Homework #2

CSE 446/546: Machine Learning
Profs. Kevin Jamieson and Ludwig Schmidt
Due: Friday 6th May, 2022 11:59pm
108 points(A)/24 points(B)

Please review all homework guidance posted on the website before submitting to GradeScope. Reminders:

- Make sure to read the "What to Submit" section following each question and include all items.
- Please provide succinct answers and supporting reasoning for each question. Similarly, when discussing experimental results, concisely create tables and/or figures when appropriate to organize the experimental results. All explanations, tables, and figures for any particular part of a question must be grouped together.
- For every problem involving generating plots, please include the plots as part of your PDF submission.
- When submitting to Gradescope, please link each question from the homework in Gradescope to the location of its answer in your homework PDF. **Failure to do so may result in deductions of up to 10% of the points for each problem improperly tagged**. For instructions, see `https://www.gradescope.com/get_started#student-submission`.
- If you collaborate on this homework with others, you must indicate who you worked with on your homework. Failure to do so may result in accusations of plagiarism.
- **The coding problems are available in a .zip file on the course website**, with some starter code. All coding questions in this class will have starter code. **Before attempting these problems make sure your coding environment is working**. See instructions in README file in the .zip file.
- For every problem involving code, please include the code as part of your PDF for the PDF submission *in addition to* submitting your code to the separate assignment on Gradescope created for code.

Not adhering to these reminders may result in point deductions.

Short Answer and "True or False" Conceptual questions

A1. The answers to these questions should be answerable without referring to external materials. Briefly justify your answers with a few words.

a. *[2 points]* Suppose that your estimated model for predicting house prices has a large positive weight on the feature `number of bathrooms`. If we remove this feature and refit the model, will the new model have a strictly higher error than before? Why?

b. *[2 points]* Compared to L2 norm penalty, explain why a L1 norm penalty is more likely to result in sparsity (a larger number of 0s) in the weight vector.

c. *[2 points]* In at most one sentence each, state one possible upside and one possible downside of using the following regularizer: $\left(\sum_i |w_i|^{0.5}\right)$.

d. *[1 point]* True or False: If the step-size for gradient descent is too large, it may not converge.

e. *[2 points]* In your own words, describe why stochastic gradient descent (SGD) works, even though only a small portion of the data is considered at each update.

f. *[2 points]* In at most one sentence each, state one possible advantage of SGD over GD (gradient descent), and one possible disadvantage of SGD relative to GD.

What to Submit:

- a). No, If there were many features with a large positive weight and related to bathrooms, then remove bathroom wouldn't higher the error a lot. Or the error even can be smaller if the bathroom's true weight wasn't quite large.
- b). In the 2D graph, L2 is a circle but L1 is a square. This means when $\hat{w}$ achieves the minimum, L1 is more likely to intersect it at axis point (0).
- c). Upside: more sparsity than L1.
  Downside: The prediction may not be very accurate, since many feature will become 0.
- d). True, it may miss the minimum point
- e). This is because each time, SGD will find the minimum point by choosing a random point and based on the Convex theory, as long as the data set is large enough and SGD takes enough iterations, it can eventually converge.
- f). advantage of SGD over GD: Less computation.
  disadvantage of SGD relative to GD: SGD may not find the exactly minimum at each iteration.

Convexity and Norms

A2. A *norm* $\|\cdot\|$ over $\mathbb{R}^n$ is defined by the properties: (i) non-negativity: $\|x\| \geq 0$ for all $x \in \mathbb{R}^n$ with equality if and only if $x = 0$, (ii) absolute scalability: $\|a\,x\| = |a|\,\|x\|$ for all $a \in \mathbb{R}$ and $x \in \mathbb{R}^n$, (iii) triangle inequality: $\|x + y\| \leq \|x\| + \|y\|$ for all $x, y \in \mathbb{R}^n$.

a. *[3 points]* Show that $f(x) = (\sum_{i=1}^n |x_i|)$ is a norm. (Hint: for (iii), begin by showing that $|a+b| \leq |a|+|b|$ for all $a, b \in \mathbb{R}$.)

b. *[2 points]* Show that $g(x) = \left(\sum_{i=1}^n |x_i|^{1/2}\right)^2$ is not a norm. (Hint: it suffices to find two points in $n = 2$ dimensions such that the triangle inequality does not hold.)

Context: norms are often used in regularization to encourage specific behaviors of solutions. If we define $\|x\|_p := (\sum_{i=1}^n |x_i|^p)^{1/p}$ then one can show that $\|x\|_p$ is a norm for all $p \geq 1$. The important cases of $p = 2$ and $p = 1$ correspond to the penalty for ridge regression and the lasso, respectively.

What to Submit:

- a). $|a - b| = |a + (-b)| \leq |a| + |-b| = |a| + |b|$, so $|a + b| \leq |a| + |b|$
  non-negativity: Since f(x) equal to the sum of $|x_i|$, so $f(x) \geq 0$ because for all $x_i, |x_i| \geq 0$. And $f(x) = 0$ only if all $x_i = 0$

  absolute scalability: $\sum_{i=1}^n |ax_i| = \sum_{i=1}^n |a||x_i| = |a| \sum_{i=1}^n |x_i| = |a| f(x)$

  triangle inequality: $f(x + y) = \sum_{i=1}^n |x_i + y_i| \leq \sum_{i=1}^n (|x_i| + |y_i|) = \sum_{i=1}^n |x_i| + \sum_{i=1}^n |y_i| = f(x) + f(y)$

- b). Let x = (0,1) and y = (1,0).
  $f(x) = f(y) = 1$, but $f(x + y) = 4$. So $f(x) + f(y) < f(x + y)$, the triangle inequality doesn't hold.

