CSE546 HW2

Haoxin Luo

December 8, 2022

Exercise A1-Conceptual

a). With L1 regularization, we penalize the model by a loss function $L_1(w) = \sum_i |W_i|$, with L2 regularization, we penalize the model by a loss function $L_1(w) = \sum_i (w_i)^2$.If using gradient descent, we will iteratively make the weights change in the opposite direction of the gradient with a step size multiplied with the gradient, which indicates that more steep gradient takes a larger step, while a more flat gradient takes a smaller step. The gradient for L1 is either 1 or -1, except for when it is 0. Therefore, L1 regularization will move any weight towards 0 with the same step size, no matter what the weight's value is. In contrast, th L2 gradient is linearly decreasing towards 0 as the weight goes towards 0. Although L2-regularization will move any weight towards 0, it will take smaller and smaller steps as a weight approaches 0. In addition, if we plot out the regions of constrains given by L1 and L2 norm, L1 will give us a diamond shape, which has spikes at sparse points. We can find a touch point on a spike tip by letting the diamond touch the solution surface and thus resulting in a sparse solution.

b).

Upside is that it promotes a sparsity better than L1, the downside is that it's not convex.

c).

True, it won't capture the optimal solution

d).

One iteration of SGD save much computational power than GD, but SGD needs more iterations to achieve get the same error. GD takes fewer steps to converge to the minimal.

Exercise A2-Convexity and norms

a). Show that $f(x) = (\sum |xi|)$ is a norm.

We begin by showing that $|a+b|^2 = a^2 + 2ab + b^2 \leq a^2 + 2|a||b| + b^2 = (|a| + |b|)^2$, since $|a+b| > 0$, and $|a| + |b| > 0$, taking the square root for both sides and we have $|a + b| \leq |a| + |b|$.

We are going to show that $f(x) = (\sum |xi|)$ is a norm by checking

(i) non-negativity: $|xi| \geq 0$ for all i, so the sum is still non negative

(ii) absolute scalability: $f(ax) = \sum |ax_i| = \sum |a||x_i| = |a|f(x)$.

(iii) triangle inequality. $f(x) + f(y) = \sum |x_i| + \sum |y_i| = \sum |y_i| + |x_i| \geq \sum |y_i + x_i| = f(x+y)$.

end of proof.

b).

We want to make sure that g(x) satisfied all the three conditions. I will prove that it is not a norm by giving a counter example. Suppose k =2 and we have two arbitrary points $x = [0, 2]^T$ and $y = [2, 0]^T$, then $g(x) + g(y) = 4 < g(x + y) = 8$ which does not follow the triangular inequality $||x + y|| \leq ||x|| + ||y||$ for all $x, y \in R^n$ when n = 2

Exercise B1

We want to prove that $||x||_\infty \leq ||x||_2 \leq ||x||_1$

Start from the left hand side:

$$||x||_1 = (\sum |x_i|)^2 = \sum (|x_i|^2 + 2 \sum |x_i||x_j| \geq \sum |x_i|^2 = ||x||_2^2$$

Taking the square root since they are all positive:

$$||x||_1 \geq ||x||_2$$

Next:

$$||x||_2 = \sum |x_i|^2 \geq max(|x_i|)^2 = (max(|x_i|)^2) = ||x||_\infty$$

Taking the square root since they are all positive:

$$||x||_2 \geq ||x||_\infty$$

Therefore, $||x||_\infty \leq ||x||_2 \leq ||x||_1$

end of proof.

Exercise A3

a). I is not convex, line segment between point b and c is not in the set, but b and c are in the set. In other words, $\lambda x + (1 - \lambda)y \in A$ does not always hold when $(x, y) = (b, c)$

b). II is not convex, line segment between point a and d is not in the set, but a and d are in the set. In other words, $\lambda x + (1 - \lambda)y \in A$ does not always hold when $(x, y) = (a, d)$

Exercise A4.

a).

I is convex on [a, c]

b).

