Class: CSE 546 Name: Hongda Li HW2A

A.0: Conceptual Questions

A.0.a

False. a large value of w_i can be caused by overfitting. The error will only increase if it's proven that the features of "Number of Bathrooms" is not having collinearity with other features, or it's the last feature that goes to zero as we increase the lambda value for Ridge Regression.
Another way to think about it is, assume that there is another feature says: "The size of bath room", then in that case, this feature is essentially the same as the removed feature: "Number of Bathrooms", removing it will not have a huge impact on the amount of errors for the model.

A.0.b

The L1 norm is more likely because the derivative of w_j is ± 1, which means that it's not related to the actual value of w_j. Therefore, which ever has the most amount of reduction on the error, that w_j is getting pushed to zero.
Or, use professor's Simon's way of doing it, the L1 Norm $||w||_1$ forms a simplex situated around the origin. If the optimal is not inside of the simplex, then the algorithm will always goes to the vertex of the simplex to optimize it, and going to the vertex of the L1 Ball is setting one of the features to zero.

A.0.c

The regularizer $\sum_i |w|^{0.5}$ promotes sparsity because the region $\sum_i |w|^{0.5} = 1$ is very pointy but this is also a bad regularizer because such region is not convex, potentially given multiple solutions, or, making gradient descent slow.

A.0.d

True, assume it's so large thta it just shoots out of the convex region.

A.0.e

It works by randomly sample a batch, and we know that the sampled samples are likely to be representative of the whole sample, and in that way, the gradient computed is pointing, somewhat, in the correct direction. The dot product between the gradient given by FIRST iteration GSD and the best Gradient direction is always positive. This is true because the first iteration will always pull the w_j into the range where the data is located in.
This is made reasonable by consider extreme choices of sample x_i that gives maximal, or minimal value of predictor, and this quantity will be bounded. Hence, the first few descent step will always pull the model into that range.

A.0.f

SGD is faster to compute compare to GD becausit only uses a few samples but it's less likely to have a clear convergence near the optimal because it's random.

A.1: Convexity and Norms

Here is the axioms for norms, let $x, y \in \mathbb{R}^n$, and let $|| \bullet ||$ be the norm operator, then it must satisfy:

1. $\|x\| > 0$
2. $\|x + y\| \leq \|x\| + \|y\|$

3. $\|ax\| = |a|\|x\|$

A.1.a

Objective: Show that $f(x) = \sum_{i=1}^{n} |x_i|$ is a norm.

1. It's always positive because:

$$|x_i| \geq 0 \quad \forall 1 \leq i \leq n \tag{A.1.a.1}$$
$$\sum_{i=1}^{n} |x_i| > 0$$
$$\|x\| \geq 0$$

2. The trianle inequality is always ture. Let's take 2 vectors $x, y \in \mathbb{R}^n$, and for each of their corresponding elements, we can apply the triangule inequality for absolute value, which is:

$$|x_i + y_i| \leq |x_i| + |y_i| \quad \forall 1 \leq i \leq n \tag{A.1.a.2}$$
$$\sum_{i=1}^{n} |x_i + y_i| \leq \sum_{i=1}^{n} |x_i| + \sum_{i=1}^{n} |y_i|$$
$$\|x + y\| \leq \|x\| + \|y\|$$

3. And the absolute scaler it's like:

$$\begin{aligned} \|\alpha x\| &= \sum_{i=1}^{n} |\alpha x_i| \\ &= \sum_{i=1}^{n} |\alpha||x_i| \\ &= \alpha|\|x\| \end{aligned} \tag{A.1.a.3}$$

A.1.b

Let's consider the case when:

$$x = \begin{bmatrix} \frac{1}{9} \\ \frac{1}{9} \end{bmatrix} \quad y = \begin{bmatrix} \frac{8}{9} \\ \frac{8}{9} \end{bmatrix} \tag{A.1.b.1}$$

Then let's compute:

$$x + y = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{A.1.b.2}$$
$$\implies \|x + y\| = 1 + 1 = 2$$

However:

$$\begin{aligned} \|x\| &= \times \left(\frac{2}{\sqrt{9}} \right)^2 = \frac{4}{9} \\ \|y\| &= \left(2 \times \frac{2\sqrt{2}}{3} \right)^2 = \left(\frac{4\sqrt{2}}{2} \right)^2 \\ &= \frac{8}{81} \end{aligned} \tag{A.1.b.3}$$
$$\|x\| + \|y\| = \frac{44}{81} < 2 = \|x + y\|$$

This is a contradiction, therefore this cannot be a way of computing norm.

