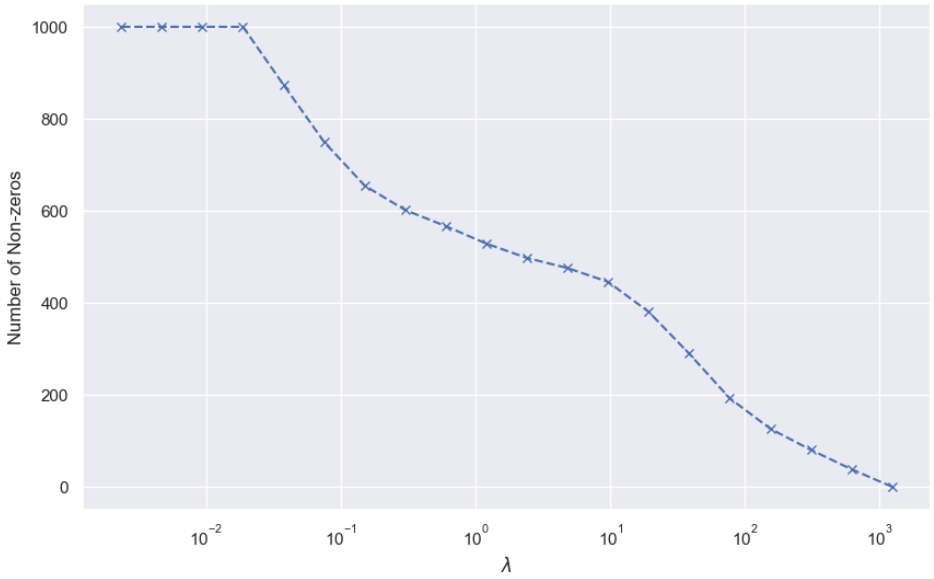

Figure 2: Number of non-zeros in $\widehat{w}_j$ with Lasso using different λ values ($\lambda_{max} = 1234.85$).

A.4.b

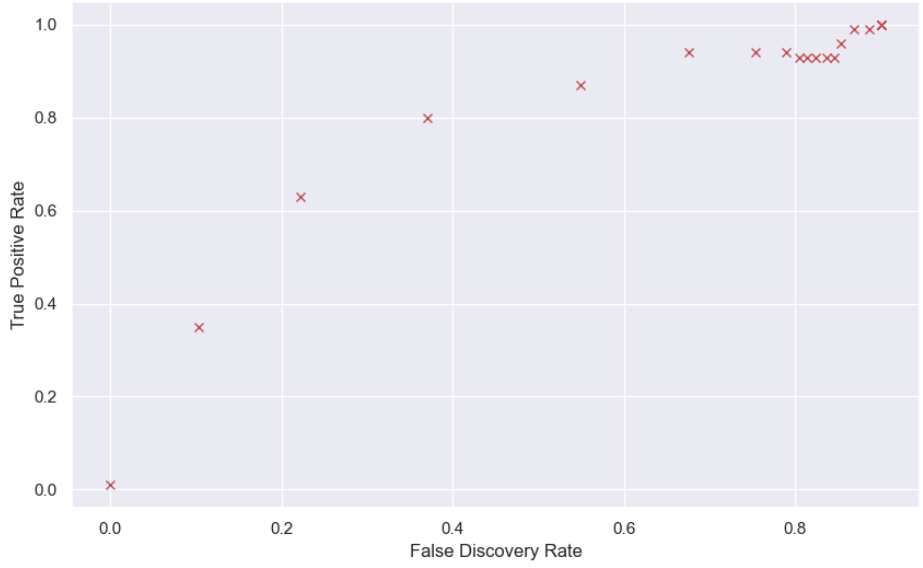

Figure 3: Plot of false discovery rate versus true positive rate with Lasso using different λ values.

A.4.c

From A.4.a, we can observe that larger λ will make more weights in $\widehat{w}_j$ being zeros while smaller λ will force the model to use all features by setting weights in $\widehat{w}_j$ to non-zeros. From A.4.b, we can further observe that by the name we are decreasing the λ, the TPR will stop to increase at a certain point while the FDR increase at a much faster rate (Note that in an ideal situation we would have an (FDR,TPR) pair in the upper left corner). Therefore combining the two observations, the λ value need to be carefully chosen in order to preserve a sparse yet accurate model using Lasso.

A.4.code (Lasso)

```
import numpy as np


class Lasso:
    def __init__(self, X, y, reg_lambda):
        self.reg_lambda = reg_lambda
        self.X = X
        self.y = y
        self.w = np.zeros(X.shape[1])
        self.b = 0
        self.delta = 1e-3
        self.global_w_history = {}
        self.local_w_history = []
        self.cur_iter = 0

    def reset(self, reg_lambda):
        self.reg_lambda = reg_lambda
        self.w = np.zeros(self.X.shape[1])
        self.b = 0
        self.local_w_history = []
        self.cur_iter = 0

    def fit(self):
        a = 2 * np.sum(self.X**2, axis=0)

        while self.cur_iter == 0 or np.linalg.norm(self.w - self.local_w_history
            [-1]) > self.delta:
            self.cur_iter += 1
            self.local_w_history.append(np.copy(self.w))
            # print('Iteration {}'.format(self.cur_iter))
            self.b = np.mean(self.y - self.X.dot(self.w))
            for k in range(self.X.shape[1]):
                a_k = a[k]
                c_k = 2 * np.sum(self.X[:,k] * (self.y - (self.b + self.X.dot(
                    self.w) - self.X[:,k].dot(self.w[k]))), axis=0)
                if c_k < -self.reg_lambda:
                    self.w[k] = (c_k + self.reg_lambda) / a_k
                elif c_k > self.reg_lambda:
                    self.w[k] = (c_k - self.reg_lambda) / a_k
                else:
                    self.w[k] = 0

        self.global_w_history[self.reg_lambda] = self.w
```

```
    def loss(self):
        return (np.linalg.norm(self.X.dot(self.w) + self.b - self.y))**2 + \
                    self.reg_lambda * np.linalg.norm(self.w)

    def predict(self, X_pred):
        return X_pred.dot(self.w) + self.b
```