A3. *[3 points]* A set $A \subseteq \mathbb{R}^n$ is *convex* if $\lambda x + (1-\lambda)y \in A$ for all $x, y \in A$ and $\lambda \in [0,1]$. For each of the grey-shaded sets below (I-III), state whether each one is convex, or state why it is not convex using any of the points a, b, c, d in your answer.

What to Submit:

- I). Not convex, the line between b, c lies outside the set.
- II). Convex, all line lie inside the set.
- III). Not convex, the line between a, d lies outside the set.

A4. *[4 points]* We say 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 $\lambda \in [0, 1]$. For each of the functions shown below (I-III), state whether each is convex on the specified interval, or state why not with a counterexample using any of the points a, b, c, d in your answer.

a. Function in panel I on $[a, c]$

b. Function in panel II on $[a, c]$

c. Function in panel III on $[a, d]$

d. Function in panel III on $[c, d]$

What to Submit:

- a). Convex, all the line segment of this function is lie above the function.
- b). Not Convex, lines segment between a, b of this function is lie below the function.
- c). Not Convex, lines segment between a, c of this function is lie below the function.
- d). Convex, all the line segment of this function between c, d is lie above the function.

Lasso on a Real Dataset

Given $\lambda > 0$ and data $\left(x_i, y_i\right)_{i=1}^n$, the Lasso is the problem of solving

$$\arg \min_{w \in \mathbb{R}^d, b \in \mathbb{R}} \sum_{i=1}^n (x_i^T w + b - y_i)^2 + \lambda \sum_{j=1}^d |w_j|$$

where λ is a regularization parameter. For the programming part of this homework, you will implement the coordinate descent method shown in Algorithm 1 to solve the Lasso problem.
You may use common computing packages (such as `numpy` or `scipy`), but do not use an existing Lasso solver (e.g., of `scikit-learn`).

Algorithm 1: Coordinate Descent Algorithm for Lasso

while *not converged* **do**
  $b \leftarrow \frac{1}{n} \sum_{i=1}^n \left(y_i - \sum_{j=1}^d w_j x_{i,j}\right)$
  for $k \in \{1, 2, \cdots d\}$ **do**
    $a_k \leftarrow 2 \sum_{i=1}^n x_{i,k}^2$
    $c_k \leftarrow 2 \sum_{i=1}^n x_{i,k} \left(y_i - (b + \sum_{j \neq k} w_j x_{i,j})\right)$
    $w_k \leftarrow \begin{cases} (c_k + \lambda)/a_k & c_k < -\lambda \\ 0 & c_k \in [-\lambda, \lambda] \\ (c_k - \lambda)/a_k & c_k > \lambda \end{cases}$
  end
end

Before you get started, the following hints may be useful:

- Wherever possible, use matrix libraries for matrix operations (not `for` loops).
- There are opportunities to considerably speed up parts of the algorithm by precomputing quantities like a_k before the for loop; you are permitted to add these improvements (and it may save you some time).
- As a sanity check, ensure the objective value is nonincreasing with each step.
- It is up to you to decide on a suitable stopping condition. A common criteria is to stop when no element of w changes by more than some small δ during an iteration. If you need your algorithm to run faster, an easy place to start is to loosen this condition.
- You will need to solve the Lasso on the same dataset for many values of λ. This is called a regularization path. One way to do this efficiently is to start at a large λ, and then for each consecutive solution, initialize the algorithm with the previous solution, decreasing λ by a constant ratio (e.g., by a factor of 2).
- The smallest value of λ for which the solution $\widehat{w}$ is entirely zero is given by

$$\lambda_{max} = \max_{k=1,\ldots,d} 2 \left| \sum_{i=1}^n x_{i,k} \left(y_i - \left(\frac{1}{n} \sum_{j=1}^n y_j \right) \right) \right| \tag{1}$$

  This is helpful for choosing the first λ in a regularization path.

A5. We will first try out your solver with some synthetic data. A benefit of the Lasso is that if we believe many features are irrelevant for predicting y, the Lasso can be used to enforce a sparse solution, effectively differentiating between the relevant and irrelevant features. Suppose that $x \in \mathbb{R}^d, y \in \mathbb{R}, k < d$, and data are generated independently according to the model $y_i = w^T x_i + \epsilon_i$ where

$$w_j = \begin{cases} j/k & \text{if } j \in \{1, \ldots, k\} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

and $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ is noise (note that in the model above $b = 0$). We can see from Equation (2) that since $k < d$ and $w_j = 0$ for $j > k$, the features $k + 1$ through d are irrelevant for predicting y.
Generate a dataset using this model with $n = 500, d = 1000, k = 100$, and $\sigma = 1$. You should generate the dataset such that each $\epsilon_i \sim \mathcal{N}(0, 1)$, and y_i is generated as specified above. You are free to choose a distribution from which the x's are drawn, but make sure standardize the x's before running your experiments.

a. *[10 points]* With your synthetic data, solve multiple Lasso problems on a regularization path, starting at λ_{max} where no features are selected (see Equation (1)) and decreasing λ by a constant ratio (e.g., 2) until nearly all the features are chosen. In plot 1, plot the number of non-zeros as a function of λ on the x-axis (Tip: use `plt.xscale('log')`).