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

Exercise B2.

a).

Let $g(x)$, $f(x)$ be convex, take any $\lambda \in [0, 1]$, and any x, by convexity of f and g:

$$f(\lambda x + (1-\lambda)x) + g(\lambda x + (1-\lambda)x)$$

$$\leq \lambda f(x) + (1-\lambda)f(x) + \lambda g(x) + (1-\lambda)g(x)$$
$$= \lambda(g(x) + f(x)) + (1-\lambda)(g(x) + f(x))$$

This means $f(x) + g(x)$ is convex if $g(x)$, $f(x)$ are convex. Because of the above, $\sum_i^n f_i(x)$ be a sum of n convex functions, $f_i(x)$ is also convex. This can be shown by first $f_1(x) + f_2(x)$ is convex, so $f_1(x) + f_2(x) + f_3(x)$ is convex. By induction, $\sum_i^n f_i(x)$ is convex, and $f_{j+1}x$ is convex too, and it follows that $\sum_{i=1}^{j+1} f_i(x)$ As a result, $\sum_i^n f_i(x)$ is convex.

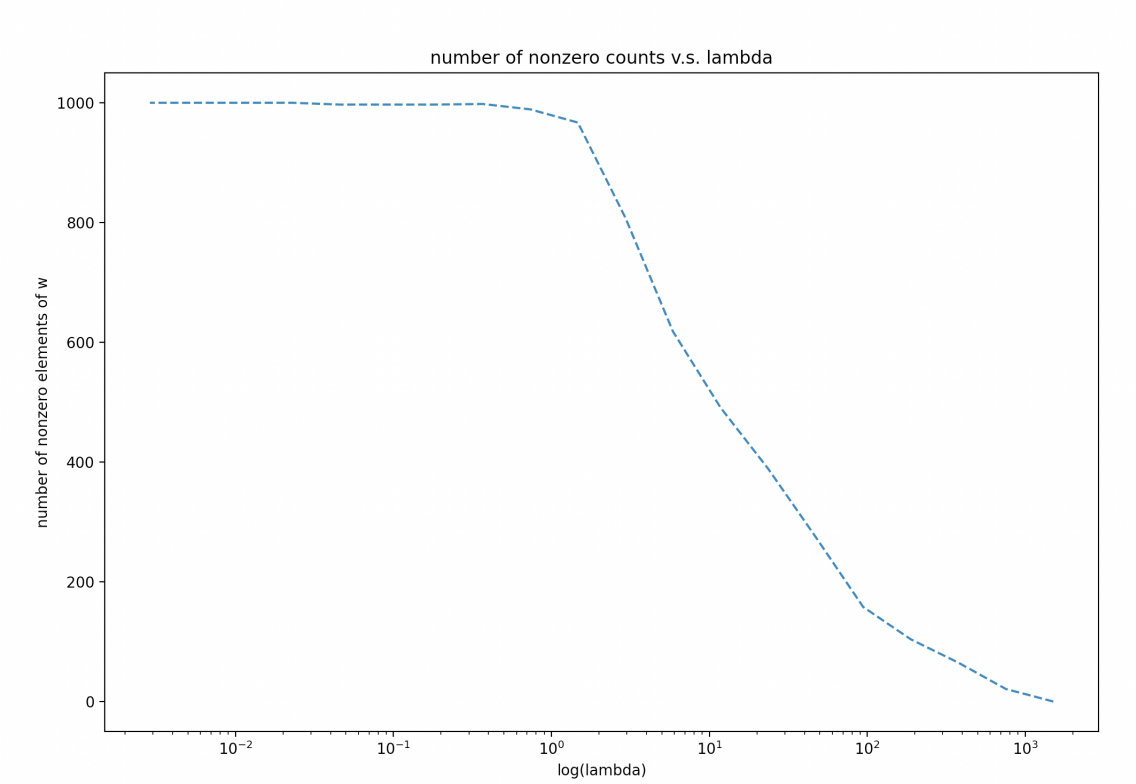
Since a norm is convex, $||w||$ is convex and $\lambda||w||$ is also convex with $\lambda > 0$. In addition, by the description of our question and what we have derived above, as $l_i(w)$ is convex, $\sum_i^n l_i(w)$ is convex. Thus, $\sum_i^n l_i(w) + \lambda||w||$ is convex as it is the sum of convex functions.

