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]* Explain why a L1 norm penalty is more likely to result in sparsity (a larger number of 0s) in the weight vector, as compared to the L2 norm.

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

c. *[2 points]* True or False: If the step-size for gradient descent is too large, it may not converge.

d. *[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.

e. *[2 points]* Why is it necessary to apply the gradient descent algorithm on logistic regression but not linear regression?

What to Submit:

- **Part c:** True or False.
- **Parts a-e:** Brief (2-3 sentence) explanation.

a. With L1 regression there is a tendency to shrink some of the weights towards zero because the L1 normn ipenalizes the absolute value of coefficients. The L2 norm tends to shrink all of the coefficients uniformly without setting them at zero because the L2 norm penalizes the aquare of coefficents. These differences cause a difference in shape where the L1 will be more "spiky" and the L2 more ball-shaped. The shape of the L1 is more likely to introduce sparsity because of the abnormal shape of the function

b. Downside is that is not a convex function. The upside is that it will have a more aggressive penalty than the L1 norm for small weights driving more of them to zero producing an even sparse solution

c. True. If the step size is too large wit is possible that it will skip the minimum. It could potentially cause it to skip pass the minimum and fail to find a solution

d. SGD takes significantly shorter amount of time because it is much less computationally expensive. A disadvantage odf SGD to GD is that can possibly take more steps to converfe to SGD than GD.

e. linear regression has a convex cost function that can be minimalized using best fit methods, but GD algorithms can be applied. Where logistic regression requires a non-linear transformation resulting in a non closed-form solution.

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:

- **Parts a, b:** Proof.

a.

i (non-negativity):

show $|)x|)\geq 0$ for all $x \in R^n$

Because $f(x)=\sum_{i=1}^{n} |x_i)$ the function is a sum of $|x_i)$ which are all non-zero. Therfore because the function is a sum of non-zero numbers $f(x)=0$ if and only if all values of x are = 0.

ii (absolute scalability):

prove $|)a\,x|)=|a)|x)$ for all $a \in R$ and $x \in R^n$

say $f(a\,x)=\sum_{i=1}^{n} |)a\,x_i|)=\sum_{i=1}^{n} |a)|x_i)=|a)\sum_{i=1}^{n} |x_i)=a\,f(x)$ because $f(x)=\sum_{i=1}^{n} |x_i)$. Therefore it has absolute scalability

iii (triangle inequality):

Prove $|)x+y\,|)\leq|)x|)+|)\,y|)$ for all $x\,, y \in R^n$

starting at the hint show $|a+b)\leq|a)+|b)$ for all cases

$a\geq 0$ and $b\geq 0$: $|a+b)\leq|a)+|b)$

$a\leq 0$ and $b\leq 0$: $|-a-b)\leq|-a)+|-b)=a+b \vee$ ¿

$a\leq 0$ and $b\geq 0$: $|-a+b)\leq|-a)+|b)=a+b$

$a\geq 0$ and $b\leq 0$: $|a-b)\leq|a)+|-b)=a+b$

now with the above $f(x+y) = \sum^{n}*{i=1} //x_i+y_i//|leq{|sum^{n}*}{i=1} |x_i|+|y_i|} =\sum^{n} *{i=1}|x_i| + |sum^{n}*}{i=1}|y_i| $ => because this is of the same form as the proof above

$|)x+y|)\leq|)x|)+|)y|)$ for all of $x, y \in (R^n)$

b.

prove $g(x)=\left(\sum_{i=1}^{n}|x_i)^{0.5}\right)^2$ is not a norm

n = 2 dimensions

Take two vectors x = (0,1 and y = (1,0)

We will utilize the hint and try to prove it is not a norm using the traingle inequality property.

If the function was a norm then for the triangle inequality it would happen that

$g(x+y)\leq g(x)+g(y)$

Taking the two poaints above:

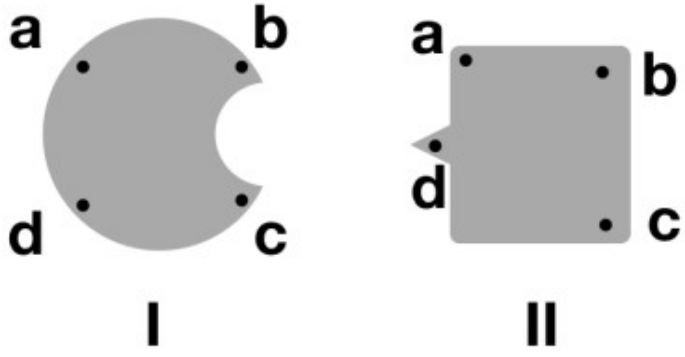
$g(x)=(|0)^{0.5}+|1)^{0.5})^2=(0+1)^2=1$

$g(y)=(|1)^{0.5}+|0)^{0.5})^2=(1+0)^2=1$

$g(x+y)=g(1,1)=¿$

therefore $g(x+y)\geq g(x)+g(y)$ in this case and fails the triangle inequality and g(x) is not a norm

A3. *[2 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-II), 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:

- **Parts I, II:** 1-2 sentence explanation of why the set is convex or not.

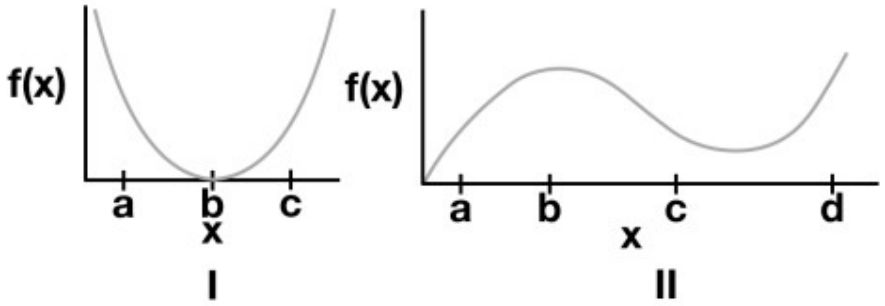
I: for the grey shaded sets of I it is convex because there are no points (b,c,d) that when connected to a would leave the shaded region

II: The set is not convex because while [a,b] and [a,c] do not leave the shaded region [a,d] does and the function is not convex

A4. *[2 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-II), 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 $[a, d]$



What to Submit:

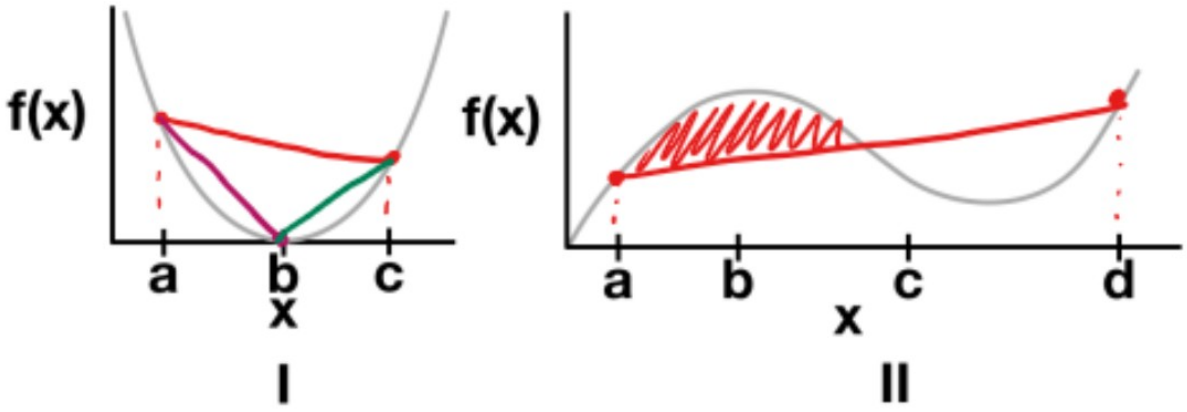
- **Parts a, b:** 1-2 sentence explanation of why the function is convex or not.

a:

this is a convex function. If we connect any two points [a,b], [a,c], [b,c] we do not leave the graph and the line lies abbove the function over the entire interval

b:

this is not a convext function on [a,d]. As a counterexample we can take [a,d] which does leave the graph and at point a to c it lies under the funciton



B1. Use just the definitions above and let $\|\cdot\|$ be a norm.

a. *[3 points]* Show that $f(x) = \|x\|$ is a convex function.

b. *[3 points]* Show that $\{x \in \mathbb{R}^n : \|x\| \leq 1\}$ is a convex set.

c. *[2 points]* Draw a picture of the set $\{(x_1, x_2) : g(x_1, x_2) \leq 4\}$ where $g(x_1, x_2) = \left(|x_1|^{1/2} + |x_2|^{1/2}\right)^2$. (This is the function considered in 1b above specialized to $n = 2$.) We know g is not a norm. Is the defined set convex? Why not?

Context: It is a fact that a function f defined over a set $A \subseteq \mathbb{R}^n$ is convex if and only if the set $\{(x, z) \in \mathbb{R}^{n+1} : z \geq f(x), x \in A\}$ is convex. Draw a picture of this for yourself to be sure you understand it.