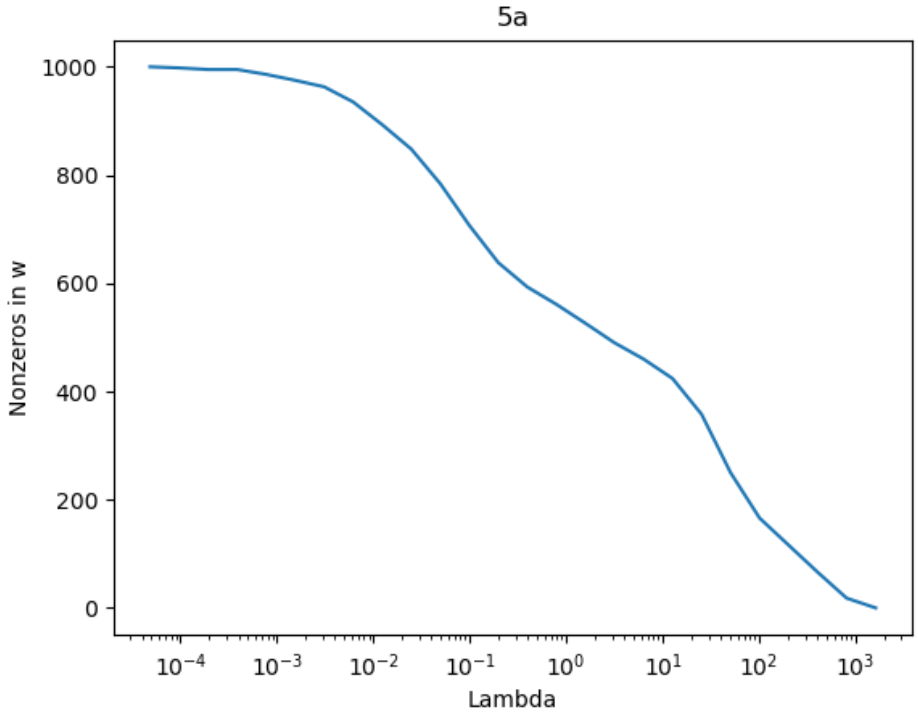
b. *[10 points]* For each value of λ tried, record values for false discovery rate (FDR) (number of incorrect nonzeros in $\widehat{w}$/total number of nonzeros in $\widehat{w}$) and true positive rate (TPR) (number of correct nonzeros in $\widehat{w}$/k). Note: for each j, $\widehat{w}_j$ is an incorrect nonzero if and only if $\widehat{w}_j \neq 0$ while $w_j = 0$. In plot 2, plot these values with the x-axis as FDR, and the y-axis as TPR.

   Note that in an ideal situation we would have an (FDR,TPR) pair in the upper left corner. We can always trivially achieve $(0, 0)$ and $(\frac{d-k}{d}, 1)$.

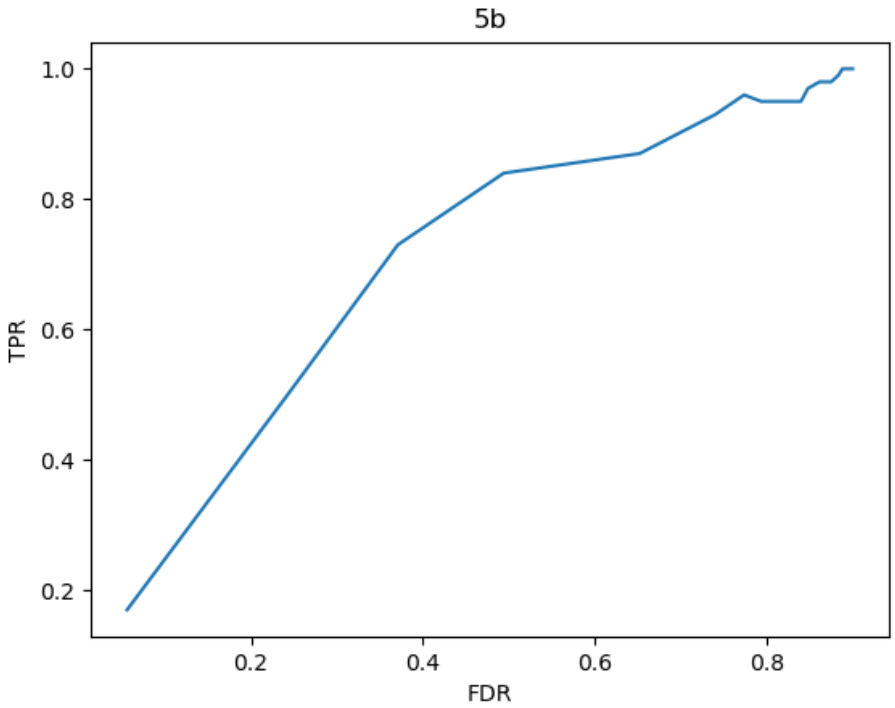
c. *[5 points]* Comment on the effect of λ in these two plots in 1-2 sentences.

What to Submit:

- Part a:)



- Part b:)



- Part c:) As λ increase, it makes less features been selected and makes both TRP and FDR increased as well.

Listing 1: A5

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


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

    b = 1/X.shape[0] * np.sum(y - np.dot(X, weight))
    for k in range(X.shape[1]):
        c_k = 2*np.dot(X[:, k], (y - (b + np.dot(X, weight) - X[:, k] * weight[k])))

        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, b


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

    loss = 0
    norm = 0
    for n in range(X.shape[0]):
        loss += (np.dot(X[n], np.transpose(weight)) + bias - y[n])**2
        norm = np.linalg.norm(weight, 1)
    return loss + _lambda * norm


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

    b = 0
    while not convergence_criterion(start_weight, old_w, convergence_delta):
        old_w = np.copy(start_weight)
        start_weight, b = step(X, y, start_weight, a, _lambda)

    return start_weight, b


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

    if old_w is None:
        return False
    else:
        return np.max(np.abs(old_w - weight)) < convergence_delta




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

    w = np.arange(1000) / 100
    w[100 + 1:] = 0
    X = np.random.normal(0.0, 1.0, (500, 1000))
    y = np.dot(X, w) + np.random.normal(0.0, 1, (500, ))


    lam_max = 2*np.max(np.abs(np.dot(y.T - np.mean(y), X)))
    cur_lam = lam_max
    lam_list = []
    W = np.zeros((1000, 1))
    prev_w = None