A.4.code

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

from lasso import Lasso

sns.set()


def count(w_true, w_train, k):
    nonzero_cnt = np.sum(abs(w_train) > 0)
    fdr = np.sum(abs(w_train[k:]) > 0) / nonzero_cnt
    tpr = np.sum(abs(w_train[:k]) > 0) / k
    return nonzero_cnt, fdr, tpr

def main():
    np.random.seed(1968990)

    n, d, k, sigma = 500, 1000, 100, 1
    m = 20
    w = np.zeros((d, ))
    for j in range(1, k+1):
        w[j-1] = j/k
    X = np.random.normal(size=(n, d))
    y = X.dot(w) + np.random.normal(size=(n,))
    # print(X.shape, y.shape, w.shape)

    reg_lambda = max(2*np.sum(X*(y-np.mean(y))[:, None], axis=0))
    # reg_lambda = max(2*np.sum(X.T*(y-np.mean(y)), axis=0))
    print('Max lambda: ', reg_lambda)

    model = Lasso(X, y, reg_lambda)

    lambdas = []
    nonzeros = []
    fdrs = []
    tprs = []
    for _ in range(m):
        model.fit()
        lambdas.append(reg_lambda)

        nz, fdr, tpr = count(w, model.w, k)
        nonzeros.append(nz)
        fdrs.append(fdr)
```

```
        tprs.append(tpr)

        reg_lambda /= 2
        model.reset(reg_lambda)

    plt.figure(figsize=(10,6))
    plt.plot(lambdas, nonzeros, '--x')
    plt.xscale('log')
    plt.xlabel('$\lambda$')
    plt.ylabel('Number of Non-zeros')
    plt.savefig('A4a.png')
    plt.tight_layout()

    plt.figure(figsize=(10,6))
    plt.plot(fdrs, tprs, 'rx')
    plt.xlabel('False Discovery Rate')
    plt.ylabel('True Positive Rate')
    plt.savefig('A4b.png')
    plt.tight_layout()

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

A.5

A.5.a

- **(OfficAssgnDrugUnits)**: number of officers assigned to special drug units (numeric - decimal)
- **(NumKindsDrugsSeiz)**: number of different kinds of drugs seized (numeric - decimal)
- **(LemasPctOfficDrugUn)**: percent of officers assigned to drug units (numeric - decimal)

A.5.b

After reading the documentation for the dataset, we think that the features relating to **requests** will probably be found to have non-zero weights. However, the numbers of request should be consider as the result rather than the reason of the crime rate.

- **(LemasTotReqPerPop)**: total requests for police per 100K population (numeric - decimal)
- **(LemasTotalReq)**: total requests for police (numeric - decimal)
- **(PolicReqPerOfficg)**: total requests for police per police officer (numeric - decimal)

A.5.c

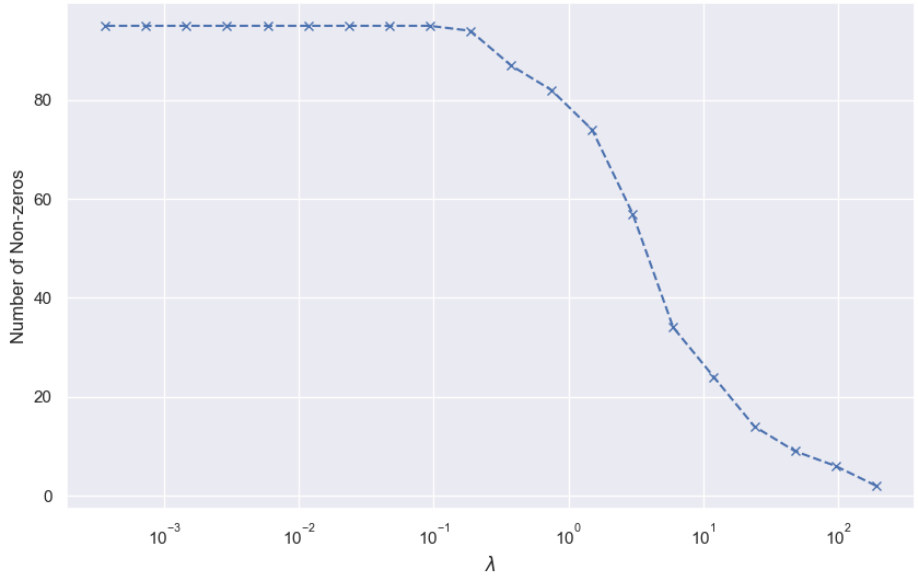

Figure 4: Number of non-zeros in $\widehat{w}_j$ with Lasso using different λ values ($\lambda_{max} = 191.98$).

A.5.d

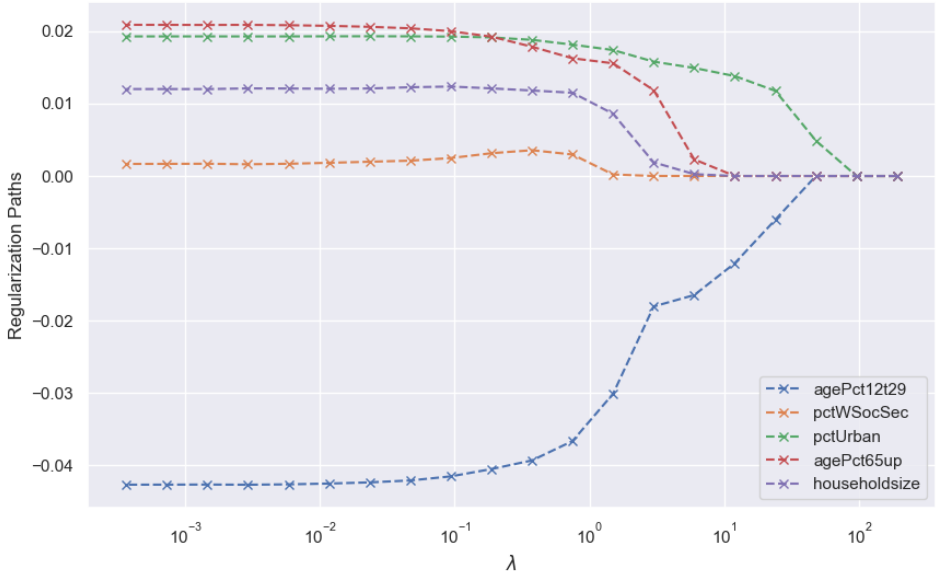

Figure 5: Regularization paths for various coefficients with Lasso using different λ values.