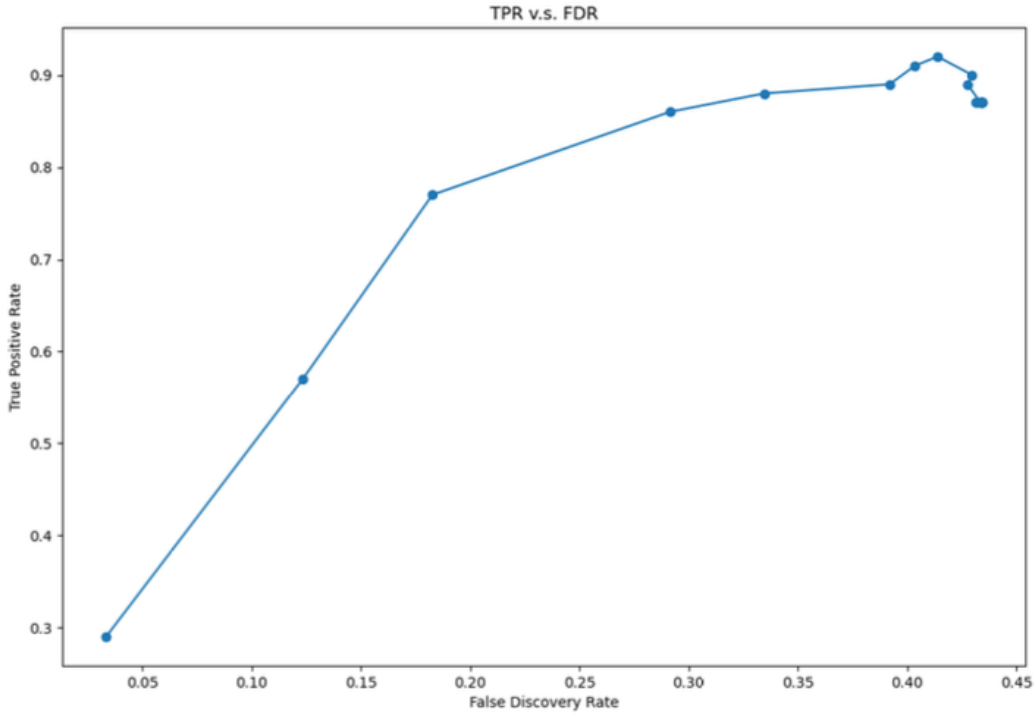
b).

This is because every local minimum of a convex function is a global minimum.

Exercise A5

a).





b).
Third λ would be the ideal solution.

c). Large λ forces more weights to become zeros (i.e, more sparsity) When λ become too large, we will get very low true positive rate. But if it is too small, we would end up having a high