    FDR = []
    TPR = []

    while np.count_nonzero(W[:, -1]) != 1000:
        lam_list.append(cur_lam)
        cur_lam = cur_lam / 2

        w_train, b = train(X, y, _lambda=cur_lam, start_weight=prev_w)
        W = np.concatenate((W, np.expand_dims(w_train, axis=1)), axis=1)
        non_zero_w_train = np.count_nonzero(w_train)

        if (non_zero_w_train != 0):
```

```
            FDR.append(np.count_nonzero(w_train[100:]) / non_zero_w_train)
        else:
            FDR.append(0)

        TPR.append(np.count_nonzero(w_train[:100]) / 100)
        prev_w = np.copy(w_train)


# 5.a ------------------------------------------------------------------
    lam_list.append(cur_lam)
    plt.figure(1)
    plt.xscale('log')
    plt.plot(lam_list, np.count_nonzero(W, axis=0))
    plt.xlabel('Lambda')
    plt.ylabel('Nonzeros in w')
    plt.title('5a')
    plt.show()

# 5.b ------------------------------------------------------------------
    plt.figure(2)
    plt.plot(FDR, TPR)
    plt.xlabel('FDR')
    plt.ylabel('TPR')
    plt.title('5b')
    plt.show()


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

A6. We'll now put the Lasso to work on some real data. Download the training data set "crime-train.txt" and the test data set "crime-test.txt" from the website. Store your data in your working directory, ensure you have the `pandas` library for Python installed, and read in the files with:

```
import pandas as pd
df_train = pd.read_table("crime-train.txt")
df_test = pd.read_table("crime-test.txt")
```

This stores the data as Pandas `DataFrame` objects. `DataFrame`s are similar to Numpy `array`s but more flexible; unlike `array`s, `DataFrame`s store row and column indices along with the values of the data. Each column of a `DataFrame` can also store data of a different type (here, all data are floats). Here are a few commands that will get you working with Pandas for this assignment:

```
df.head()                    # Print the first few lines of DataFrame df.
df.index                     # Get the row indices for df.
df.columns                   # Get the column indices.
df[''foo'']                 # Return the column named ''foo''.
df.drop(''foo'', axis = 1)  # Return all columns except ''foo''.
df.values                    # Return the values as a Numpy array.
df[''foo''].values          # Grab column foo and convert to Numpy array.
df.iloc[:3,:3]               # Use numerical indices (like Numpy) to get 3 rows and cols.
```

The data consist of local crime statistics for 1,994 US communities. The response y is the rate of violent crimes reported per capita in a community. The name of the response variable is `ViolentCrimesPerPop`, and it is held in the first column of `df_train` and `df_test`. There are 95 features. These features include many variables. Some features are the consequence of complex political processes, such as the size of the police force and other systemic and historical factors. Others are demographic characteristics of the community, including self-reported statistics about race, age, education, and employment drawn from Census reports.

The goals of this problem are threefold: (i) to encourage you to think about how data collection processes affect the resulting model trained from that data; (ii) to encourage you to think deeply about models you might train and how they might be misused; and (iii) to see how Lasso encourages sparsity of linear models in settings where d is large relative to n. **We emphasize that training a model on this dataset can suggest a degree of correlation between a community's demographics and the rate at which a community experiences and reports violent crime. We strongly encourage students to consider why these correlations may or may not hold more generally, whether correlations might result from a common cause, and what issues can result in misinterpreting what a model can explain.**

The dataset is split into a training and test set with 1,595 and 399 entries, respectively[1]. We will use this training set to fit a model to predict the crime rate in new communities and evaluate model performance on the test set. As there are a considerable number of input variables and fairly few training observations, overfitting is a serious issue. In order to avoid this, use the coordinate descent Lasso algorithm implemented in the previous problem.

a. *[4 points]* Read the documentation for the original version of this dataset: `http://archive.ics.uci.edu/ml/datasets/communities+and+crime`. Report 3 features included in this dataset for which historical *policy* choices in the US would lead to variability in these features. As an example, the *number of police* in a community is often the consequence of decisions made by governing bodies, elections, and amount of tax revenue available to decision makers.

b. *[4 points]* Before you train a model, describe 3 features in the dataset which might, if found to have nonzero weight in model, be interpreted as *reasons* for higher levels of violent crime, but which might actually be a *result* rather than (or in addition to being) the cause of this violence.

[1]The features have been standardized to have mean 0 and variance 1.

Now, we will run the Lasso solver. Begin with $\lambda = \lambda_{\max}$ defined in Equation (1). Initialize all weights to 0. Then, reduce λ by a factor of 2 and run again, but this time initialize $\hat{w}$ from your $\lambda = \lambda_{\max}$ solution as your initial weights, as described above. Continue the process of reducing λ by a factor of 2 until $\lambda < 0.01$. For all plots use a log-scale for the λ dimension (Tip: use `plt.xscale('log')`).

c. *[4 points]* Plot the number of nonzero weights of each solution as a function of λ.

d. *[4 points]* Plot the regularization paths (in one plot) for the coefficients for input variables `agePct12t29`, `pctWSocSec`, `pctUrban`, `agePct65up`, and `householdsize`.