A.5.e

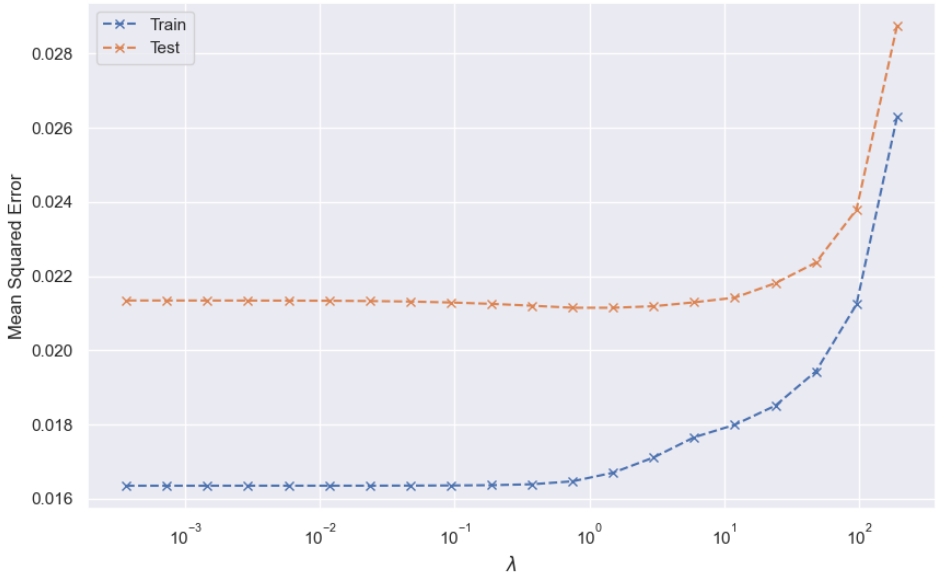

Figure 6: Plot of training and testing MSE with Lasso using different λ values.

A.5.f

- **(Largest Lasso Coefficient):** PctIlleg ($\lambda_{PctIlleg} = 0.068$).
- **(Smallest Lasso Coefficient):** PctKids2Par ($\lambda_{PctKids2Par} = -0.070$).

A.5.g

A large negative weight only suggest high negative correlation between the two but not the consequence. For example, as the problem stated, `agePct65up`, the higher percentage of population that is 65 and over in age may not be the true reason for lower crime rate. Instead, it may be cause by the fact that people who are 65 and over in age are more likely to move out of the areas which have the higher crime rate and move in to the areas with the lower crime rate.

A.5.code

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns

from lasso import Lasso

sns.set()


def count(w_train):
    nonzero_cnt = np.sum(abs(w_train) > 0)
    return nonzero_cnt

def main():
    m = 20
    df_train = pd.read_table("crime-train.txt")
    df_test = pd.read_table("crime-test.txt")

    X = df_train.drop('ViolentCrimesPerPop', axis=1)
    y = df_train['ViolentCrimesPerPop']
    # print(X.shape, y.shape)

    X_test = df_test.drop('ViolentCrimesPerPop', axis=1)
    y_test = df_test['ViolentCrimesPerPop']

    reg_lambda = max(2 * np.sum(X.T * (y - np.mean(y)), axis=0))
    print('Max lambda: ', reg_lambda)

    model = Lasso(X.values, y.values, reg_lambda)

    lambdas = []
    nonzeros = []
    mse_train = []
    mse_test = []

    for _ in range(m):
        model.fit()
        lambdas.append(reg_lambda)
        nonzeros.append(np.sum(abs(model.w) > 0))
```

```
        y_train_preds = model.predict(X.values)
        y_test_preds = model.predict(X_test.values)

        mse_train.append(np.mean((y.values-y_train_preds)**2))
        mse_test.append(np.mean((y_test.values-y_test_preds)**2))

        reg_lambda /= 2
        model.reset(reg_lambda)

    plt.figure(figsize=(10, 6))
    plt.plot(lambdas, nonzeros, '--x')
    plt.xscale('log')
    plt.xlabel('$\lambda$')
    plt.ylabel('Number of Non-zeros')
    plt.savefig('A5c.png')
    plt.tight_layout()

    plt.figure(figsize=(10, 6))
    vis_feat_name = ['agePct12t29', 'pctWSocSec', 'pctUrban', 'agePct65up', '
        householdsize']
    for name in vis_feat_name:
        idx = X.columns.get_loc(name)
        w_path = []
        for key, val in model.global_w_history.items():
            w_path.append(val[idx])
        plt.plot(lambdas, w_path, '--x', label=name)
    plt.xscale('log')
    plt.xlabel('$\lambda$')
    plt.ylabel('Regularization Paths')
    plt.legend()
    plt.savefig('A5d.png')
    plt.tight_layout()

    plt.figure(figsize=(10, 6))
    plt.plot(lambdas, mse_train, '--x', label='Train')
    plt.plot(lambdas, mse_test, '--x', label='Test')
    plt.xscale('log')
    plt.xlabel('$\lambda$')
    plt.ylabel('Mean Squared Error')
    plt.legend()
    plt.savefig('A5e.png')
    plt.tight_layout()

    model.reset(30)
    model.fit()
    print('Largest Lasso coefficient: ', X.columns[np.argmax(model.w)], max(
        model.w))
    print('Smallest Lasso coefficient: ', X.columns[np.argmin(model.w)], min(
        model.w))


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

A.6

A.6.a

First we derive $\nabla_w J(w,b)$:

$$\nabla_w J(w,b) = \nabla_w \left(\frac{1}{n} \sum_{i=1}^{n} \log(1+\exp(-y_i(b+x_i^T w))) + \lambda ||w||_2^2 \right) \tag{2}$$

$$= \frac{1}{n} \sum_{i=1}^{n} \nabla_w \left(\log(1+\exp(-y_i(b+x_i^T w))) \right) + \nabla_w \lambda ||w||_2^2 \tag{3}$$

$$= \frac{1}{n} \sum_{i=1}^{n} \nabla_w \left(-\log(\mu_i(w,b)) \right) + \nabla_w \lambda ||w||_2^2 \tag{4}$$

$$= -\frac{1}{n} \sum_{i=1}^{n} \frac{\nabla_w \mu_i(w,b)}{\mu_i(w,b)} + 2\lambda w \tag{5}$$

$$= -\frac{1}{n} \sum_{i=1}^{n} \nabla_w \left(\frac{1}{1+\exp(-y_i(b+x_i^T w))} \right) \frac{1}{\mu_i(w,b)} + 2\lambda w \tag{6}$$