false discovery rate. We had better use a validation set to get the most optimal choice of λ

```
@problem.tag("hw2-A")
def precalculate_a(X: np.ndarray) -> np.ndarray:
    """Precalculate a vector. You should only call this function once.

    Args:
        X (np.ndarray): An (n x d) matrix, with n observations each with d features.

    Returns:
        np.ndarray: An (d, ) array, which contains a corresponding `a` value for each feature
    """
    #raise NotImplementedError("Your Code Goes Here")
    return 2*np.sum(X**2, 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]:
    """Single step in coordinate gradient descent.
    It should update every entry in weight, and then return an updated version of weight alon

    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.
        a (np.ndarray): An (d,) array. Respresents precalculated value a that shows up in the
        _lambda (float): Regularization constant. Determines when weight is updated to 0, and

    Returns:
        Tuple[np.ndarray, float]: Tuple with 2 entries. First represents updated weight vecto
            Bias should be calculated using input weight to this function (i.e. before any up

    Note:
        When calculating weight[k] you should use entries in weight[0, ..., k - 1] that have
        This has no effect on entries weight[k + 1, k + 2, ...]
    """
    #raise NotImplementedError("Your Code Goes Here")
    b= np.mean(y-(X.dot(weight)))
    n, d = X.shape
    for j in range(d):

        col_nk = np.arange(d) != j
        a_j = a[j]
        c_j = 2*np.sum(X[:,j] * (y-(b+X[:,col_nk].dot(weight[col_nk]))), axis=0)

        if c_j < -_lambda:
```

```
            weight[j] = (c_j + _lambda) /a_j
        elif c_j > _lambda:
            weight[j] = (c_j - _lambda) /a_j
        else:
            weight[j] =0

    return weight, b

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

    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
    """
    #raise NotImplementedError("Your Code Goes Here")
    return (np.linalg.norm(X.dot(weight) + bias - y))**2 +  _lambda * np.linalg.norm(weight,

@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]:
    """Trains a model and returns predicted 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.
        _lambda (float): Regularization constant. Should be used for both step and loss.
        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.
```

```
    Returns:
        Tuple[np.ndarray, float]: A tuple with first item being array of shape (d,) represent
            and second item being a float .

    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 prob
            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:
        start_weight = np.zeros(X.shape[1])
    a = precalculate_a(X)
    old_w: Optional[np.ndarray] = None
    #raise NotImplementedError("Your Code Goes Here")

    while not convergence_criterion(start_weight, old_w, convergence_delta):
        old_w = np.copy(start_weight)

        start_weight, bias = step(X,y,start_weight,a,_lambda)

    return start_weight, bias

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

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

    Returns:
        bool: False, if weight has not converged yet. True otherwise.
    """
    #raise NotImplementedError("Your Code Goes Here")
    if old_w is None:
        return False

    return (np.linalg.norm(weight-old_w, ord=np.inf) < convergence_delta)
```

```
@problem.tag("hw2-A")
def main():
    """
    Use all of the functions above to make plots.
    """
    #raise NotImplementedError("Your Code Goes Here")
    n,d,k = 500,1000,100

    # sample X from normal(0,1)
    X=np.random.normal(0,1,(n,d))
    err =np.random.normal(0,1,(n,))

    #generates weights
    weight = np.zeros(d)
    for i in range(k):
        weight[i] = (i+1)/k

    #generate y
    y = X.dot(weight) + err

    # generate lambdas
    lambda_max = np.max(np.sum(2*X*(y - np.mean(y))[:, None], axis =0))
    lambdas = [lambda_max / (2**i) for i in range(20)]

    #save results
    non_zero = []
    tp = []
    fd = []


    for k in lambdas:
        train_w, _ = train(X,y,k,0.01)

        res_non_zero = np.sum(abs(train_w)>1e-10)
        res_false = np.sum(train_w[abs(weight) <= 1e-10] > 1e-10)
        res_true=np.sum(train_w[abs(weight) > 1e-10] > 1e-10)

        non_zero.append(res_non_zero)
        tp.append(res_true / k)
        fd.append(res_false / res_non_zero)

    #generate plots
    plt.figure(figsize=(12,8))
    plt.plot(lambdas, non_zero, '--')
    plt.xscale('log')
    plt.title('number of nonzero counts v.s. lambda')
    plt.xlabel('log(lambda)')
```

```
plt.ylabel('number of nonzero elements of w')
plt.show()

plt.figure(figsize=(12,8))
plt.plot(fd, tp, '--')
plt.title('TPR vs FDR')
plt.xlabel('False Discovery Rate')
plt.ylabel('True Positive Rate')
plt.show()
```