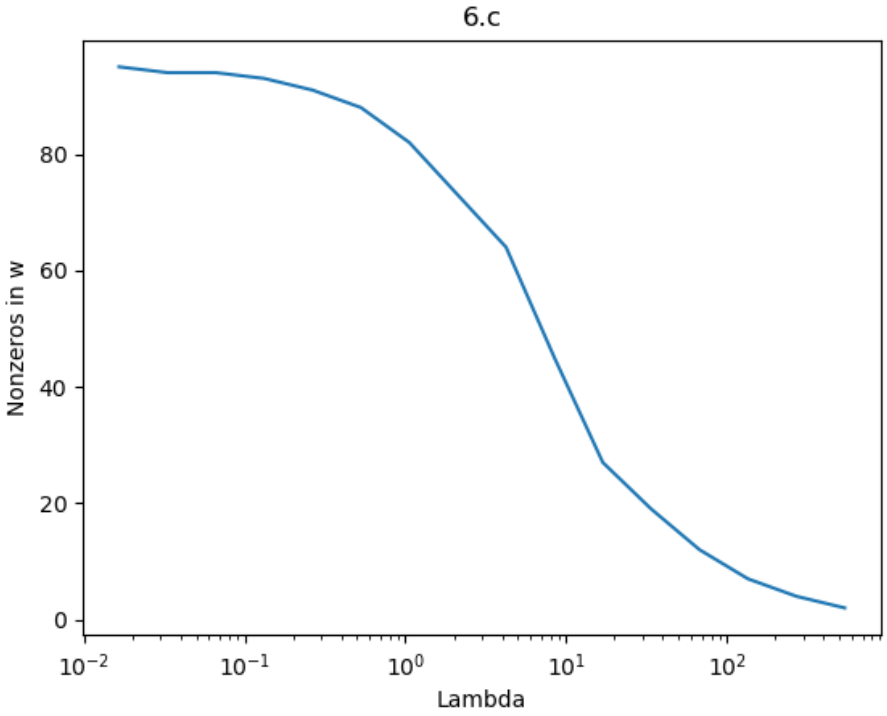
e. *[4 points]* On one plot, plot the squared error on the training and test data as a function of λ.

f. *[4 points]* Sometimes a larger value of λ performs nearly as well as a smaller value, but a larger value will select fewer variables and perhaps be more interpretable. Inspect the weights $\hat{w}$ for $\lambda = 30$. Which feature had the largest (most positive) Lasso coefficient? What about the most negative? Discuss briefly.

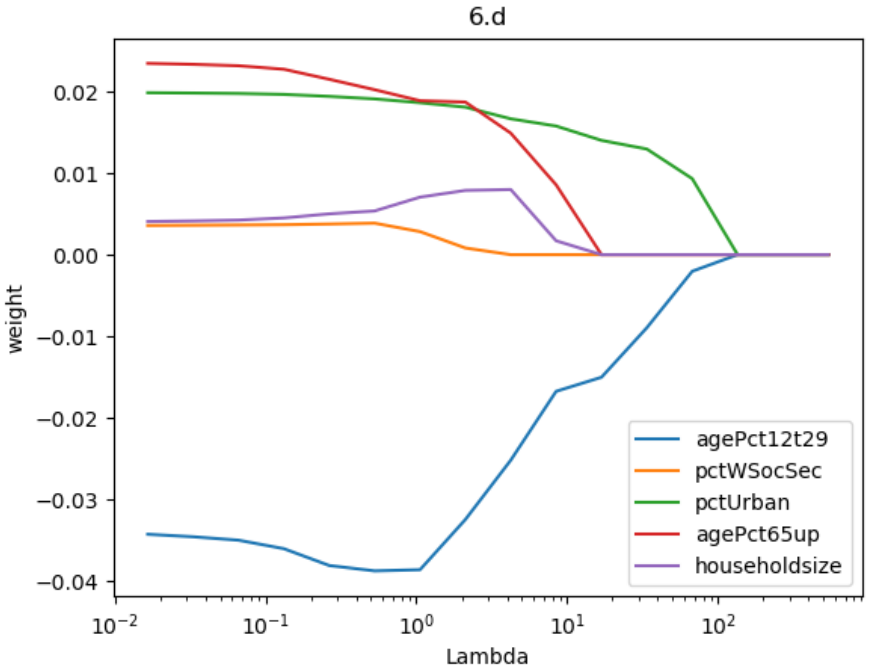
g. *[4 points]* Suppose there was a large negative weight on `agePct65up` and upon seeing this result, a politician suggests policies that encourage people over the age of 65 to move to high crime areas in an effort to reduce crime. What is the (statistical) flaw in this line of reasoning? (Hint: fire trucks are often seen around burning buildings, do fire trucks cause fire?)

What to Submit:

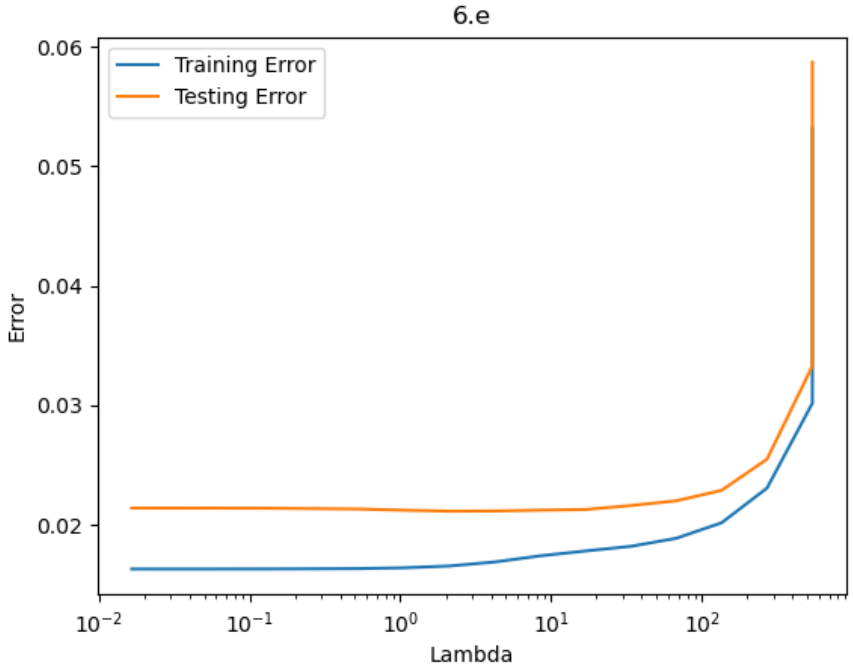
- a:) The newborn allowance depend no median household income, percentage of moms of kids 6 and under in labor force and mean persons per household.
- b:) agePct65up: percentage of population that is 65 and over in age
  PctPopUnderPov : percentage of people under the poverty level
  PolicBudgPerPop: police operating budget per population
  The people's age, income and police budget may interpreted as reasons for higher levels of violent crime.
- Part c:)



- Part d:)



- Part e:)



- part f:) NumIlleg: number of kids born to never married, had the largest coefficien 0.06872528529683818, PctFam2Par: percentage of families (with kids) that are headed by two parents, had the smallest coefficien -0.06921580730627477.
- Part g:) The politician didn't understand what are correlation and causation. Old people isn't the cause of crime, and the old people who own better property may have a low crime rate than those who didn't.