What to Submit:

- **Parts a, b:** Proof.
- **Part c:** A picture of the set, and 1-2 sentence explanation.

a:

to prove if the function is convex we will utilize the third (triangle identity) of the function to prove if it is convex or not because it is ||.|| a norm. We can assume that it is non-negative and scalable. If it is convex, the convex function f for any $x, y \in R^n$ and $\lambda \in [0,1)$

iii (triangle inequality):

trtiangle inequality: $|)x+y|)\leq|)x|)+|)y|)$ for all $x, y \in R^n$

As stated above: $f: R^d = ¿ R$ is convex on set A if $f(\lambda x+(1-\lambda)y) \leq \lambda f(x)+(1-\lambda)f(y)$ for all $x, y \in A$ and $\lambda \in [0,1)$ by applying the trangle inequality we solve:

$$f(\lambda x+(1-\lambda)y)=|)\lambda x+(1-\lambda)y|)\leq|)\lambda x|)+|)(1-\lambda)y|)=\lambda|)x|)+(1-\lambda)|)y|) ¿= \lambda f(x)(1-\lambda)f(y)$$

so it is in fact convex the function f for any $x, y \in S$ and $\lambda \in [0,1)$

b:

Show that the set S = $x \in R^n : ||x|| \leq 1$ is convex

Because $||x|| \leq 1$:

$||x|| \leq 1 \text{ and } ||y|| \leq 1$

assume x=y=1:
$||\lambda x + (1-\lambda) y|| \leq ||\lambda x|| + ||(-1-\lambda) y|| = \lambda ||x|| + (1-\lambda) ||1|| = \lambda(1) + (1-\lambda)(1) = \lambda - \lambda + 1 = 1$

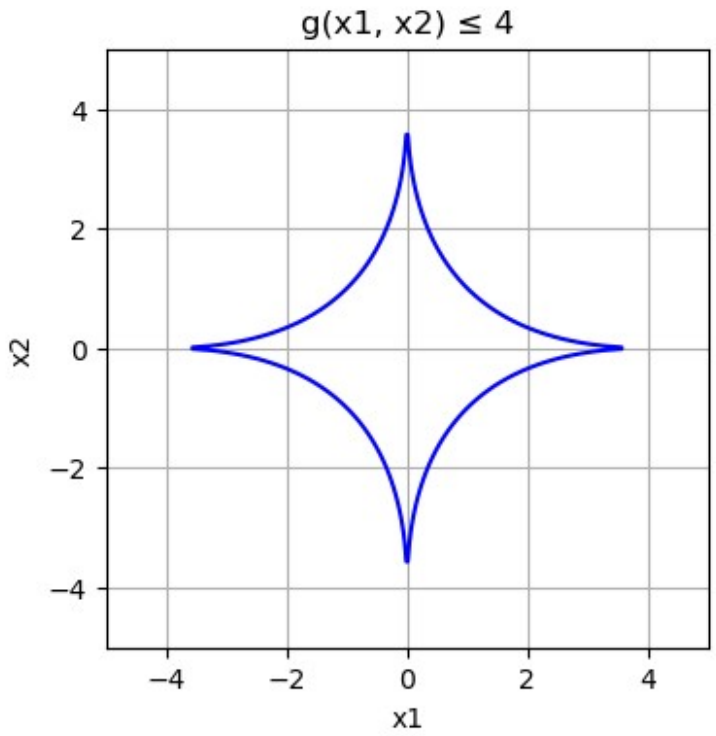
So the function contained in the set is in fact convex

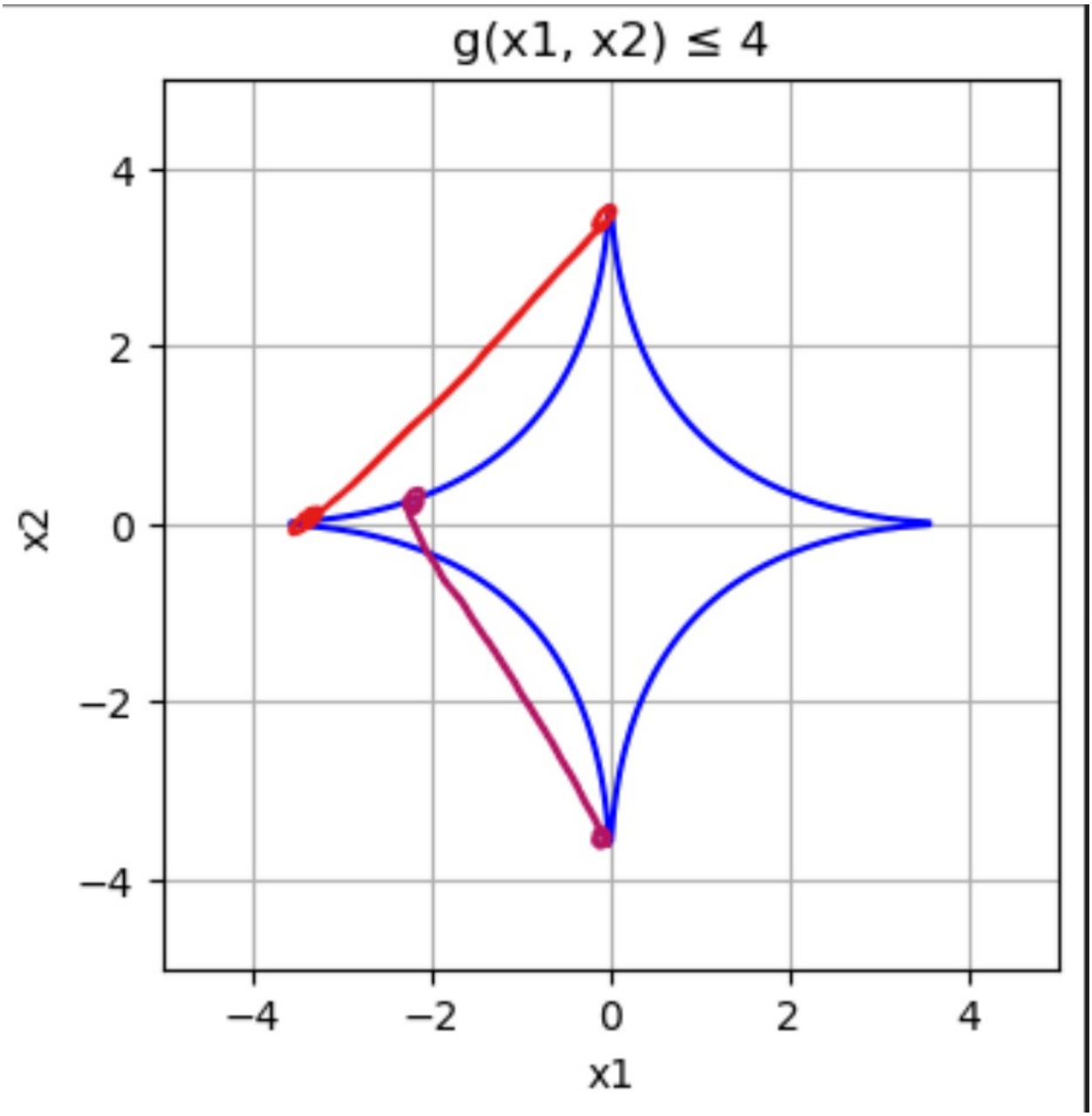
```
import numpy as np
import matplotlib.pyplot as plt


def g(x1, x2):
    return (np.abs(x1)**0.5 + np.abs(x2)**0.5)**2

x1 = np.linspace(-5, 5, 400)
x2 = np.linspace(-5, 5, 400)
x1, x2 = np.meshgrid(x1, x2)
f = g(x1, x2)

plt.figure(figsize=(4, 4))
contour = plt.contour(x1, x2, z, levels=[4], colors='blue')
plt.title('g(x1, x2) ≤ 4')
plt.xlabel('x1')
plt.ylabel('x2')
plt.grid(True)
plt.show()
```


g(x1, x2) ≤ 4
4
2
0
−2
−4
x2
−4
−2
0
2
4
x1



There are multiple sets pf two points that when connecting them exit the function as seen above therefore the function is not convex

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.

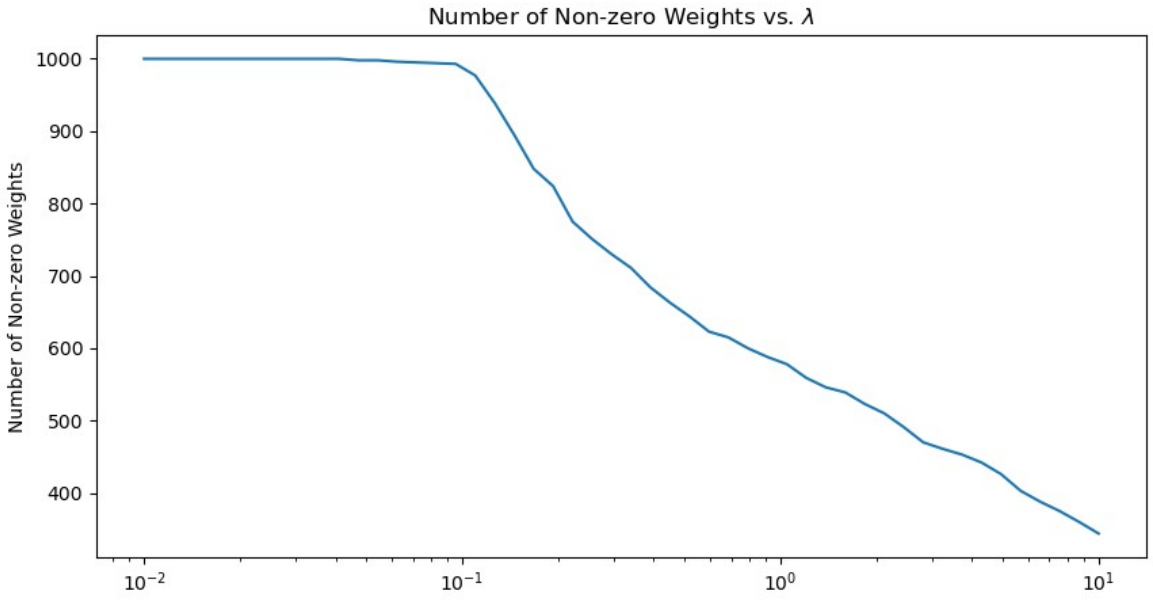
   Note that in an ideal situation we would have an (FDR,TPR) pair in the upper left corner. We can always trivially achieve $(0, 0)$ and $(\frac{d-k}{d}, 1)$.