A.2

A.2.I

This is not convex because a the intersection of the line connecting point b, c are not the whole line. A section of the line lies outside of the shaded region.

A.2.II

Triangle is a convex. Any line segment defined by 2 points in the triangle is inside the triangle

A.2.III

It's not convex. Line segment connection A a, b lies outside of the shape.

A.3

A.3.a

It's convex because $f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y)\ \forall \lambda \in [0,1]$ is true. All the line segment defined via 2 points on the function lies above the function, which can be easily verified graphically.

A.3.b

The function is not convex because $f(\frac{b+c}{2}) > \frac{1}{2}f(b) + \frac{1}{2}f(c)$. The mid point of the line segment defined by $f(b), f(c)$ below the function value at the mid point.

A.3.c

The midpoint of the line segment defined by $f(a), f(c)$ lies below the function evaluated at the mid point of the function.

A.3.d

The function III is convex on $[c, d]$. It can be verified visually.

A.4

This is the code for producing plots in A.4:

```
"""
This code is for CSE 546, Spring 2021, Question A4.

Author: Hongda Alto Li, Github Account: iluvjava.

Please don't copy my code cause my code is well crafted and has my style in it.

"""

import numpy as np
array = np.array
zeros = np.zeros
norm = np.linalg.norm
inf = np.inf
mean = np.mean
sum = np.sum
abs = np.abs
sign = np.sign
ones = np.ones
```