Exercise A6

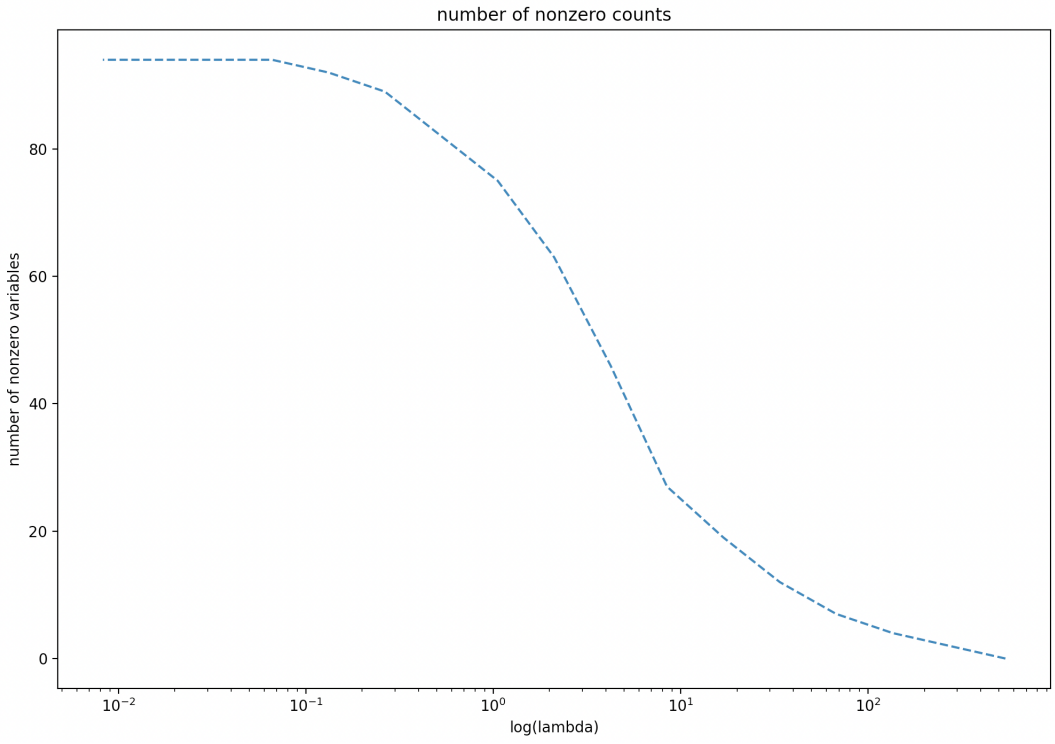
a).