$$= -\frac{1}{n} \sum_{i=1}^{n} \left(\frac{-y_i x_i \exp(-y_i(b+x_i^T w))}{(1+\exp(-y_i(b+x_i^T w)))^2} \right) \frac{1}{\mu_i(w,b)} + 2\lambda w \tag{7}$$

$$= -\frac{1}{n} \sum_{i=1}^{n} \left(\frac{-y_i x_i \left(\frac{1-\mu_i(w,b)}{\mu_i(w,b)} \right)}{\mu_i(w,b)^2} \right) \frac{1}{\mu_i(w,b)} + 2\lambda w = -\frac{1}{n} \sum_{i=1}^{n} \left(-y_i x_i (1-\mu_i(w,b)) \right) + 2\lambda w \tag{8}$$

where we use $\mu_i(w,b) = \frac{1}{1+\exp(-y_i(b+x_i^T w))}$ and $\exp(-y_i(b+x_i^T w)) = \frac{1-\mu_i(w,b)}{\mu_i(w,b)}$ to substitute out the exponential terms in the steps.

Then we derive $\nabla_b J(w,b)$ using the similar logistic:

$$\nabla_b J(w,b) = \nabla_b \left(\frac{1}{n} \sum_{i=1}^{n} \log(1+\exp(-y_i(b+x_i^T w))) + \lambda ||w||_2^2 \right) \tag{9}$$

$$= \frac{1}{n} \sum_{i=1}^{n} \nabla_b \left(\log(1+\exp(-y_i(b+x_i^T w))) \right) \tag{10}$$

$$= \frac{1}{n} \sum_{i=1}^{n} \nabla_b \left(-\log(\mu_i(w,b)) \right) \tag{11}$$

$$= -\frac{1}{n} \sum_{i=1}^{n} \frac{\nabla_b \mu_i(w,b)}{\mu_i(w,b)} \tag{12}$$

$$= -\frac{1}{n} \sum_{i=1}^{n} \nabla_b \left(\frac{1}{1+\exp(-y_i(b+x_i^T w))} \right) \frac{1}{\mu_i(w,b)} \tag{13}$$

$$= -\frac{1}{n} \sum_{i=1}^{n} \left(\frac{-y_i \exp(-y_i(b+x_i^T w))}{(1+\exp(-y_i(b+x_i^T w)))^2} \right) \frac{1}{\mu_i(w,b)} \tag{14}$$

$$= -\frac{1}{n} \sum_{i=1}^{n} \left(\frac{-y_i \left(\frac{1-\mu_i(w,b)}{\mu_i(w,b)} \right)}{\mu_i(w,b)^2} \right) \frac{1}{\mu_i(w,b)} = -\frac{1}{n} \sum_{i=1}^{n} \left(-y_i (1-\mu_i(w,b)) \right) \tag{15}$$

where we use $\mu_i(w,b) = \frac{1}{1+\exp(-y_i(b+x_i^T w))}$ and $\exp(-y_i(b+x_i^T w)) = \frac{1-\mu_i(w,b)}{\mu_i(w,b)}$ to substitute out the exponential terms in the steps.

A.6.b

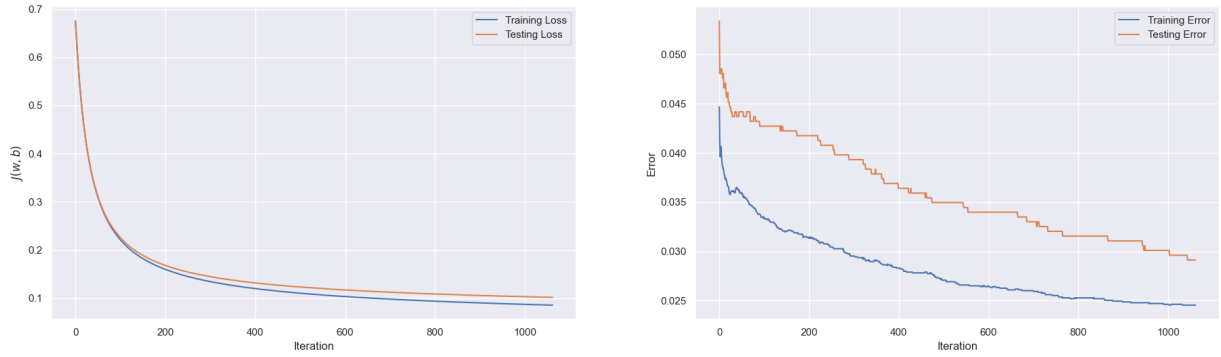

Figure 7: Plot of training and testing loss and error using gradient descent over iteration.

A.6.c

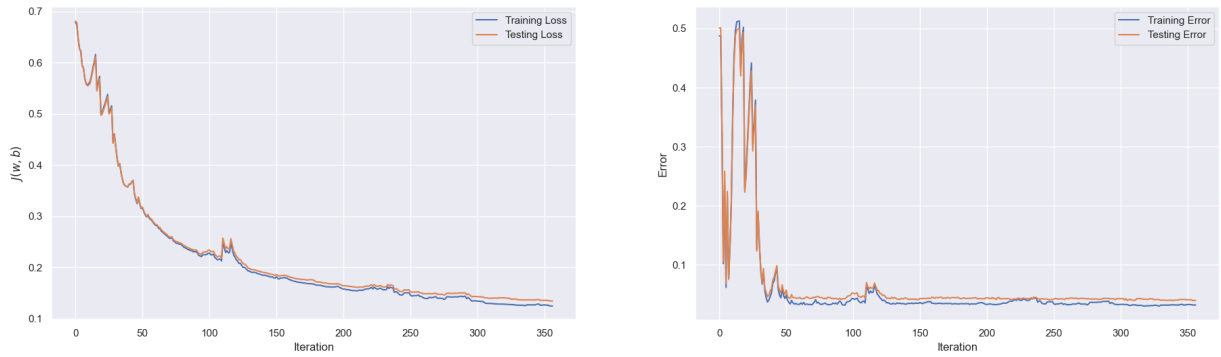

Figure 8: Plot of training and testing loss and error using SGD (batch size 1) over iteration.