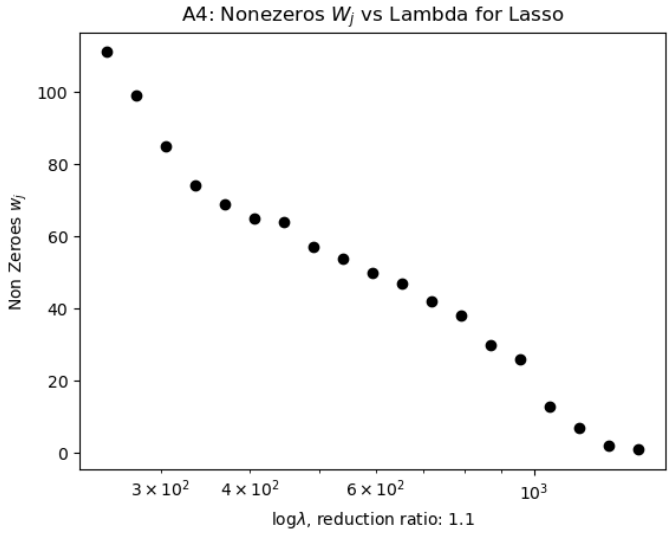
```
max = np.max
randn = np.random.randn

import matplotlib.pyplot as plt
plot = plt.plot
xscale = plt.xscale
show = plt.show
title = plt.title
xlabel = plt.xlabel
ylabel = plt.ylabel
legend = plt.legend
saveas = plt.savefig

class LassoRegression:

    def __init__(this, regularization_lambda, delta=1e-6, verbose=False):
        """
            create an instance of Lasso fit
        :param regularization_lambda:
            The lambda for L2 norm
        :param delta:
            The tolerance for measuring the convergence of the w parameter for coordinate descend
                .
        :param verbose:
            Whether to print out all the message when doing the coordinate descned.
        """
        this.Lambda = regularization_lambda
        this.Verbose = verbose
        this.Delta = delta
        this._w = None
        this._b = None

    def fit(this, X, y):
        """
            Fit that data.
            NOTE: Data standardization is not used.
        :param X:
            Row data matrix. Should be n by d where n is the number of samples and d is the
                number of features.
        :param y:
            Label vector, strictly on dimensional.
        :return:
            This model.
        """
        assert type(X) is np.ndarray and type(y) is np.ndarray, \
            "Must␣use␣numpy␣array␣to␣train␣the␣lasso."
        assert len(X.shape) == 2 and len(y.shape) == 1, \
            "X␣must␣be␣row␣data␣matrix,␣2d␣and␣y␣must␣be␣a␣1d␣vector,␣numerical."
        assert X.shape[0] == y.shape[0], \
            "The␣number␣of␣rows␣of␣X␣must␣equal␣to␣the␣number␣elements␣in␣y.␣"
        MaxItr = 10000
        n, d = X.shape[0], X.shape[1]
        y2d = y[:, np.newaxis]
        deltaW = this.Delta*ones((n, 1))*1.1
        # Use previous for optimization if model is asked to optimize for a second time.
        w = zeros((d, 1)) if this._w is None else this.w
        l = this.Lambda
        Itr = 0
        while norm(deltaW, inf) > this.Delta and Itr < MaxItr:
            Itr += 1
            # compute offset vector.
            b = mean(y2d - X@w)
            # Compuate all the k at once, because it's not related to w_k
            a = 2*sum(X**2, axis=0)
            for k in range(d):
                a_k = a[k]
                Indices = [J for J in range(d) if J != k]
                c_k = 2*sum(
```

```
                    X[:, [k]]
                    *
                    (y2d - (b + X[:, Indices]@w[Indices]))
                )
                w_k = 0 if (abs(c_k) < l) else (c_k - sign(c_k)*l)/a_k
                deltaW = abs(w_k - w[k])
                w[k] = w_k
                this._print(f"optimizing␣on␣w_{k},␣get␣w_{k}␣=␣{w[k]}")
        if MaxItr == Itr: raise Exception("Coordinate␣descent␣Max␣Itr␣reached␣without␣converging"
            )
        this._weights = w
        this._b = b
        return this


    @property
    def w(this):
        return this._weights.copy()

    @property
    def b(this):
        return this._b.copy()

    def _print(this, mesg):
        if this.Verbose: print(mesg)


def LassoLambdaMax(X, y):
    """
        Given the samples matrix and the labels, the function reurns the minimal lambda
        such that after running the lasso algorithm, all the features model
        parameters will be set to zeros by this lambda.
    :param X:
        This is the row data matrix, n by d matrix.
    :param y:
        This is label vector, 1d vector with a length of d.
    :return:
    """
    assert type(X) is np.ndarray and type(y) is np.ndarray, \
        "Must␣use␣numpy␣array␣to␣train␣the␣lasso."
    assert len(X.shape) == 2 and len(y.shape) == 1, \
        "X␣must␣be␣row␣data␣matrix,␣2d␣and␣y␣must␣be␣a␣1d␣vector,␣numerical."
    assert X.shape[0] == y.shape[0], \
        "The␣number␣of␣rows␣of␣X␣must␣equal␣to␣the␣number␣elements␣in␣y.␣"
    y = y[:, np.newaxis]
    return max(2*abs(X.T@(y - mean(y))))


def GetLassoSyntheticTestdata(n:int, d:int, k:int, sigma=1):
    assert (n > 0) and (d > 0) and (k > 0), "n,␣d,␣k␣all␣have␣to␣be␣larger␣than␣zeros"
    assert k < n, "k␣has␣to␣be␣<␣n"
    WTruth = array([JJ/k if JJ <= k else 0 for JJ in range(1, d + 1)], dtype=np.float)[:, np.
        newaxis]
    Noise = np.random.randn(n, 1)*sigma
    X = randn(n, d) # std normal for best stability of the coordinate descend algorithm.
    return X, (X@WTruth + Noise).reshape(-1), WTruth


def A4a_b():

    # Part (a)
    n, d, k = 500, 1000, 100
    X, y, Wtrue = GetLassoSyntheticTestdata(n, d, k)
    LambdaMax = LassoLambdaMax(X, y)
    Ws = []
    # Feature Chosen for each lambda
    Lfc = [] # lambda and features count
    Lambda = LambdaMax
```