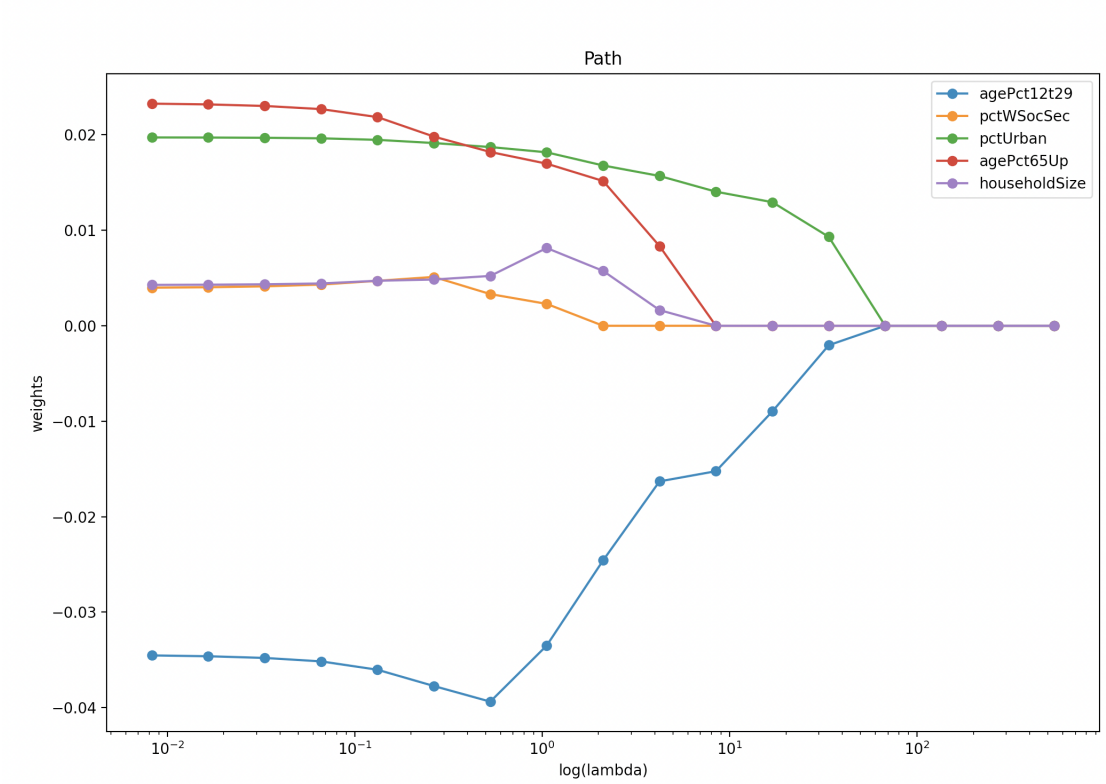
The historical policy choices in the US would lead to variability in PctPolicBlack (percent of police that are african american), LemasPctOfficDrugUn(percent of officers assigned to drug units), and PolicOperBudg.

b).

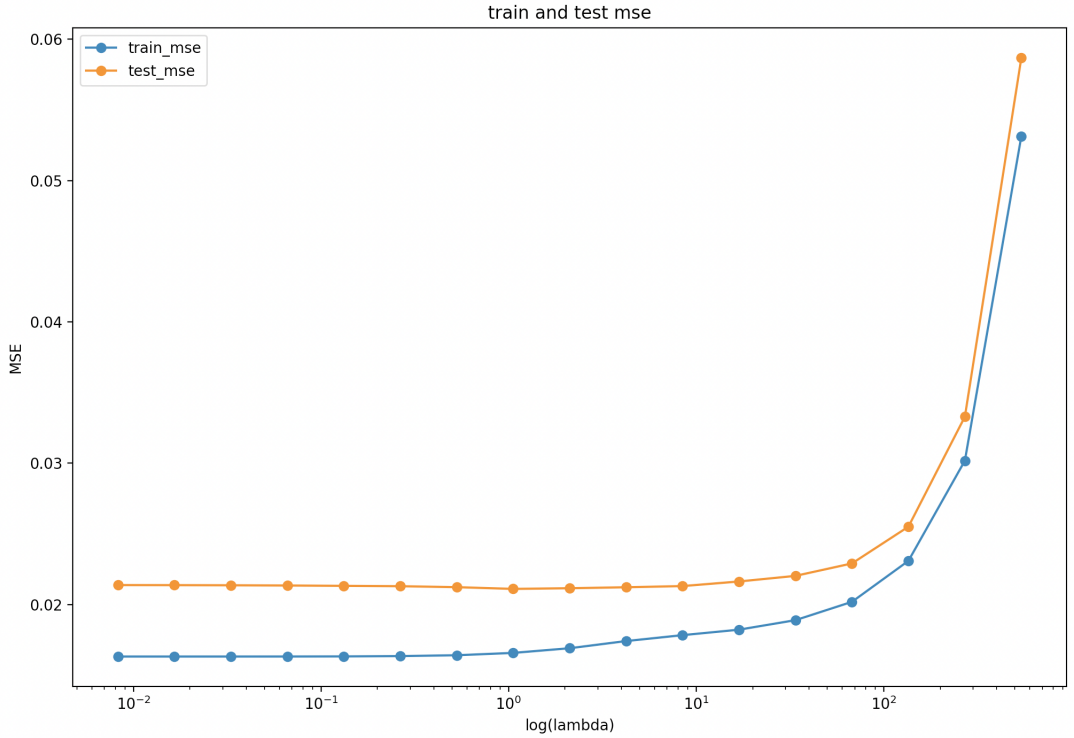
PctUnemployed, PctIlleg, NumInShelters could be interpreted as reasons for higher levels of violent crime, might might actually be a result rather than (or in addition to being) the cause. People would find it hard to find a job if being involved in a violent crime and thus the percentage of unemployed may be higher. The percentage of kids born to never married would be higher in high criminal districts, those parents are not capable to raising up a child and they fail to take any interventions for pregnancy. Higher levels of violent crime results in more homeless people (Those lose home, family, job and social status end up living in homeless shelters).