Listing 2: A6

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

import matplotlib.pyplot as plt
import numpy as np

from utils import load_dataset, problem


@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")

    # raise NotImplementedError("Your Code Goes Here")
    # get data
    y_train = df_train.values[:,0]#.reshape(len(df_train),1)
    X_train = df_train.values[:,1:].reshape(len(df_train),df_train.shape[1]-1)
    y_test = df_test.values[:,0]#.reshape(len(df_test),1)
    X_test = df_test.values[:,1:].reshape(len(df_test),df_test.shape[1]-1)

    n, d = X_train.shape
    lam_max = 2*np.max(np.abs(np.dot(y_train.T - np.mean(y_train), X_train)))
    cur_lam = lam_max
    lam_list = [cur_lam]

    prev_w, b_train = train(X_train, y_train, _lambda=cur_lam)
    error_train = [np.mean(np.square(np.dot(X_train, prev_w) + b_train - y_train))]
    error_test = [np.mean(np.square(np.dot(X_test, prev_w) + b_train - y_test))]
    W = np.expand_dims(prev_w, axis=1)

    while cur_lam >= 0.01:
        lam_list.append(cur_lam)
        cur_lam = cur_lam / 2
        w_train, b_train = train(X_train, y_train, _lambda=cur_lam, start_weight=prev_w)
        error_train.append(np.mean(np.square(np.dot(X_train, prev_w) + b_train - y_train)))
        error_test.append(np.mean(np.square(np.dot(X_test, prev_w) + b_train - y_test)))
        W = np.concatenate((W, np.expand_dims(w_train, axis=1)), axis=1)
        prev_w = np.copy(w_train)

    # 6.c ---------------------------------------------------------------------
    plt.figure(1)
    plt.xscale('log')
    plt.plot(lam_list, np.count_nonzero(W, axis=0))
    plt.xlabel('Lambda')
    plt.ylabel('Nonzeros in w')
    plt.title('6.c')
    plt.show()

    # 6.d ---------------------------------------------------------------------
    # column index
```

```
    agePct12t29 = df_train.columns.get_loc("agePct12t29") - 1
    pctWSocSec = df_train.columns.get_loc("pctWSocSec") - 1
    pctUrban = df_train.columns.get_loc("pctUrban") - 1
    agePct65up = df_train.columns.get_loc("agePct65up") - 1
    householdsize = df_train.columns.get_loc("householdsize") - 1

    plt.figure(2)
    plt.xscale('log')
    plt.plot(lam_list, np.reshape(W[agePct12t29, :], (len(lam_list))))
    plt.plot(lam_list, np.reshape(W[pctWSocSec, :], (len(lam_list))))
    plt.plot(lam_list, np.reshape(W[pctUrban, :], (len(lam_list))))
    plt.plot(lam_list, np.reshape(W[agePct65up, :], (len(lam_list))))
    plt.plot(lam_list, np.reshape(W[householdsize, :], (len(lam_list))))
    plt.legend(["agePct12t29", "pctWSocSec", "pctUrban", "agePct65up", "householdsize"])
    plt.xlabel('Lambda')
    plt.ylabel('weight')
    plt.title('6.d')
    plt.show()

    # 6.e ------------------------------------------------------------------
    plt.figure(3)
    plt.xscale('log')
    plt.plot(lam_list, error_train)
    plt.plot(lam_list, error_test)
    plt.legend(["Training Error", "Testing Error"])
    plt.xlabel('Lambda')
    plt.ylabel('Error')
    plt.title('6.e')
    plt.show()

    # 6.f ------------------------------------------------------------------
    w30, b30 = train(X_train, y_train, _lambda=30, start_weight=None)
    largest = np.argmax(w30)
    smallest = np.argmin(w30)

    print ("index of larget feature: ", largest, " coefficien: ", w30[largest])
    print ("index of smallest feature: ", smallest, " coefficien: ", w30[smallest])



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

Logistic Regression

Binary Logistic Regression

A7. Here we consider the MNIST dataset, but for binary classification. Specifically, the task is to determine whether a digit is a 2 or 7. Here, let $Y = 1$ for all the "7" digits in the dataset, and use $Y = -1$ for "2". We will use regularized logistic regression. Given a binary classification dataset $\{(x_i, y_i)\}_{i=1}^n$ for $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$ we showed in class that the regularized negative log likelihood objective function can be written as

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

Note that the offset term b is not regularized. For all experiments, use $\lambda = 10^{-1}$. Let $\mu_i(w, b) = \frac{1}{1+\exp(-y_i(b+x_i^T w))}$.

a. *[8 points]* Derive the gradients $\nabla_w J(w, b)$, $\nabla_b J(w, b)$ and give your answers in terms of $\mu_i(w, b)$ (your answers should not contain exponentials).

b. *[8 points]* Implement gradient descent with an initial iterate of all zeros. Try several values of step sizes to find one that appears to make convergence on the training set as fast as possible. Run until you feel you are near to convergence.