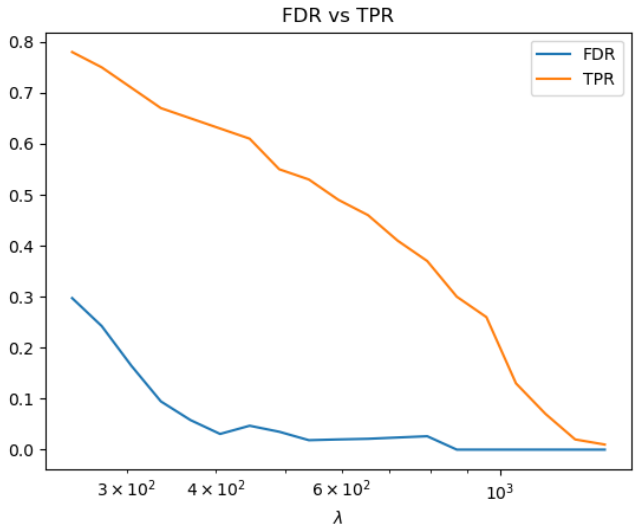
```
    r = 1.1
    while len(Lfc) == 0 or Lfc[-1][1] < k:
        Model = LassoRegression(regularization_lambda=Lambda)
        Model.fit(X, y)
        NonZeros = sum(Model.w != 0)
        Ws.append(Model.w)
        print(f"NonZeros: {NonZeros}")
        Lfc.append((Lambda, NonZeros))
        Lambda /= r
    plot([_[0] for _ in Lfc], [_[1] for _ in Lfc], "ko")
    xscale("log")
    xlabel(f"$\log{{\lambda}}$, reduction ratio: {r}")
    ylabel("Non Zeroes $w_j$")
    title("A4: Nonezeros $W_j$ vs Lambda for Lasso")
    show()
    saveas("A4a-plot.png", format="png")


    # Part (b)
    # The first k elements in Wtrue is always going to be non-zeroes.
    # FDR: (Incorrect Nonzeroes in w_hat)/(total number of nonzeroes in w_hat)
    # TPR: (# of correct non-zeroes in w_har)/(k)

    WTrueNonZeroes = Wtrue != 0
    FDR = []
    TPR = []
    Lambdas = [_[0] for _ in Lfc]
    for WHat in Ws:
        WHatNonZeros = WHat != 0
        if sum(WHatNonZeros) == 0:
            Lambdas.pop(0)
            continue
        FDR.append(sum(WHatNonZeros * ~WTrueNonZeroes)/sum(WHatNonZeros))
        TPR.append(sum(WHatNonZeros[:100])/k)
    plot(Lambdas, FDR)
    plot(Lambdas, TPR)
    title("FDR vs TPR")
    xlabel("$\lambda$")
    xscale("log")
    legend(["FDR", "TPR"])
    show()
    saveas("A4b-plot.png", format="png")




def main():
    def SimpleTest():
        N, d = 40, 4
        X = np.random.rand(N, d)
        Noise = np.random.randn(N, 1)*1e-3
        Wtrue = np.random.randint(0, 100, (d, 1))
        y = X@Wtrue + Noise
        y = y.reshape(-1)

        LambdaMax = LassoLambdaMax(X, y)
        print(f"LambdaMax: {LambdaMax}")

        Model = LassoRegression(regularization_lambda=LambdaMax)
        Model.fit(X, y)
        print(Model.w)
        print(Model.b)

        Model = LassoRegression(regularization_lambda=LambdaMax/2)
        Model.fit(X, y)
        print(Model.w)
        print(Model.b)
```

```
        Model = LassoRegression(regularization_lambda=0)
        Model.fit(X, y)
        print(Model.w)
        print(Model.b)
    A4a_b()


if __name__ == "__main__":
    import os
    print(f"cwd: {os.getcwd()}")
    print(f"dir: {os.curdir}")
    main()
```

A.4.a



A.4.b



A.4.c

A.5

A.4

A.6