c).





d).



e).

f). PctIlleg has the most positive weight, indicating that crime rate is most positively related with PctIlleg; PctKids2Par has the most negative weight, indicating that crime rate is most negatively related with PctKids2Par

g).

Correlation does not imply causality, fewer people over 65 in high crime locations does not

mean number of people over 65 decreases crime rate, it may be the case that older people does not live in those areas. Similarly, seeing fire trucks around buildings on fire does not mean they cause the fire, the root cause is the fire.

```
@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")
    X_train = df_train.drop('ViolentCrimesPerPop', axis=1).to_numpy()
    y_train = df_train['ViolentCrimesPerPop'].to_numpy()

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


    #generate lambdas
    lambda_max = np.max(np.sum(2*X_train*(y_train - np.mean(y_train))[:, None], axis =0))
    lambdas = [lambda_max / (2**i) for i in range(17)]

    #save results
    non_zero = []
    train_mse = []
    test_mse = []
    w_path = []

    #make predictions
    def predict(X, w, b):
        return X.dot(w) + b

    def mse(x,y):
        return np.mean((x-y) ** 2)

    w_train, bias = train(X_train, y_train, 30)
    for k in lambdas:
        train_w, bias = train(X_train,y_train,k)
        res_non_zero = np.sum(abs(train_w)>1e-10)
        non_zero.append(res_non_zero)

        w_path.append(np.copy(train_w))

        train_mse. append(mse(predict(X_train, train_w, bias), y_train))
        test_mse.append(mse(predict(X_test, train_w, bias), y_test))

    #generate plots
    plt.figure(figsize=(12,8))
```

```
plt.plot(lambdas, non_zero, '--')
plt.xscale('log')
plt.title('number of nonzero counts')
plt.xlabel('log(lambda)')
plt.ylabel('number of nonzero variables')
plt.show()


#generate plots
plt.figure(figsize=(12,8))
w_path = np.array(w_path)
col_names = ['agePct12t29', 'pctWSocSec', 'pctUrban', 'agePct65Up', 'householdSize']
col_index = [3,12,7,5,1]
for p, label in zip(w_path[:, col_index].T, col_names):
    plt.plot(lambdas, p, '-o', label = label)
plt.legend()
plt.xscale('log')
plt.title('Path')
plt.xlabel('log(lambda)')
plt.ylabel('weights')
plt.show()


#generate plots
plt.figure(figsize=(12,8))
plt.plot(lambdas, train_mse, '-o', label = 'train_mse')
plt.plot(lambdas, test_mse, '-o', label = 'test_mse')
plt.xscale('log')
plt.title('train and test mse')
plt.legend()
plt.xlabel('log(lambda)')
plt.ylabel('MSE')
plt.show()
```

Exercise A7

a)

$$\nabla_w L(w) = argmin(y - Xw)^T(y - Xw)$$
$$= argminy^T y - w^T X^T y - y^T Xw + w^T X^T Xw$$
$$= argminy^T y - 2w^T X^T y + w^T X^T Xw$$