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

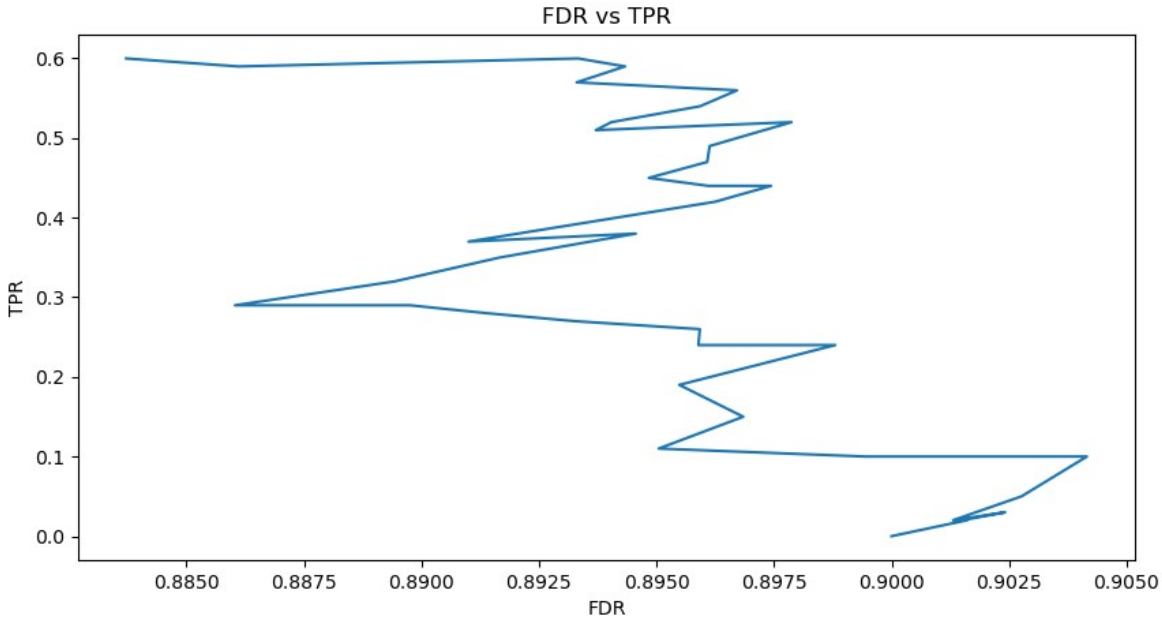
What to Submit:

- **Part a:** Plot 1.
- **Part b:** Plot 2.
- **Part c:** 1-2 sentence explanation.
- **Code** on Gradescope through coding submission

a:



b:



c:

plot I: As the λ there is a stronger penalty is imposed and the more non-zero weights goes down.

plotII: As λ increases the model ecomes more selective increasing the FDR and lowering the TPR if too high because it is overlooking potentially important features.

d (code):

```
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, eta: float
) -> Tuple[np.ndarray, float]:
    """Single step in ISTA algorithm.
    It should update every entry in weight, and then return an updated
version of weight along with calculated bias on input weight!

    Args:
        X (np.ndarray): An (n x d) matrix, with n observations each
with d features.
        y (np.ndarray): An (n, ) array, with n observations of
targets.
        weight (np.ndarray): An (d,) array. Weight returned from the
step before.
        bias (float): Bias returned from the step before.
        _lambda (float): Regularization constant. Determines when
weight is updated to 0, and when to other values.
        eta (float): Step-size. Determines how far the ISTA iteration
moves for each step.

    Returns:
        Tuple[np.ndarray, float]: Tuple with 2 entries. First
represents updated weight vector, second represents bias.

    """
    # set new bias
    bdash=bias-2*eta*np.sum(np.dot(X,weight)+bias-y)
    #set new weight
    wdash=weight-2*eta*np.dot(X.T, np.dot(X,weight)+bias-y)
    # apply threshold per algorithm
    """
    for k in range (len(wdash)):
        thresh = 2*eta*_lambda
        if wdash[k] < -thresh:
            wdash[k]+=thresh
        elif wdash[k] > thresh:
            wdash[k]-=thresh
        else:
```

```
            wdash[k]=0"""
    #should accomplish the same as above but without a for loop
    wdash = np.sign(wdash)*np.maximum(np.abs(wdash)-2*eta*_lambda,0)

    return wdash,bdash


@problem.tag("hw2-A")
def loss(
    X: np.ndarray, y: np.ndarray, weight: np.ndarray, bias: float,
_lambda: float
) -> float:
    """L-1 (Lasso) regularized SSE loss.

    Args:
        X (np.ndarray): An (n x d) matrix, with n observations each
with d features.
        y (np.ndarray): An (n, ) array, with n observations of
targets.
        weight (np.ndarray): An (d,) array. Currently predicted
weights.
        bias (float): Currently predicted bias.
        _lambda (float): Regularization constant. Should be used along
with L1 norm of weight.

    Returns:
        float: value of the loss function
    """
    # compute predicted values
    prediction = np.dot(X,weight)+bias
    #calculate the residual
    leftover = prediction-y
    #calculate sum of squared residuals
    sseloss = np.sum(leftover**2)
    #compute regularization term
    l1 = _lambda*np.sum(np.abs(weight))
    #calculate and return the total loss
    return sseloss+l1


@problem.tag("hw2-A", start_line=5)
def train(
    X: np.ndarray,
    y: np.ndarray,
    _lambda: float = 0.01,
    eta: float = 0.00001,
    convergence_delta: float = 1e-4,
    start_weight: np.ndarray = None,
    start_bias: float = None
) -> Tuple[np.ndarray, float]:
```

```
    """Trains a model and returns predicted weight and bias.

    Args:
        X (np.ndarray): An (n x d) matrix, with n observations each
with d features.
        y (np.ndarray): An (n, ) array, with n observations of
targets.
        _lambda (float): Regularization constant. Should be used for
both step and loss.
        eta (float): Step size.
        convergence_delta (float, optional): Defines when to stop
training algorithm.
            The smaller the value the longer algorithm will train.
            Defaults to 1e-4.
        start_weight (np.ndarray, optional): Weight for hot-starting
model.
            If None, defaults to array of zeros. Defaults to None.
            It can be useful when testing for multiple values of
lambda.
        start_bias (np.ndarray, optional): Bias for hot-starting
model.
            If None, defaults to zero. Defaults to None.
            It can be useful when testing for multiple values of
lambda.

    Returns:
        Tuple[np.ndarray, float]: A tuple with first item being array
of shape (d,) representing predicted weights,
            and second item being a float representing the bias.

    Note:
        - You will have to keep an old copy of weights for convergence
criterion function.
            Please use `np.copy(...)` function, since numpy might
sometimes copy by reference,
            instead of by value leading to bugs.
        - You might wonder why do we also return bias here, if we
don't need it for this problem.
            There are two reasons for it:
                - Model is fully specified only with bias and weight.
                    Otherwise you would not be able to make
predictions.
                    Training function that does not return a fully
usable model is just weird.
                - You will use bias in next problem.
    """
    if start_weight is None:
        weight = np.zeros(X.shape[1])
        bias = 0
    else:
```

```
        weight = np.copy(start_weight)
        bias = start_bias

    old_w = np.copy(weight)
    old_b = bias

    #run for set number of iteration
    #max_iteration = 1000
    #for i in range(max_iteration):
    #run until done
    while True:
        new_weight, new_bias = step(X,y,weight,bias,_lambda,eta)
        #check if converged. if yes break
        if
convergence_criterion(new_weight,old_w,new_bias,old_b,convergence_delt
a):
            break
        #update new and old weights and bias
        weight, bias = new_weight, new_bias
        old_w, old_b = np.copy(weight),bias
    return weight, bias


@problem.tag("hw2-A")
def convergence_criterion(
    weight: np.ndarray, old_w: np.ndarray, bias: float, old_b: float,
convergence_delta: float
) -> bool:
    """Function determining whether weight and bias has converged or
not.
    It should calculate the maximum absolute change between weight and
old_w vector, and compare it to convergence delta.
    It should also calculate the maximum absolute change between the
bias and old_b, and compare it to convergence delta.

    Args:
        weight (np.ndarray): Weight from current iteration of gradient
descent.
        old_w (np.ndarray): Weight from previous iteration of gradient
descent.
        bias (float): Bias from current iteration of gradient descent.
        old_b (float): Bias from previous iteration of gradient
descent.
        convergence_delta (float): Aggressiveness of the check.

    Returns:
        bool: False, if weight and bias has not converged yet. True
otherwise.
    """
    return np.max(np.abs(weight-old_w))<convergence_delta and
```

```
np.abs(bias-old_b) < convergence_delta


@problem.tag("hw2-A")
def main():
    """
    Use all of the functions above to make plots.
    """
    n=500
    d=1000
    k=100
    sigma=1

    #create the synthetic data
    X=np.random.randn(n,d)
    noise=np.random.normal(0,sigma**2,size=n)
    real_weight=np.array([1.0/(j+1)if j < k else 0 for j in range(d)])
    y = X@real_weight+noise

    lamb = np.logspace(np.log10(10),np.log10(0.01),num=50)
    num_non_zero = []
    fdr_val = []
    tpr_val = []

    for _lambda in lamb:
        #train the synthetic data
        weight,bias = train(X,y,_lambda,eta=1e-4,convergence_delta=1e-
4)
        #
        nonzero_index = np.nonzero(weight)[0]
        #
        num_non_zero.append(len(nonzero_index))
        #calculate fdr and update lists
        false_pos=np.sum(np.logical_and(weight != 0, real_weight ==
0))
        fdr=false_pos/np.maximum(len(nonzero_index),1)
        fdr_val.append(fdr)
        #calculate tpr
        true_pos=np.sum(np.logical_and(weight == 0, real_weight != 0))
        tpr=true_pos/np.maximum(np.sum(real_weight!=0),1)
        tpr_val.append(tpr)

    #part a graph
    plt.figure(figsize=(10,5))
    plt.plot(lamb, num_non_zero)
    plt.xscale('log')
    plt.xlabel('$\lambda$')
    plt.ylabel('Number of Non-zero Weights')
    plt.title('Number of Non-zero Weights vs. $\lambda$')
```

```
    #part b graph
    plt.figure(figsize=(10,5))
    plt.plot(fdr_val,tpr_val)
    plt.xlabel('FDR')
    plt.ylabel('TPR')
    plt.title('FDR vs TPR')

    plt.tight_layout
    plt.show()

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

A6. We'll now put the Lasso to work on some real data in `crime_data_lasso.py`. We have read in the data for you with the following:

```
df_train, df_test = load_dataset("crime")
```

This stores the data as Pandas `DataFrame` objects. `DataFrames` are similar to Numpy `arrays` but more flexible; unlike `arrays`, `DataFrames` 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 ISTA Lasso algorithm implemented in the previous problem.

a. *[4 points]* Read the documentation for the originalcversion 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. Provide a short (1-3 sentence) explanation.

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. Provide a short (1-3 sentence) explanation.

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 $\hat{w}_0$ and $\hat{b}_0$ as the $\hat{w}$ and $\hat{b}$ found at the end of your previous iteration. Continue the process 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 mean 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. Retrain and inspect the weights $\hat{w}$ for $\lambda = 30$ and for *all* input variables. 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:

- **Parts a, b:** 1-2 sentence explanation.
- **Part c:** Plot 1.
- **Part d:** Plot 2.
- **Part e:** Plot 3.
- **Parts f, g:** Answers and 1-2 sentence explanation.
- **Code** on Gradescope through coding submission.

a:

1. MedRent: median gross rent
2. medIncomr: median hosuehold income
3. NumImmig: total number of people to be foreign born

Both policies that have been put into place such as zoning laws as well as not put into place, rent control, will affect the model. Many of these policies have been put into place creating neighborhoods of high and low median income, one of the primary drivers is the low rent. This often creates clusters of very poor, often minority groups and immigrants who end up in the same area. There are a lot of factors that go into this from this point but to point out a few would be, that the low rent areas often have a lower income => a lower tax bracket => less money for schools => less educated => less likely to be able to leave or support themselves =>> higher crime rates. In fact there has been a lot that has come out about the systemic racism and classist policies that have caused this and many other things, as this is just one efect, that have created poor communities which have higher rates of crime

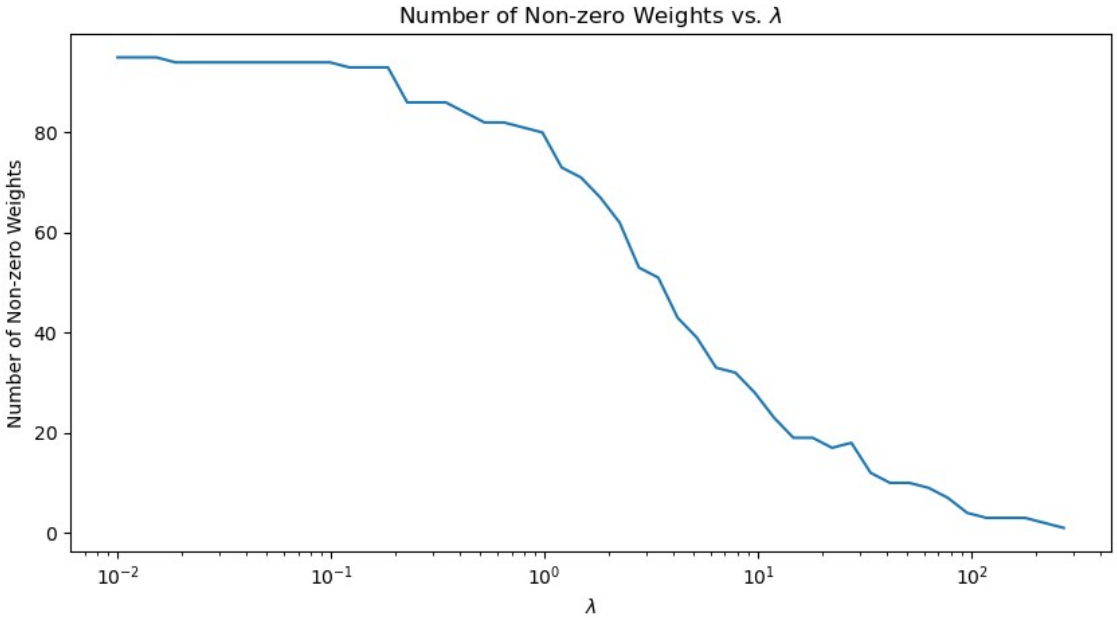
b:

1. HousVacant: number of vacant households (people could be vacating because of the crime and vacant houses could not be the casue of crime but a result or it could be that

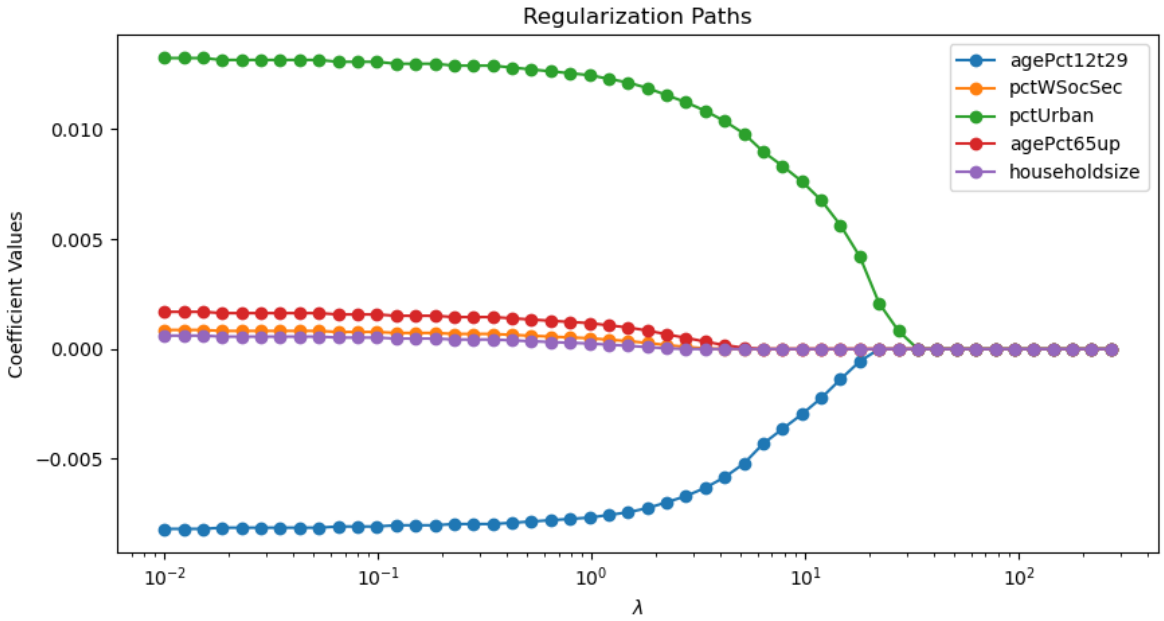
the vacancy of the houses leaves places for crimes to be committed increasing the crime rate)

2. LemasGangUnitDeploy: gang unit deployed (This is a response tactict crime and => should not be causing it, but there may be correlation as by deploying them they are much more likely to interact with crime)
3. LemasSwFTFieldPerPop: sworn full time police officers in field operations (on the street as opposed to administrative etc) per 100K population (it is possible that the number of field operatives is a result of increased crime reates and => an indicator instead of a cuase but could also be seen as having higher police in the field they will "catch" more crime happening as there are more of them)

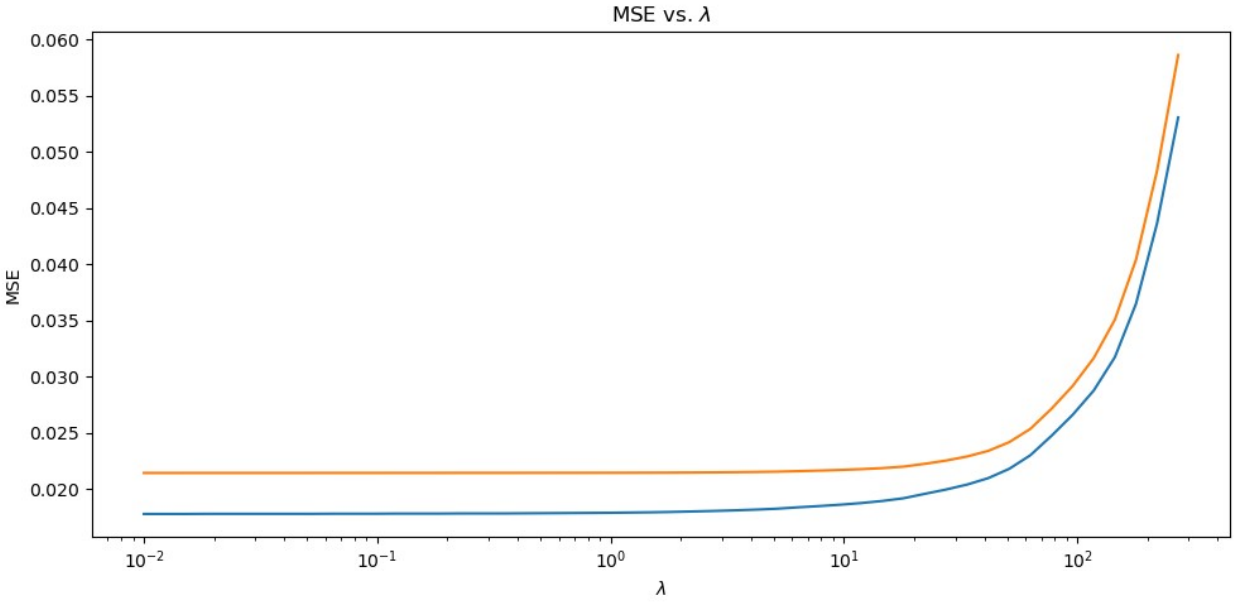
c:



d:



e:



f:

Max positive weight: 0.06289995848264814, Feature: PctIlleg Max negative weight: -0.04020784150209127, Feature: PctKids2Pa

According to the above the feature that PctIlleg: percentage of kids born to never married had the strongest correlation to an increase in crime rates. I.e. the more children were born to non-

married kids it caused crime rates to go up. On the other hand PctKids2Par: percentage of kids in family housing with two parents had the strongest inverse relation to crime rate. I.e. the more kids who had two parent the crime rate fell.

Because these feattures are inverse both in the data but also of eachother it reasons that these are related to eachother and do affect the crime rater

g:

This is a statistical flaw because they are assuming causation not corelation. Whil in the case of the hint it is true that fire trucks are often seen around burning buildings, they did not cause the fires, but instead came as a result of the fire (correlation). Unless in isolated incidents where a/few of the fireman are arsonists. ("fun" fact: It has been reported that roughly 100 U.S. firefighters are convicted of arson each year).

However in this case encouraging 65+ year olds to move into a high crime area would most likely have a negative effect. While it is true 65+ majority communities have much lower crime, that is a correlation of the crime as a predominant amount of the factors that actually cause the crime are missing and introducing a lot of 65+ year olds into a high crime area would only present vulnerable victims for crime. This may actually result in anb increase in crime.

code:

```
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


@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")
    #get the test data for X
    X_dftrain = df_train.drop("ViolentCrimesPerPop", axis=1)
    X_train = X_dftrain.values
    X_dftest = df_test.drop("ViolentCrimesPerPop", axis=1)
    X_test = X_dftest.values
    #get the test data for y
    y_dftrain = df_train["ViolentCrimesPerPop"]
    y_train = y_dftrain.values
    y_dftest = df_test["ViolentCrimesPerPop"]
    y_test = y_dftest.values
```

```
    # solve for lambda max
    lambda_max = np.max(np.abs(X_train.T @ (y_train -
np.mean(y_train))))
    lambda_min, num_lambdas = 0.01,50
    lamb =
np.logspace(np.log10(lambda_max),np.log10(lambda_min),num=num_lambdas)

    num_non_zero = []
    # add for plottin MSE
    mse_train = []
    mse_test = []

    #set weight and bias to zero
    weight = np.zeros(X_train.shape[1])
    bias = 0

    #get the specified features indexes
    all_other_features = df_train.columns.drop("ViolentCrimesPerPop")
    feat_vars = ['agePct12t29','pctWSocSec', 'pctUrban', 'agePct65up',
'householdsize']
    for i in all_other_features:
        feat_index = [list(all_other_features).index(i)]

    #add trackers for specified features for plotting
    feat_paths = {feat: [] for feat in feat_vars}

    #find max pos and neg weights with lambda = 30
    weight_2,bias_2 =
train(X_train,y_train,_lambda=30,eta=0.00001,convergence_delta=1e-
4,start_weight=weight, start_bias=bias)
    max_pos_weight = np.max(weight_2)
    max_neg_weight = np.min(weight_2)
    print(f"Max positive weight: {max_pos_weight}, Feature:
{all_other_features[np.argmax(weight_2)]}")
    print(f"Max negative weight: {max_neg_weight}, Feature:
{all_other_features[np.argmin(weight_2)]}")

    for _lambda in lamb:
        #train the data
        weight,bias =
train(X_train,y_train,_lambda,eta=0.00001,convergence_delta=1e-
4,start_weight=weight, start_bias=bias)
        #identify non-zero elements
        nonzero_index = np.nonzero(weight)[0]
        #count the number of nonzero elements
        num_non_zero.append(np.count_nonzero(weight))

        #calculate the MSE
        mse_train.append(np.mean((X_train@weight+bias-y_train)**2))
        mse_test.append(np.mean((X_test@weight+bias-y_test)**2))
```

```
        #track the weights for the directed features
        for feat in feat_vars:
            index = list(all_other_features).index(feat)
            feat_paths[feat].append(weight[index])


    #part c graph
    plt.figure(figsize=(10,5))
    plt.plot(lamb, num_non_zero)
    plt.xscale('log')
    plt.xlabel('$\lambda$')
    plt.ylabel('Number of Non-zero Weights')
    plt.title('Number of Non-zero Weights vs. $\lambda$')

    #part d graph
    plt.figure(figsize=(10,5))
    for feat, path in feat_paths.items():
        plt.plot(lamb, path, '-o', label=feat)
    plt.xscale('log')
    plt.xlabel('$\lambda$')
    plt.ylabel('Coefficient Values')
    plt.legend()
    plt.title('Regularization Paths')

    #part e graph
    plt.figure(figsize=(10,5))
    plt.plot(lamb, mse_train, label='MSE Train')
    plt.plot(lamb, mse_test, label='MSE Test')
    plt.xscale('log')
    plt.xlabel('$\lambda$')
    plt.ylabel('MSE')
    plt.title('MSE vs. $\lambda$')


    plt.tight_layout()
    plt.show()

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

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.

   (i) 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).

   (ii) 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 mini-batch 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:** Proof
- **Part b:** Separate plots for b(i) and b(ii).
- **Part c:** Separate plots for c which reproduce those from b(i) and b(ii) for this case.
- **Part d:** Separate plots for c which reproduce those from b(i) and b(ii) for this case.
- **Code** on Gradescope through coding submission.

a:

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

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

$$\lambda=10^{-1}$$

$$x_i \in R^d$$

$$y_i \in [-1, 1)$$

Derive $\nabla_w J(w,b)$ and $\nabla_b J(w,b)$ give answer in terms of $\mu_i (w,b) $

regularization term: $\nabla_w J(w,b) = \frac{d}{dw} \lambda ||w||_2^2 = 2\lambda w$

loss term: $\nabla_w J(w,b) = \frac{d}{dw} \frac{1}{n} \sum_{i=1}^{n} log\left(1+\exp\left(-y_i\left(b+x_i^T w\right)\right)\right) = \frac{1}{n} \sum_{i=1}^{n}$ ¿¿

because

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

we can rewrite the equation as:

$\nabla_w J(w,b) = \frac{1}{n} \sum^n_{i=1} -y_i x_i(1-\mu_i (w,b)) $

Putting both of them together to get

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

Following the same outline as before with the gradient wrt b: regularization term:

$$\nabla_b J(w,b) = \frac{d}{dw} \lambda ||w||_2^2 = 0$$

loss term: $\nabla_b J(w,b) = \frac{d}{dw} \frac{1}{n} \sum^n_{i=1} log(1+\exp{(-y_i(b+x^T_i w))}) = \frac{1}{n} \sum^n_{i=1} (\frac{-y_i \exp(-y_i(b+x^T_i w))}{1+ \exp(-y_i (b+x^T_i w))} = \frac{1}{n} \sum^n_{i=1} (\frac{-y_i }{1+ \exp(-y_i (b+x^T_i w))}) $

because

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

we can rewrite the equation as:

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

Putting both of them together to get

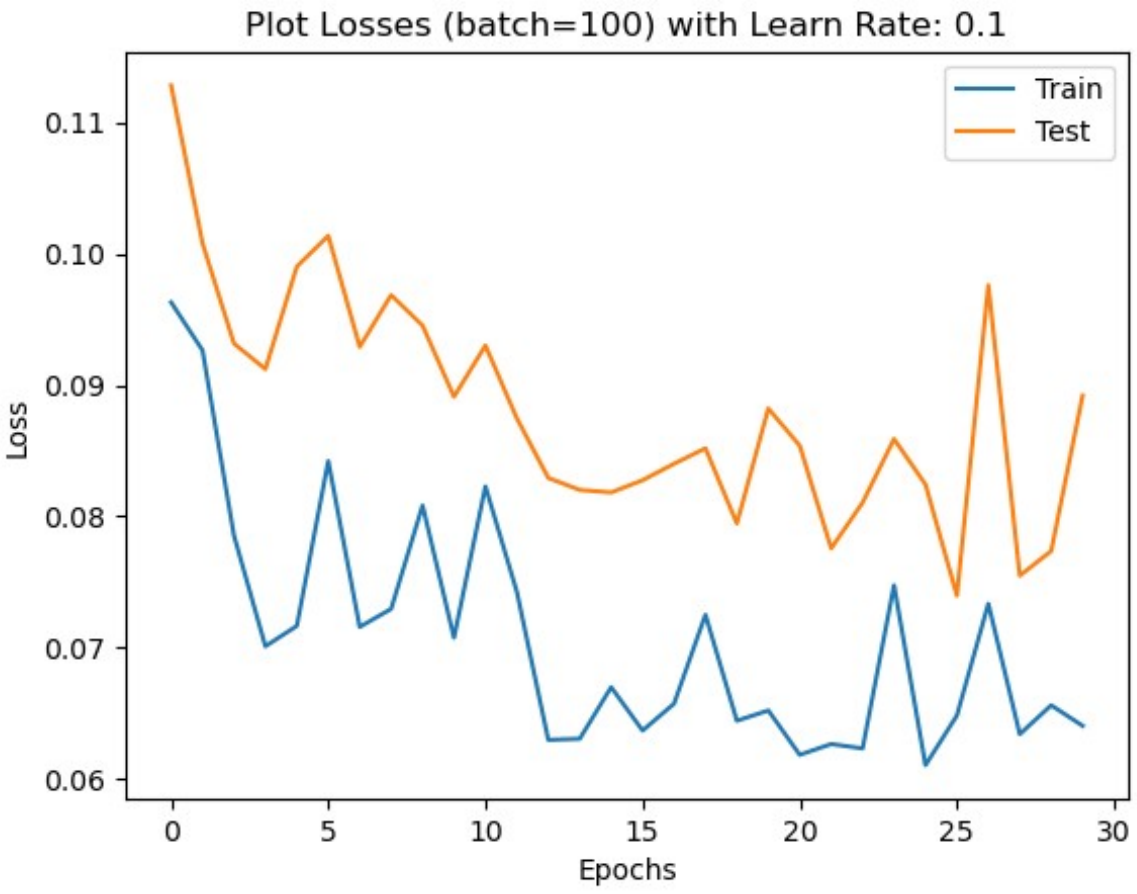
$$\nabla_b J(w,b) = \frac{1}{n}\sum_{i=1}^{n} -y_i\left(1-\mu_i(w,b)\right)$$

b:

Learn Rate:

Best Learning Rate for GD is: 0.1 with a loss of: 0.06119169226848921

i:



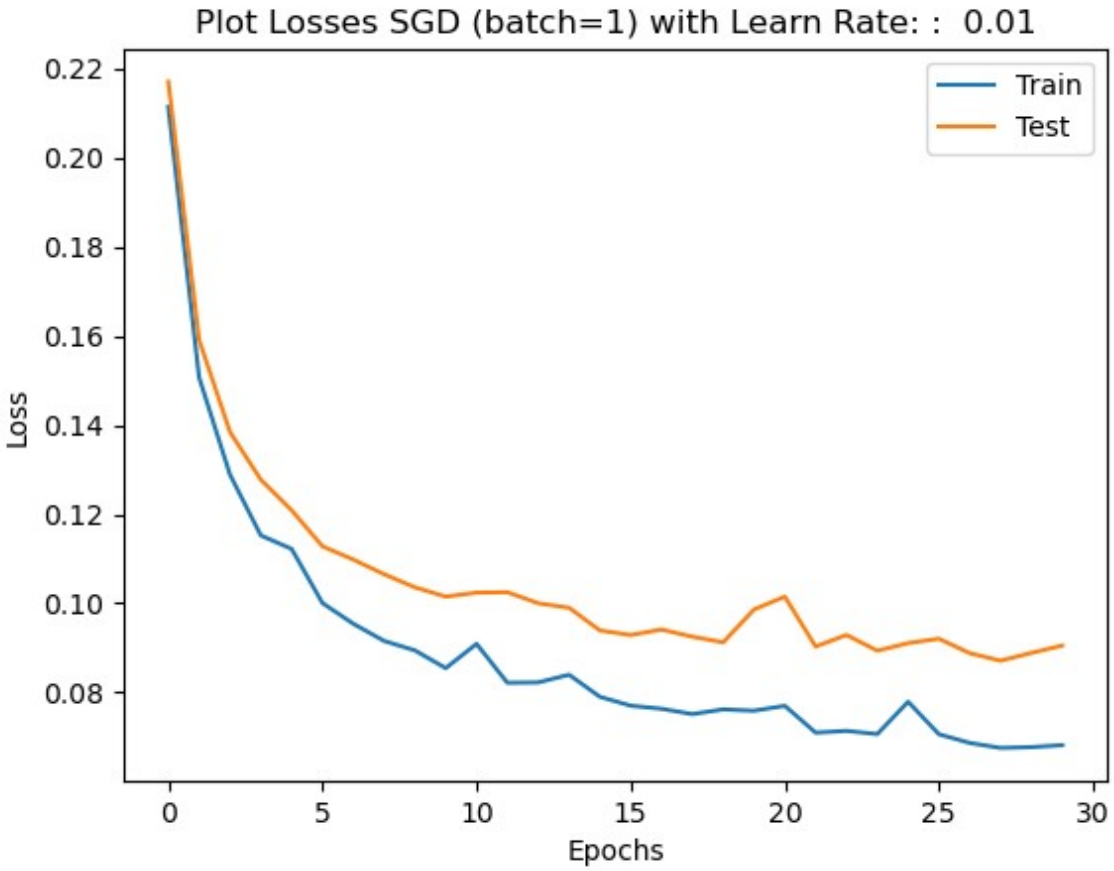
ii:

c:

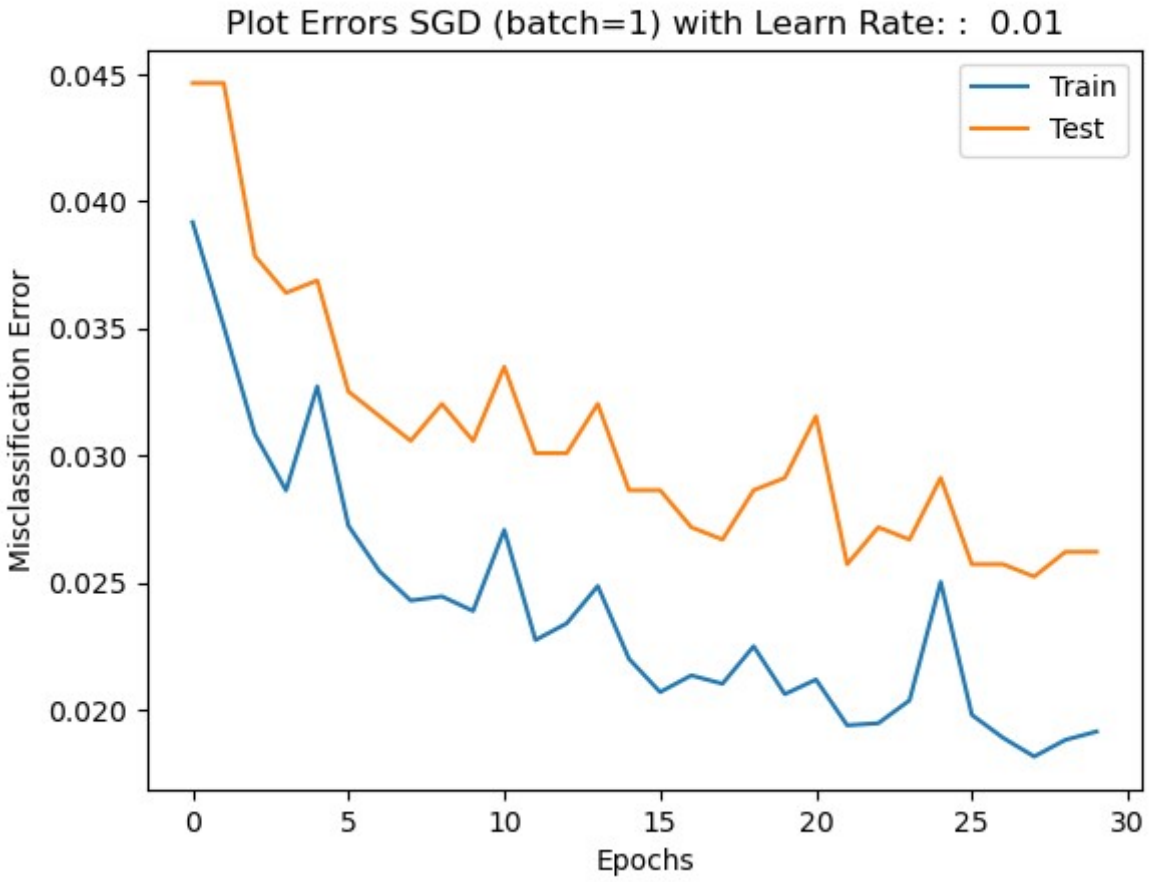
Learn Rate:

Best Learning Rate for SGD is: 0.01 with a loss of: 0.06807813094355825

i:


Plot Losses SGD (batch=1) with Learn Rate: : 0.01
Train
Test
Loss
Epochs
0.22
0.20
0.18
0.16
0.14
0.12
0.10
0.08
0
5
10
15
20
25
30

ii:



d:

Learn Rate:

Best Learning Rate for Mini-Batch GD is: 0.1 with a loss of: 0.06715990848968376

i:

ii:

```
from typing import Dict, List, Tuple

import matplotlib.pyplot as plt
import numpy as np

from utils import load_dataset, problem
#from homeworks.log_regression.binary_log_regression import RNG

# 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:
    """
```

```
    Loads MNIST data, extracts only examples with 2, 7 as labels, and
converts them into -1, 1 labels, respectively.

    Returns:
        Dataset: 2 tuples of numpy arrays, each containing examples
and labels.
            First tuple is for training, while second is for testing.
            Shapes as as follows: ((n, d), (n,)), ((m, d), (m,))
    """
    (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):
        """Initializes the Binary Log Regression model.
        Args:
            _lambda (float, optional): Ridge Regularization
coefficient. Defaults to 1e-3.
        """
        self._lambda: float = _lambda
        # Fill in with matrix with the correct shape
        self.weight: np.ndarray = np.zeros(784)  # type: ignore
        self.bias: float = 0.0


    @problem.tag("hw2-A")
    def mu(self, X: np.ndarray, y: np.ndarray) -> np.ndarray:
        """Calculate mu in vectorized form, as described in the
problem.
        The equation for i^th element of vector mu is given by:

        $$ \mu_i = 1 / (1 + \exp(-y_i (bias + x_i^T weight))) $$

        Args:
            X (np.ndarray): observations represented as `(n, d)`
matrix.
                n is number of observations, d is number of features.
                d = 784 in case of MNIST.
```

```
            y (np.ndarray): targets represented as `(n, )` vector.
                n is number of observations.

        Returns:
            np.ndarray: An `(n, )` vector containing mu_i for i^th
element.
        """
        return 1/(1+np.exp(-y*(self.bias+X.dot(self.weight))))

    @problem.tag("hw2-A")
    def loss(self, X: np.ndarray, y: np.ndarray) -> float:
        """Calculate loss J as defined in the problem.

        Args:
            X (np.ndarray): observations represented as `(n, d)`
matrix.
                n is number of observations, d is number of features.
                d = 784 in case of MNIST.
            y (np.ndarray): targets represented as `(n, )` vector.
                n is number of observations.

        Returns:
            float: Loss given X, y, self.weight, self.bias and
self._lambda
        """
        log_lik_mu = -np.mean(np.log(self.mu(X,y)))
        reg = self._lambda*np.sum(self.weight**2)
        return reg+log_lik_mu

    @problem.tag("hw2-A")
    def gradient_J_weight(self, X: np.ndarray, y: np.ndarray) ->
np.ndarray:
        """Calculate gradient of loss J with respect to weight.

        Args:
            X (np.ndarray): observations represented as `(n, d)`
matrix.
                n is number of observations, d is number of features.
                d = 784 in case of MNIST.
            y (np.ndarray): targets represented as `(n, )` vector.
                n is number of observations.
        Returns:
            np.ndarray: An `(d, )` vector which represents gradient of
loss J with respect to self.weight.
        """
        return (-np.mean(y*(X.T*(1-(self.mu(X,y)))),axis = 1) +
2*self._lambda*self.weight)

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

```
        """Calculate gradient of loss J with respect to bias.

        Args:
            X (np.ndarray): observations represented as `(n, d)`
matrix.
                n is number of observations, d is number of features.
                d = 784 in case of MNIST.
            y (np.ndarray): targets represented as `(n, )` vector.
                n is number of observations.

        Returns:
            float: A number that represents gradient of loss J with
respect to self.bias.
        """
        return (-np.mean(y*(1-(self.mu(X,y)))))

    @problem.tag("hw2-A")
    def predict(self, X: np.ndarray) -> np.ndarray:
        """Given X, weight and bias predict values of y.

        Args:
            X (np.ndarray): observations represented as `(n, d)`
matrix.
                n is number of observations, d is number of features.
                d = 784 in case of MNIST.

        Returns:
            np.ndarray: An `(n, )` array of either -1s or 1s
representing guess for each observation.
        """
        return np.sign(X.dot(self.weight)+self.bias)

    @problem.tag("hw2-A")
    def misclassification_error(self, X: np.ndarray, y: np.ndarray) ->
float:
        """Calculates misclassification error (the rate at which this
model is making incorrect predictions of y).
        Note that `misclassification_error = 1 - accuracy`.

        Args:
            X (np.ndarray): observations represented as `(n, d)`
matrix.
                n is number of observations, d is number of features.
                d = 784 in case of MNIST.
            y (np.ndarray): targets represented as `(n, )` vector.
                n is number of observations.

        Returns:
            float: percentage of times prediction did not match
target, given an observation (i.e. misclassification error).
```

```
        """
        return np.mean(self.predict(X)!=y)

    @problem.tag("hw2-A")
    def step(self, X: np.ndarray, y: np.ndarray, learning_rate: float
= 1e-4):
        """Single step in training loop.
        It does not return anything but should update self.weight and
self.bias with correct values.

        Args:
            X (np.ndarray): observations represented as `(n, d)`
matrix.
                n is number of observations, d is number of features.
                d = 784 in case of MNIST.
            y (np.ndarray): targets represented as `(n, )` vector.
                n is number of observations.
            learning_rate (float, optional): Learning rate of SGD/GD
algorithm.
                Defaults to 1e-4.
        """
        self.weight -= learning_rate*self.gradient_J_weight(X,y)
        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]]:
        """Train function that given dataset X_train and y_train
adjusts weights and biases of this model.
        It also should calculate misclassification error and J loss at
the END of each epoch.

        For each epoch please call step function `num_batches` times
as defined on top of the starter code.

        NOTE: This function due to complexity and number of possible
implementations will not be publicly unit tested.
        However, we might still test it using gradescope, and you will
be graded based on the plots that are generated using this function.

        Args:
            X_train (np.ndarray): observations in training set
```

```
represented as `(n, d)` matrix.
                n is number of observations, d is number of features.
                d = 784 in case of MNIST.
            y_train (np.ndarray): targets in training set represented
as `(n, )` vector.
                n is number of observations.
            X_test (np.ndarray): observations in testing set
represented as `(m, d)` matrix.
                m is number of observations, d is number of features.
                d = 784 in case of MNIST.
            y_test (np.ndarray): targets in testing set represented as
`(m, )` vector.
                m is number of observations.
            learning_rate (float, optional): Learning rate of SGD/GD
algorithm. Defaults to 1e-2.
            epochs (int, optional): Number of epochs (loops through
the whole data) to train SGD/GD algorithm for.
                Defaults to 30.
            batch_size (int, optional): Number of observation/target
pairs to use for a single update.
                Defaults to 100.

        Returns:
            Dict[str, List[float]]: Dictionary containing 4 keys, each
pointing to a list/numpy array of length `epochs`:
            {
                "training_losses": [<Loss at the end of each epoch on
training set>],
                "training_errors": [<Misclassification error at the
end of each epoch on training set>],
                "testing_losses": [<Same as above but for testing
set>],
                "testing_errors": [<Same as above but for testing
set>],
            }
            Skeleton for this result is provided in the starter code.

        Note:
            - When shuffling batches/randomly choosing batches makes
sure you are using RNG variable defined on the top of the file.
        """
        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": [],
        }
        #shuffle the data with given RNG and set indices
```

```
        for i in range(epochs):
            ind=np.arange(X_train.shape[0])
            RNG.shuffle(ind)
            X_train_shuff = X_train[ind]
            y_train_shuff = y_train[ind]

            for j in range(num_batches):
                #create the batches with the shuffled data
                start = i*batch_size
                end = start+batch_size
                X_batch = X_train_shuff[start:end]
                y_batch = y_train_shuff[start:end]
                #step (learnign rate defaults to 1e-4)
                self.step(X_batch,y_batch,learning_rate)

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

result["train_errors"].append(self.misclassification_error(X_train,y_t
rain))
            result["test_losses"].append(self.loss(X_test,y_test))

result["test_errors"].append(self.misclassification_error(X_test,y_tes
t))

        return result

    def sgd_train(
        self,
        X_train: np.ndarray,
        y_train: np.ndarray,
        X_test: np.ndarray,
        y_test: np.ndarray,
        learning_rate: float = 1e-3,
        epochs: int = 30,
        batch_size: int = 1,
    ) -> 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": [],
        }
        #shuffle the data with given RNG and set indices
        for i in range(epochs):
            ind=np.arange(X_train.shape[0])
            RNG.shuffle(ind)
            X_train_shuff = X_train[ind]
            y_train_shuff = y_train[ind]
```

```
            for j in range(0, len(X_train_shuff), num_batches):
                #create the batches with the shuffled data
                start = i
                end = i+batch_size
                X_batch = X_train_shuff[start:end]
                y_batch = y_train_shuff[start:end]
                #step (learnign rate defaults to 1e-4)
                self.step(X_batch,y_batch,learning_rate)

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

result["train_errors"].append(self.misclassification_error(X_train,y_t
rain))
            result["test_losses"].append(self.loss(X_test,y_test))

result["test_errors"].append(self.misclassification_error(X_test,y_tes
t))

        return result

    def mini_batch_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": [],
        }
        #shuffle the data with given RNG and set indices
        for i in range(epochs):
            ind=np.arange(X_train.shape[0])
            RNG.shuffle(ind)
            X_train_shuff = X_train[ind]
            y_train_shuff = y_train[ind]

            for j in range(0, len(X_train_shuff), num_batches):
                #create the batches with the shuffled data
                start = i
                end = i+batch_size
                X_batch = X_train_shuff[start:end]
                y_batch = y_train_shuff[start:end]
```