A.6.d

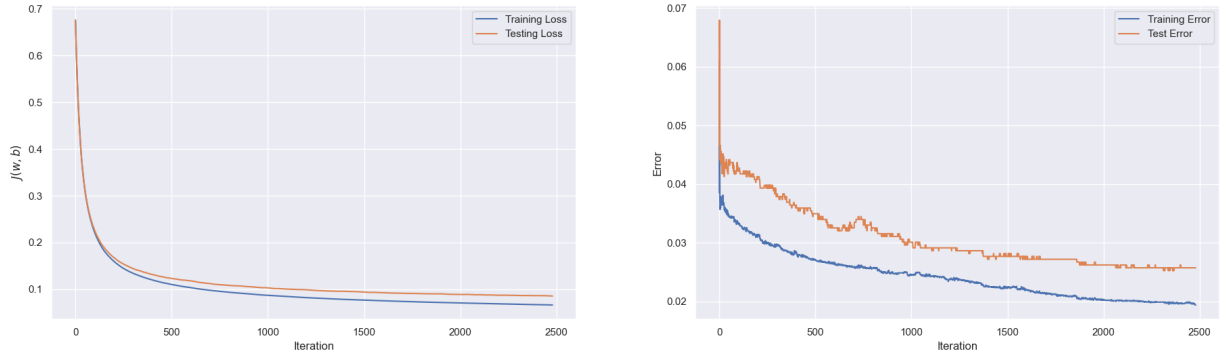

Figure 9: Plot of training and testing loss and error using SGD (batch size 100) over iteration.

A.6.code

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from mnist import MNIST

sns.set()

np.random.seed(1968990)


def load_dataset():
    mndata = MNIST('./data/')
    X_train, Y_train = map(np.array, mndata.load_training())
    X_test, Y_test = map(np.array, mndata.load_testing())
    X_train = X_train / 255.0
    X_test = X_test / 255.0
    return X_train, Y_train, X_test, Y_test


class LogisticRegression:
    def __init__(self, X, Y, X_test, Y_test, reg_lambda=1e-16, step=1e-2):
        self.X = X
        self.Y = Y
        self.X_test = X_test
        self.Y_test = Y_test
        self.reg_lambda = reg_lambda
        self.step = step
        self.w = np.zeros(X.shape[1])
        self.b = 0
        self.delta = 5e-4
        self.global_w_history = {}
        self.local_w_history = []
        self.local_b_history = []
        self.local_J_history = []
        self.local_J_test_history = []
        self.training_err = []
        self.test_err = []
        self.cur_iter = 0

    def grad_w(self, X, Y):
        return np.mean((((1/(1 + np.exp(-Y*(self.b + X.dot(self.w))))) - 1) * Y)
            [:, None] * X, axis=0) + \
                2 * self.reg_lambda * self.w

    def grad_b(self, X, Y):
        return np.mean(((1/(1 + np.exp(-Y*(self.b + X.dot(self.w))))) - 1) * Y,
            axis=0)

    def J(self, X, Y):
        return np.mean(np.log(1 + np.exp(-Y * (self.b + X.dot(self.w))))) + \
                self.reg_lambda * self.w.dot(self.w)

    def J_test(self, X, Y):
```

```
        return np.mean(np.log(1 + np.exp(-self.Y_test * (self.b + self.X_test.
            dot(self.w))))) + \
                self.reg_lambda * self.w.dot(self.w)

    def fit(self, batchsize):
        while self.cur_iter == 0 \
                or np.linalg.norm(self.w - self.local_w_history[-1]) > self.
                    delta \
                and self.cur_iter < 10000: # max iteration

            batch = np.random.choice(self.X.shape[0], batchsize)
            X = self.X[batch]
            Y = self.Y[batch]

            self.cur_iter += 1
            self.local_w_history.append(np.copy(self.w))
            self.local_b_history.append(self.b)

            self.w -= self.step * self.grad_w(X, Y)
            self.b -= self.step * self.grad_b(X, Y)

            self.local_J_history.append(self.J(self.X, self.Y))
            self.local_J_test_history.append(self.J_test(self.X_test, self.
                Y_test))

            self.predict()

            if self.cur_iter % 10 == 0:
                print('Iter {}    | Training Loss: {:.5f} Training Acc: {:.2f} |
                    '
                      'Training Loss: {:.5f} Training Acc: {:.2f} |'.format(
                        self.cur_iter,
                        self.local_J_history[-1],
                        100 * (1 - self.training_err[-1]),
                        self.local_J_test_history[-1],
                        100 * (1 - self.test_err[-1]))
                )

        self.local_w_history.append(np.copy(self.w))
        self.local_b_history.append(self.b)
        self.local_J_history.append(self.J(self.X, self.Y))
        self.local_J_test_history.append(self.J_test(self.X_test, self.Y_test))
        self.predict()

    def predict(self):
        Y_train_pred = np.sign(self.b + self.X.dot(self.w))
        Y_test_pred = np.sign(self.b + self.X_test.dot(self.w))
        self.training_err.append(1 - np.mean(self.Y == Y_train_pred))
        self.test_err.append(1 - np.mean(self.Y_test == Y_test_pred))


def main():
    print("Loading MNIST dataset from source")
    X_train, Y_train, X_test, Y_test = load_dataset()
    print(X_train.shape, X_test.shape)
```

```
X_train_binary, Y_train_binary = [], []
X_test_binary, Y_test_binary = [], []
for data, label in zip(X_train, Y_train):
    if label == 7:
        X_train_binary.append(data)
        Y_train_binary.append(1)
    elif label == 2:
        X_train_binary.append(data)
        Y_train_binary.append(-1)
for data, label in zip(X_test, Y_test):
    if label == 7:
        X_test_binary.append(data)
        Y_test_binary.append(1)
    elif label == 2:
        X_test_binary.append(data)
        Y_test_binary.append(-1)
X_train_binary = np.asarray(X_train_binary)
Y_train_binary = np.asarray(Y_train_binary)
X_test_binary = np.asarray(X_test_binary)
Y_test_binary = np.asarray(Y_test_binary)
print(X_train_binary.shape)
print(X_test_binary.shape)
print("Finished loading binary MNIST dataset from source")

model = LogisticRegression(X_train_binary, Y_train_binary, X_test_binary,
    Y_test_binary)
model.fit(X_train_binary.shape[0])

plt.figure(figsize=(10, 6))
plt.plot(model.local_J_history, label='Training Loss')
plt.plot(model.local_J_test_history, label='Testing Loss')
plt.xlabel('Iteration')
plt.ylabel('$J(w,b)$')
plt.legend()
plt.savefig('A6b1.png')

plt.figure(figsize=(10, 6))
plt.plot(model.training_err, label='Training Error')
plt.plot(model.test_err, label='Testing Error')
plt.xlabel('Iteration')
plt.ylabel('Error')
plt.legend()
plt.savefig('A6b2.png')

model = LogisticRegression(X_train_binary, Y_train_binary, X_test_binary,
    Y_test_binary)
model.fit(1)

plt.figure(figsize=(10, 6))
plt.plot(model.local_J_history, label='Training Loss')
plt.plot(model.local_J_test_history, label='Testing Loss')
plt.xlabel('Iteration')
plt.ylabel('$J(w,b)$')
plt.legend()
plt.savefig('A6c1.png')
```