   (a) For both the training set and the test, plot $J(w, b)$ as a function of the iteration number (and show both curves on the same plot).

   (b) For both the training set and the test, classify the points according to the rule $\text{sign}(b + x_i^T w)$ and plot the misclassification error as a function of the iteration number (and show both curves on the same plot).

   Reminder: Make sure you are only using the test set for evaluation (not for training).

c. *[7 points]* Repeat (b) using stochastic gradient descent with a batch size of 1. Note, the expected gradient with respect to the random selection should be equal to the gradient found in part (a). Show both plots described in (b) when using batch size 1. Take careful note of how to scale the learning rate.

d. *[7 points]* Repeat (b) using stochastic gradient descent with batch size of 100. That is, instead of approximating the gradient with a single example, use 100. Note, the expected gradient with respect to the random selection should be equal to the gradient found in part (a).

What to Submit

- **Part a:**
  Given: $J(w, b) = \frac{1}{n} \sum_{i=1}^{n} \log(1 + \exp(-y_i(b + x_i^T w))) + \lambda ||w||_2^2$

  And since $\mu_i(w, b) = \frac{1}{1+\exp(-y_i(b+x_i^T w))}$, so $\frac{1}{\mu_i(w,b)} - 1 = \exp(-y_i(b + x_i^T w))$

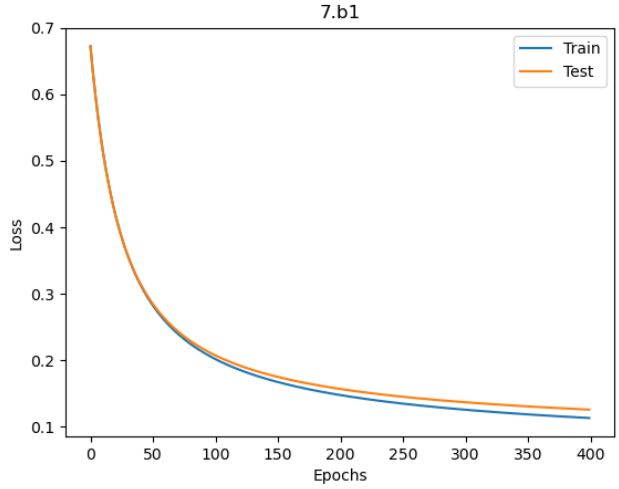
  $\nabla_w J(w, b) = \frac{1}{n} \sum_{i=1}^{n} \nabla_w \log(1 + \exp(-y_i(b + x_i^T w))) + \nabla_w \lambda ||w||_2^2$

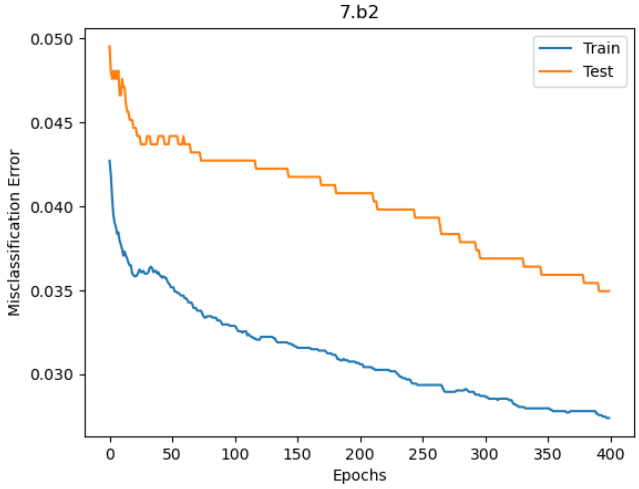
  . $= \frac{1}{n} \sum_{i=1}^{n} \mu_i(w, b)(\frac{1}{\mu_i(w,b)} - 1)(-y_i x_i^T) + 2\lambda w$

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

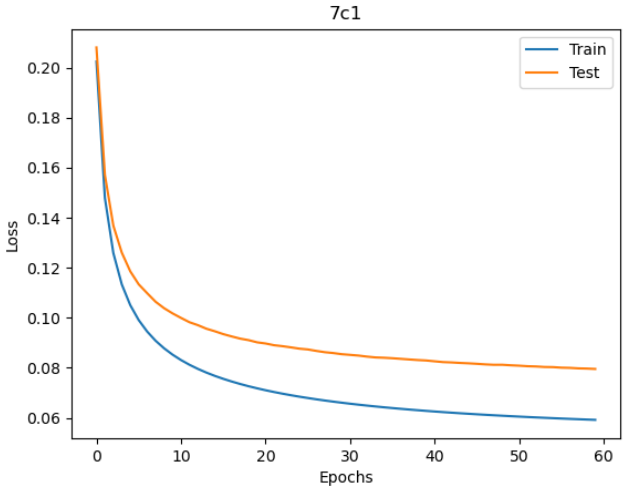
  $\nabla_b J(w, b) = \frac{1}{n} \sum_{i=1}^{n} (\mu_i(w, b) - 1) y_i$