```
                #step (learnign rate defaults to 1e-4)
                self.step(X_batch,y_batch,learning_rate)

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

        return result




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

    learning_rates = [1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1]
    scenarios = [
        {"name": "GD"},  # Assuming GD uses the full dataset
        {"name": "SGD"},
        {"name": "Mini-Batch GD"}
    ]
    best_rates = {}
    for scenario in scenarios:
        best_rate = None
        best_loss = float('inf')
        for rate in learning_rates:
            model = BinaryLogReg()
            history = model.train(x_train, y_train, x_test, y_test,
learning_rate=rate)
            train_loss = history['train_losses'][-1]
            if train_loss < best_loss:
                best_loss = train_loss
                best_rate = rate
        print(f"Best Learning Rate for {scenario['name']} is:
{best_rate} with a loss of: {best_loss}")
        best_rates[scenario['name']] = best_rate

    #Retrain and plot with best learnign rates
    model = BinaryLogReg()
    learn_rate_gd = best_rates['GD']
    history = model.train(x_train, y_train, x_test, y_test,
learning_rate=learn_rate_gd)

    # Plot losses
```

```
    plt.plot(history["train_losses"], label="Train")
    plt.plot(history["test_losses"], label="Test")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.title(f"Plot Losses (batch=100) with Learn Rate:
{learn_rate_gd}")
    plt.legend()
    plt.show()

    # Plot error
    plt.plot(history["train_errors"], label="Train")
    plt.plot(history["test_errors"], label="Test")
    plt.xlabel("Epochs")
    plt.ylabel("Misclassification Error")
    plt.title(f"Plot Errors (batch=100) with Learn Rate:
{learn_rate_gd}")
    plt.legend()
    plt.show()

    model_sgd = BinaryLogReg()
    learn_rate_sgd = best_rates['SGD']
    history_sgd = model_sgd.train(x_train, y_train, x_test, y_test,
learning_rate=learn_rate_sgd)

    # Plot losses
    plt.plot(history_sgd["train_losses"], label="Train")
    plt.plot(history_sgd["test_losses"], label="Test")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.title(f"Plot Losses SGD (batch=1) with Learn Rate: :
{learn_rate_sgd}")
    plt.legend()
    plt.show()

    # Plot error
    plt.plot(history_sgd["train_errors"], label="Train")
    plt.plot(history_sgd["test_errors"], label="Test")
    plt.xlabel("Epochs")
    plt.ylabel("Misclassification Error")
    plt.title(f"Plot Errors SGD (batch=1) with Learn Rate: :
{learn_rate_sgd}")
    plt.legend()
    plt.show()

    model_minibatch = BinaryLogReg()
    learn_rate_minibatch = best_rates['Mini-Batch GD']
    history_minibatch = model_minibatch.train(x_train, y_train,
x_test, y_test, learning_rate=learn_rate_minibatch)

    # Plot losses
```

```
    plt.plot(history_minibatch["train_losses"], label="Train")
    plt.plot(history_minibatch["test_losses"], label="Test")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.title(f"Plot Losses Mini Batch (batch=100) with Learn Rate:
{learn_rate_minibatch}")
    plt.legend()
    plt.show()

    # Plot error
    plt.plot(history["train_errors"], label="Train")
    plt.plot(history["test_errors"], label="Test")
    plt.xlabel("Epochs")
    plt.ylabel("Misclassification Error")
    plt.title(f"Plot Errors Mini Batch (batch=100) with Learn Rate:
{learn_rate_minibatch}")
    plt.legend()
    plt.show()
```

```
---------------------------------------------------------------------------
-----
ModuleNotFoundError                       Traceback (most recent call
last)
Cell In[14], line 6
      3 import matplotlib.pyplot as plt
      4 import numpy as np
----> 6 from utils import load_dataset, problem
      7 #from homeworks.log_regression.binary_log_regression import
RNG
      8
      9 # When choosing your batches / Shuffling your data you should
use this RNG variable, and not `np.random.choice` etc.
     10 RNG = np.random.RandomState(seed=446)

ModuleNotFoundError: No module named 'utils'
```

Bounding the Estimate

B2. Let us consider the setting, where we have n inputs, $X_1, ..., X_n \in \mathbb{R}^d$, and n observations $Y_i = \langle X_i, \beta^* \rangle + \epsilon_i$, for $i = 1, ..., n$. Here, β^* is a ground truth vector in $\mathbb{R}^d$ that we are trying to estimate, the noise $\epsilon_i \sim \mathcal{N}(0, 1)$, and the n examples piled up — $X \in R^{n \times d}$. To estimate, we use the least squares estimator $\widehat{\beta} = \min_\beta \|X\beta - Y\|_2^2$. Moreover, we will use $n = 20000$ and $d = 10000$ in this problem.

a. *[3 points]* Show that $\widehat{\beta}_j \sim \mathcal{N}(\beta_j^*, (X^T X)^{-1}_{j,j})$ for each $j = 1, ..., d$. *(Hint: see notes on confidence intervals.)*

b. *[4 points]* Fix $\delta \in (0, 1)$ suppose $\beta^* = 0$. Applying the proposition from the notes, conclude that for each $j \in [d]$, with probability at least $1 - \delta$, $|\widehat{\beta}_j| \leq \sqrt{2(X^T X)^{-1}_{j,j} \log(2/\delta)}$. Can we 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? Why or why not?

c. *[5 points]* Let's explore this question empirically. Assume data is generated as $x_i = \sqrt{(i \mod d) + 1} \cdot e_{(i \mod d)+1}$ where e_i is the ith canonical vector and $i \mod d$ is the remainder of i when divided by d. Generate each y_i according to the model above. Compute $\widehat{\beta}$ and plot each $\widehat{\beta}_j$ as a scatter plot with the x-axis as $j \in \{1, \ldots, d\}$. Plot $\pm\sqrt{2(X^T X)^{-1}_{j,j} \log(2/\delta)}$ as the upper and lower confidence intervals with $1 - \delta = 0.95$. How many $\widehat{\beta}_j$'s are outside the confidence interval? *Hint: Due to the special structure of how we generated x_i, we can compute $(X^T X)^{-1}$ analytically without computing an inverse explicitly.*

What to Submit:

- **Parts a, b:** Proof.
- **Part b:** Answer.
- **Part c:** Plots of $\hat{\beta}$ and its confidence interval **on the same plot**.

---

a:

Show: $\beta_j N\left(\beta_j^{¿}, \left(X^t X\right)^{-1}_{j,j}\right)$ for each j = 1,...,d

$\hat{\beta} = |) X\beta - Y|)_2^2 = \left(X^t X\right)^{-1} X^T Y$

because the noise is normally distributed $\epsilon\ N(0,1)$ => $Y = X\beta^{¿} + \epsilon$

Substituting: $\hat{\beta} = \left(X^t X\right)^{-1} X^T \left(X\beta^{¿} + \epsilon\right) ¿ = \beta^{¿} + \left(X^t X\right)^{-1} X^T n$ be because $(0, I)$ according to proposition 1 in the notes then $\hat{\beta}$ is also normally distributed

For $\hat{\beta}$ it is scaled by σ^2 which is 1. Also because it is normally distibuted each $\hat{\beta}$ is centered at $\beta_j^{¿}$

So it is a normal distribution, centered at $\beta_j^{¿}$ with $\left(X^t X\right)^{-1}$ scaled by 1 =>

$$\beta_j N\left(\beta_j^{¿}, \left(X^t X\right)^{-1}_{j,j}\right) \text{ for each j = 1,...,d}$$

b:

We cannot conclude this. If we look at the union bound used for spurious noise

$$\mathbb{P}(\bigcup_{i=1}^{d}\{|\widehat{\theta}_i - \theta_{*,i}| > \sqrt{2\sigma^2[(X^\top X)^{-1}]_{i,i}\log(2d/\delta)}\}) \leq \sum_{i=1}^{d}\mathbb{P}(|\widehat{\theta}_i - \theta_{*,i}| > \sqrt{2\sigma^2[(X^\top X)^{-1}]_{i,i}\log(2d/\delta)})$$
$$\leq \sum_{i=1}^{d}\frac{\delta}{d} = \delta.$$

the observation is that $1-\delta,\ \widehat{\theta}_i > \sqrt{2\sigma^2[(X^\top X)^{-1}]_{i,i}\log(2d/\delta)}$ so if we replace $\hat{\beta}$ with $\hat{\theta}$ we cannot conclude it and would be better suited to use a per-comparison basis on run all of the j's at the same time

c:

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

n=20000
d=10000
# 1-DELTA = 0.95
delta = 0.05
sigma = 1

# Yi = <Xi, β*> + epsilon (β* = 0) => Yi = epsilon (noise)
# generate random noise
y = np.random.normal(0, 1, n)
#generate the data
X=np.zeros((n,d))
row_ind = np.arange(n)
col_ind = row_ind % d
X[row_ind, col_ind]=np.sqrt(col_ind + 1)
XTX = np.linalg.inv(X.T@X)
beta_hat = XTX@X.T@y
```

Administrative

A8.

a. *[2 points]* About how many hours did you spend on this homework? There is no right or wrong answer :)

25+ hours