```
    plt.figure(figsize=(10, 6))
    plt.plot(model.training_err, label='Training Error')
    plt.plot(model.test_err, label='Testing Error')
    plt.xlabel('Iteration')
    plt.ylabel('Error')
    plt.legend()
    plt.savefig('A6c2.png')

    model = LogisticRegression(X_train_binary, Y_train_binary, X_test_binary,
        Y_test_binary)
    model.fit(100)

    plt.figure(figsize=(10, 6))
    plt.plot(model.local_J_history,  label='Training Loss')
    plt.plot(model.local_J_test_history, label='Testing Loss')
    plt.xlabel('Iteration')
    plt.ylabel('$J(w,b)$')
    plt.legend()
    plt.savefig('A6d1.png')

    plt.figure(figsize=(10, 6))
    plt.plot(model.training_err, label='Training Error')
    plt.plot(model.test_err, label='Test Error')
    plt.xlabel('Iteration')
    plt.ylabel('Error')
    plt.legend()
    plt.savefig('A6d2.png')


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

B.1

First we show that $||x||_2 \leq ||x||_1$:

$$||x||_2^2 = \sum_{i=1}^{n} |x_i|^2 \leq \left(\sum_{i=1}^{n} |x_i|^2 + 2 \sum_{i \neq j} |x_i||x_j| \right) = ||x||_1^2, \tag{16}$$

then we show that $||x||_\infty \leq ||x||_2$:

$$||x||_\infty^2 = (\max_{i=1,\cdots,n} |x_i|)^2 = (\max_{i=1,\cdots,n} |x_i|^2) \leq \sum_{i=1}^{n} |x_i|^2 = ||x||_2^2. \tag{17}$$

Combing the above inequalities, we have:

$$\|x\|_\infty \leq \|x\|_2 \leq \|x\|_1. \tag{18}$$

B.2

B.2.a

To show that $f(x) = ||x||$ is a convex we need to show $f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y)$ for all $x, y \in A$ and $\lambda \in [0,1]$, by using the absolute scalability and triangle inequality we previously proven:

$$f(\lambda x + (1-\lambda)y) = ||\lambda x + (1-\lambda)y|| \tag{19}$$
$$\leq ||\lambda x|| + ||(1-\lambda)y|| \tag{20}$$
$$\leq \lambda||x|| + (1-\lambda)||y|| = \lambda f(x) + (1-\lambda)f(y). \tag{21}$$

B.2.b

To show that $A = \{x \in \mathbb{R}^n : \|x\| \leq 1\}$ is a convex set we need to show $\lambda x + (1-\lambda)y \in A$ for all $x, y \in A$ and $\lambda \in [0,1]$:

$$\lambda x + (1-\lambda)y \leq \lambda||x|| + (1-\lambda)||y|| \tag{22}$$
$$\leq \lambda + (1-\lambda) = 1 \in \{x \in \mathbb{R}^n : \|x\| \leq 1\}. \tag{23}$$

B.2.c

The set $G = \{(x_1, x_2) \ : \ g(x_1, x_2) \leq 4\}$ where $g(x_1, x_2) = \left(|x_1|^{1/2} + |x_2|^{1/2}\right)^2$ **is not convex**. We can simply show that be choose x as $(0, 4)$ and y as $(4, 0)$ which will not follow that $\lambda x + (1-\lambda)y \in G$.

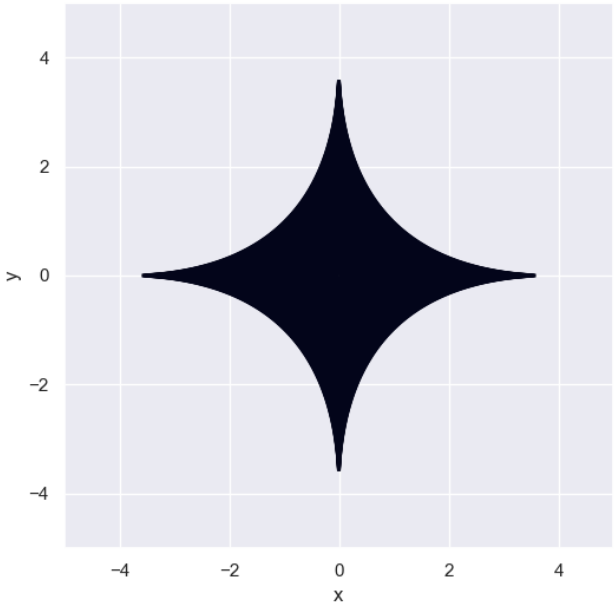

Figure 10: Picture of the set $\{(x_1, x_2) \ : \ g(x_1, x_2) \leq 4\}$ where $g(x_1, x_2) = \left(|x_1|^{1/2} + |x_2|^{1/2}\right)^2$.

B.3

B.3.a

We first start by showing that if $f(x), g(x)$ are convex functions, for any $x, y \in \mathbb{R}^d$:

$$f(\lambda x + (1-\lambda)y) + g(\lambda x + (1-\lambda)y) \leq f(\lambda x) + f((1-\lambda)y) + g(\lambda x) + g((1-\lambda)y) \tag{24}$$
$$= \lambda(f(x) + g(x)) + (1-\lambda)(f(y) + g(y)) \tag{25}$$

stands, then $f(x) + g(x)$ is also convex.