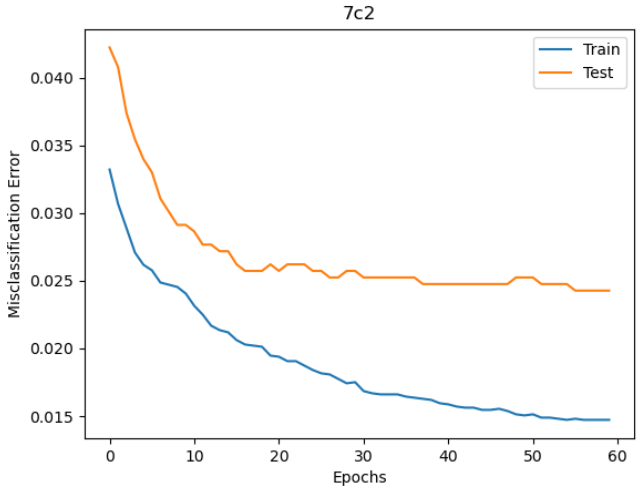
- **Part b:**



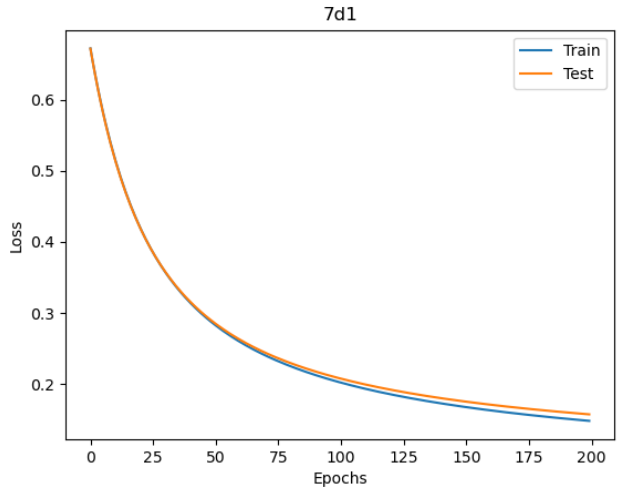


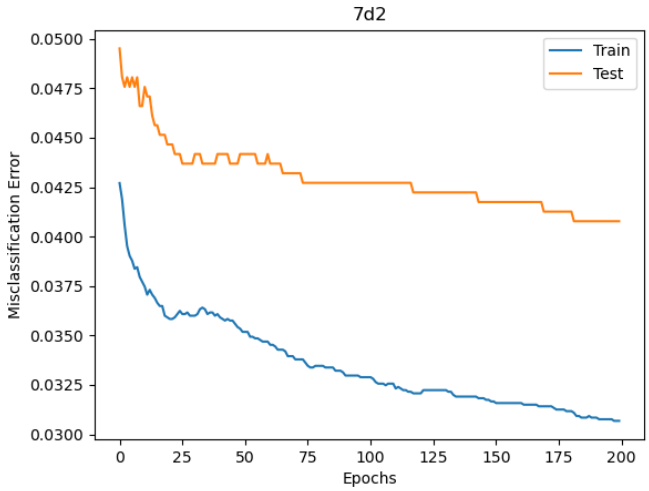
- **Part c:**





- **Part d:**





Listing 3: A7

```
from typing import Dict, List, Tuple

import matplotlib.pyplot as plt
import numpy as np

from utils import load_dataset, problem

RNG = np.random.RandomState(seed=446)
Dataset = Tuple[Tuple[np.ndarray, np.ndarray], Tuple[np.ndarray, np.ndarray]]


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

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

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

        return np.mean(np.log(1/self.mu(X,y))) + self._lambda * np.dot(self.weight.T, self.w

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

        return np.mean((self.mu(X,y) * y - y)[:,None]*X, axis = 0) + (2 * self._lambda * sel

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

        return np.mean((self.mu(X, y)-1) * y, axis=0)

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

        return np.sign(self.bias + np.dot(X, self.weight))

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

        return 1 - np.mean(y == self.predict(X))

    @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": [],
        }
        #raise NotImplementedError("Your Code Goes Here")
        n, d = X_train.shape
        self.weight = np.zeros(d)
        self.bias = 0

        for e in range(epochs):
            train_order = list(range(n))
            RNG.shuffle(train_order)

            X_train = X_train[train_order,]
            y_train = y_train[train_order,]

            for b in range(num_batches):
                X = X_train[b * batch_size : (b+1)*batch_size]
                y = y_train[b * batch_size : (b+1)*batch_size]
                self.step(X, y)



            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()

    #7b ------------------------------------------------------------------
    history_b = model.train(x_train, y_train, x_test, y_test, epochs=400)
    # Plot losses
    plt.figure(1)
    plt.plot(history_b["train_losses"], label="Train")
    plt.plot(history_b["test_losses"], label="Test")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.title('7b1')
    plt.legend()
    plt.show()
```

```
# Plot error
plt.figure(2)
plt.plot(history_b["train_errors"], label="Train")
plt.plot(history_b["test_errors"], label="Test")
plt.xlabel("Epochs")
plt.ylabel("Misclassification Error")
plt.title('7b2')
plt.legend()
plt.show()


#7c ----------------------------------------------------------------------
history_c = model.train(x_train, y_train, x_test, y_test, batch_size=1, epochs=60)
# Plot losses
plt.figure(3)
plt.plot(history_c["train_losses"], label="Train")
plt.plot(history_c["test_losses"], label="Test")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title('7c1')
plt.legend()
plt.show()

# Plot error
plt.figure(4)
plt.plot(history_c["train_errors"], label="Train")
plt.plot(history_c["test_errors"], label="Test")
plt.xlabel("Epochs")
plt.ylabel("Misclassification Error")
plt.title('7c2')
plt.legend()
plt.show()


#7d ----------------------------------------------------------------------
history_d = model.train(x_train, y_train, x_test, y_test, batch_size=100, epochs=200)
# Plot losses
plt.figure(5)
plt.plot(history_d["train_losses"], label="Train")
plt.plot(history_d["test_losses"], label="Test")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title('7d1')
plt.legend()
plt.show()

# Plot error
plt.figure(6)
plt.plot(history_d["train_errors"], label="Train")
plt.plot(history_d["test_errors"], label="Test")
plt.xlabel("Epochs")
plt.ylabel("Misclassification Error")
plt.title('7d2')
plt.legend()
plt.show()
```

Administrative

A8.

a. *[2 points]* About how many hours did you spend on this homework? There is no right or wrong answer :)
I dropped this class before, this's my second time doing this homework.
The first time I spend about 30-35 hours on this HW
This time I spend about 8-10 hours.