Taking the derivative with respect to w, we get:

$$\nabla_w L(w) = 0 - 2X^T y + 2(X^T X)w$$

$$\nabla_w L(w) = -2X^T y + 2(X^T X)w$$

b)

$$w_{k+1} - w^* = w_k - \alpha(\nabla_{w_k} L(w_k) - \nabla_{w^*} L(w^*) - w^*$$

Using what we have in part a and substitute $\nabla_{w_k} L(w_k)$, $\nabla_{w^*} L(w^*)$:

$$w_{k+1} - w^* = w_k - \alpha(-2X^T y + 2(X^T X)w - (-2X^T y + 2(X^T X)w^*)) - w^*$$

$$= w_k - \alpha(2(X^T X)w_k - 2(X^T X)w^*) - w^*$$
$$= w_k - 2\alpha X^T X(w_k - w^*) - w^*$$
$$= (w_k - w^*) - 2\alpha X^T X(w_k - w^*)$$
$$= (I - 2\alpha X^T X)(w_k - w^*)$$

Finally

$$||w_{k+1} - w^*|| = ||(I - 2\alpha X^T X)(w_k - w^*)||$$

c)

From part b we knew

$$||w_{k+1} - w^*|| = ||(I - 2\alpha X^T X)(w_k - w^*)||$$

Let $A = X^T X$, which is always positive semidefinite, then

$$||w_{k+1} - w^*|| = ||(I - 2\alpha A)(w_k - w^*)||$$

Let, h, H be the smallest and the largest eigenvalue of A respectively, and let $\lambda = max(h, H)$, we know that λ is the eigenvalue of matrix $A = X^T X \in R^{d \times d}$ that has the largest magnitude, and for any vector $x \in d$, we have $||Ax|| = ||X^T Xx|| \leq |\lambda| \cdot ||x||$. In addition, $(I - 2\alpha A)x = Ix - 2\alpha Ax = Ix - 2\alpha\lambda x = (I - 2\alpha\lambda)x$, where $I - 2\alpha\lambda$ is an eigenvalue of $(I - 2\alpha A)$. As a result of that, when $\rho = max(|1 - 2\alpha h|, |1 - 2\alpha H|)$:

$$||w_{k+1} - w^*|| = ||(I - 2\alpha X^T X)(w_k - w^*)|| \leq \rho||w_k - w^*||$$

Exercise B3

a).

By the previous part,

$$||w_k - w^*|| \leq \rho||w_{k-1} - w^*||$$

Base case: when k = 1, then $||w_1 - w^*|| \leq \rho||w_0 - w^*||$, which is true.

Now, we want to assume when k=n, $||w_n - w^*|| \leq \rho^n||w_0 - w^*||$ is still true when $n \geq 1$ and n is an integer

We want to show that k = n+1, such that $||w_{n+1} - w^*|| \leq \rho^{n+1}||w_0 - w^*||$ is also true for $n \geq 1$ and n is an integer. We knew that $||w_{n+1} - w^*|| \leq \rho||w_n - w^*||$ and $||w_n - w^*|| \leq \rho||w_{n-1} - w^*||$, this gives us $||w_{n+1} - w^*|| \leq \rho^2||w_{n-1} - w^*||$ where we can see 2+(n-1) = n+1

By induction, $||w_{n+1} - w^*|| \leq \rho^{n+1}||w_0 - w^*||$ where 0+(n+1)=n+1. In conclusion, $||w_k - w^*|| \leq \rho^k||w_0 - w^*||$

b).

$k > log_{1/\rho}(\frac{||w_0 - w^*||}{\varepsilon})$ implies $(1/\rho)^k > (\frac{||w_0 - w^*||}{\varepsilon})$, and this gives $\rho^k < (\frac{\varepsilon}{||w_0 - w^*||})$, which further gives $\rho^k(||w_0 - w^*||) < \varepsilon$.

Since from the previous part, we have $||w_k - w^*|| \leq \rho^k||w_0 - w^*||$, we can conclude that $||w_k - w^*|| < \varepsilon$

Exercise A8

about 20 hours

Collaborator: Andy Chen, Yi-Ling Chen