Then $\sum_{i=1}^{n} \ell_i(w)$ is also convex as the summation can be decompose into addition of two convex functions sequentially. Finally, we show that for any $\lambda > 0$, $1 \leq k \leq 0$ and the existing fact that $||w||$ is convex, for any $x, y \in \mathbb{R}^d$:

$$\lambda||kx + (1-k)y|| \leq \lambda||kx|| + \lambda||(1-k)y|| \tag{26}$$
$$= k(\lambda||x||) + (1-k)(\lambda||y||) \tag{27}$$

as we use the absolute scalability and triangle inequality properties of norm, $\lambda||w||$ is also convex.

By combining the above, $\sum_{i=1}^{n} \ell_i(w) + \lambda\|w\|$ is convex.

B.3.b

We prefer to use loss functions that are convex because this properties ensure that every local minimum will be a global minimum which is exactly what we hope for during training.

B.4

B.4.a

The negative log-likelihood function is equal to:

$$\mathcal{L}(W) = -\sum_{i=1}^{n}\sum_{\ell=1}^{k} \mathbf{1}\{y_i = \ell\} \log\left(\frac{\exp(\mathbf{w}^{(\ell)} \cdot \mathbf{x_i})}{\sum_{j=1}^{k} \exp(\mathbf{w}^{(j)} \cdot \mathbf{x_i})}\right) \tag{28}$$

And we can derive the $\nabla_W \mathcal{L}(W)$:

$$\begin{aligned}
\nabla_W \mathcal{L}(W) &= -\nabla_W \sum_{i=1}^{n}\sum_{\ell=1}^{k} \mathbf{1}\{y_i = \ell\} \log\left(\frac{\exp(\mathbf{w}^{(\ell)} \cdot \mathbf{x_i})}{\sum_{j=1}^{k} \exp(\mathbf{w}^{(j)} \cdot \mathbf{x_i})}\right) && (29)\\
&= -\nabla_W \sum_{i=1}^{n}\sum_{\ell=1}^{k} \mathbf{1}\{y_i = \ell\} \left(\log(\exp(\mathbf{w}^{(\ell)} \cdot \mathbf{x_i})) - log\left(\sum_{j=1}^{k} \exp(\mathbf{w}^{(j)} \cdot \mathbf{x_i})\right)\right) && (30)\\
&= -\nabla_W \sum_{i=1}^{n}\sum_{\ell=1}^{k} \mathbf{1}\{y_i = \ell\} \left(\mathbf{w}^{(\ell)} \cdot \mathbf{x_i} - log\left(\sum_{j=1}^{k} \exp(\mathbf{w}^{(j)} \cdot \mathbf{x_i})\right)\right) && (31)\\
&= -\nabla_W \sum_{i=1}^{n} \left(\sum_{\ell=1}^{k} \mathbf{1}\{y_i = \ell\}\mathbf{w}^{(\ell)} \cdot \mathbf{x_i} - \sum_{\ell=1}^{k} \mathbf{1}\{y_i = \ell\} log\left(\sum_{j=1}^{k} \exp(\mathbf{w}^{(j)} \cdot \mathbf{x_i})\right)\right) && (32)\\
&= -\nabla_W \sum_{i=1}^{n} \left(\sum_{\ell=1}^{k} \mathbf{1}\{y_i = \ell\} \left(\mathbf{w}^{(\ell)} \cdot \mathbf{x_i}\right) - log\left(\sum_{j=1}^{k} \exp(\mathbf{w}^{(j)} \cdot \mathbf{x_i})\right)\right) && (33)\\
&= -\sum_{i=1}^{n} \left(\sum_{\ell=1}^{k} \mathbf{1}\{y_i = \ell\}\mathbf{x_i} - \frac{\mathbf{w}^{(l)} \cdot \mathbf{x_i}}{\sum \mathbf{w}^{(j)} \cdot \mathbf{x_i}}\right) && (34)\\
&= -\sum_{i=1}^{n} \mathbf{x_i}(\mathbf{y}_i - \widehat{\mathbf{y}}_i^{(W)})^\top && (35)
\end{aligned}$$

B.4.b

The least squares linear regression defined $J(W)$ as:

$$J(W) = \frac{1}{2}\sum_{i=1}^{n} \|\mathbf{y}_i - W^\top \mathbf{x}_i\|_2^2 \tag{36}$$

And we can derive the $\nabla_W J(W)$:

$$\begin{aligned}
\nabla_W J(W) &= \nabla_W \frac{1}{2}\sum_{i=1}^{n} \|\mathbf{y}_i - W^\top \mathbf{x}_i\|_2^2 && (37)\\
&= \frac{1}{2}\sum_{i=1}^{n} 2(\mathbf{y}_i - W^\top \mathbf{x}_i)\nabla_W(\mathbf{y}_i - W^\top \mathbf{x}_i) && (38)\\
&= -\sum_{i=1}^{n} \mathbf{x}_i(\mathbf{y}_i - W^\top \mathbf{x}_i)^\top && (39)\\
&= -\sum_{i=1}^{n} \mathbf{x}_i(\mathbf{y}_i - \widetilde{\mathbf{y}}_i^{(W)})^\top && (40)
\end{aligned}$$

B.4.c

These are the results of gradient descent implementations for both $L(W)$ and $J(W)$ with step being 0.05 and epoch being 30:

- Training Accuracy for $L(W)$: 91.3115%
- Testing Accuracy for $L(W)$: 89.6680%
- Training Accuracy for $J(W)$: 84.7457%
- Testing Accuracy for $J(W)$: 83.5840%

B.4.code

```
import torch
import torchvision

from tqdm import tqdm


def load_dataset():
    MNIST_transform = torchvision.transforms.Compose(
        [
            torchvision.transforms.ToTensor(),
            torchvision.transforms.Lambda(lambda x: torch.flatten(x))
        ]
    )
    train_dataset = torchvision.datasets.MNIST(root='./data', train=True,
        download=True, transform=MNIST_transform)
    test_dataset = torchvision.datasets.MNIST(root='./data', train=False,
        download=True, transform=MNIST_transform)
    train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=256,
        shuffle=True)
    test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=256,
        shuffle=False)
    return train_loader, test_loader


def eval(dataloader, W):
    correct = 0
    total = len(dataloader) * 256
    for X, Y in dataloader:
        Y_pred = torch.matmul(X, W)
        correct += torch.sum(torch.argmax(Y_pred, 1) == Y)
    return (correct/total).item()


def train_CE(epochs, step_size, train_loader, test_loader):
    W = torch.zeros(784, 10, requires_grad=True)
    for epoch in range(epochs):
        correct = 0
        total = len(train_loader) * 256
        for X_train, Y_train in tqdm(train_loader):
            Y_pred = torch.matmul(X_train, W)
            # cross entropy combines softmax calculation with NLLLoss
            loss = torch.nn.functional.cross_entropy(Y_pred, Y_train)
```

```
            # computes derivatives of the loss with respect to W
            loss.backward()
            # gradient descent update
            W.data = W.data - step_size * W.grad
            # .backward() accumulates gradients into W.grad instead
            # of overwriting, so we need to zero out the weights
            W.grad.zero_()
            correct += torch.sum(torch.argmax(Y_pred, 1) == Y_train)
            # print(loss)
        print('Training Acc: {} | Testing Acc: {}'.format((correct/total).item()
            , eval(test_loader, W)))


def train_MSE(epochs, step_size, train_loader, test_loader):
    W = torch.zeros(784, 10, requires_grad=True)
    for epoch in range(epochs):
        correct = 0
        total = len(train_loader) * 256
        for X_train, Y_train in tqdm(train_loader):
            Y_pred = torch.matmul(X_train, W)
            loss = torch.nn.functional.mse_loss(Y_pred, torch.nn.functional.
                one_hot(Y_train, 10).type(torch.FloatTensor))
            loss.backward()
            W.data = W.data - step_size * W.grad
            W.grad.zero_()
            correct += torch.sum(torch.argmax(Y_pred, 1) == Y_train)
            # print(loss)
        print('Training Acc: {} | Testing Acc: {}'.format((correct / total).item
            (), eval(test_loader, W)))


def main():
    train_loader, test_loader = load_dataset()
    train_CE(30, 0.05, train_loader, test_loader)
    train_MSE(30, 0.05, train_loader, test_loader)


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

B.5

B.5.a

From the problem statement, we use the least squares estimator:

$$\widehat{\beta} = \min_{\beta} \|X\beta - Y\|_2^2 \tag{41}$$

where we compute the derivative and set it to 0:

$$\widehat{\beta} = \min_{\beta} \|X\beta - Y\|_2^2 \tag{42}$$

$$\Rightarrow 2X(X\widehat{\beta} - Y) = 0 \tag{43}$$

$$\Rightarrow X\widehat{\beta} = Y \tag{44}$$

$$\Rightarrow X^T X\widehat{\beta} = X^T Y \tag{45}$$

$$\Rightarrow \widehat{\beta} = (X^T X)^{-1} X^T Y \tag{46}$$

$$\tag{47}$$

then from the proposition from the notes we know that:

$$X \sim \mathcal{N}(\mu, \Sigma) \Rightarrow AX + b \sim \mathcal{N}(A\mu + b, A^T) \tag{48}$$

we have:

$$Y \sim \mathcal{N}(X\beta, 1) \tag{49}$$

$$\Rightarrow X^T Y \sim \mathcal{N}(X^T X\beta, X^T X) \tag{50}$$

$$\Rightarrow (X^T X)^{-1} X^T Y \sim \mathcal{N}(\beta, (X^T X)^{-1}) \tag{51}$$

$$\Rightarrow \widehat{\beta}_j \sim \mathcal{N}(\beta^*)j, (X^T X)^{-1}_{j,j}). \tag{52}$$

B.5.b

No, we can not conclude that with probability at least $1 - \delta$, $|\widehat{\beta}_j| \leq \sqrt{2(X^T X)^{-1}_{j,j} \log(2/\delta)}$ for all $j \in [d]$ simultaneously. The probability addressed in this sub-problem is consider as point-wise confidence band which is independent for each β_j and if we want to consider all j at the same time, then we will need to consider the simultaneous confidence interval of β as a whole.

B.5.c

There are **03** $\widehat{\beta}_j$ outside the confidence interval with $1 - \sigma = 0.95$.

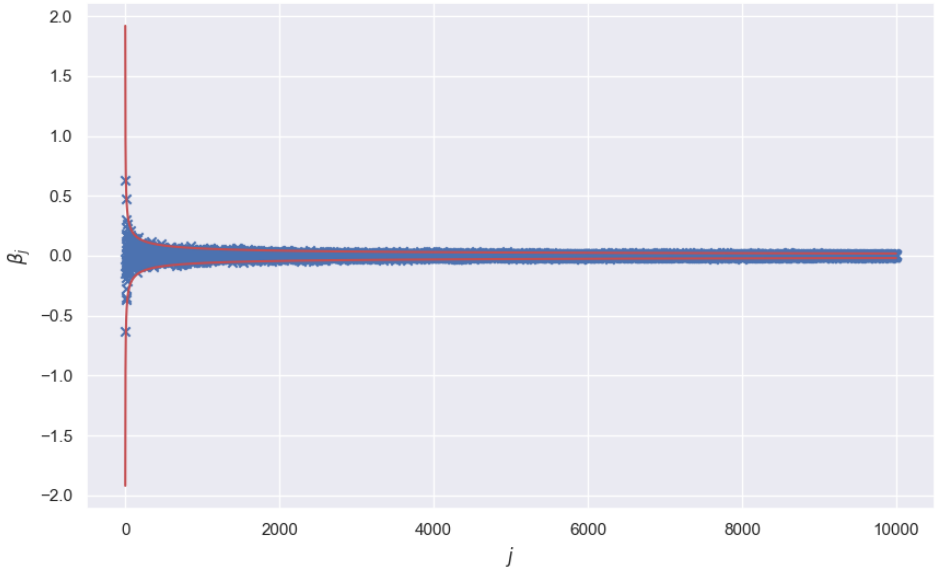

Figure 11: Plot of $\widehat{\beta}_j$ with the confidence interval of $1 - \sigma = 0.95$.

B.5.d

From the notes on confidence intervals, we take a union bound over all $i \in \{1, \ldots, d\}$:

$$\sum_{i=1}^{d} \mathbf{P}\left(|\widehat{\theta}_i - \theta_{*,i}| > \sqrt{2\sigma^2 (X^\top X)^{-1}_{i,i} log(2d/\delta)}\right) \leq \delta \tag{53}$$

we chose the threshold $\gamma = \sqrt{2\sigma^2 (X^\top X)^{-1}_{i,i} log(2d/\delta)} = 1.92065$ so that the probability that any of the coefficients $|\widehat{\beta}_i - \beta_{*,i}|$ exceed γ is less than δ.

B.5.e

B.5.f