Math 7243 Machine Learning - Homework 2

For programming questions, you can only use numpy library. You should not use any build in function from Scikit-learn or StatsModels libraries.

Problem 1 Loss Functions:

Let X be the data matrix and θ be the parameter vector.

a) In Lecture 2, we showed that the residual sum of square can be written

$$RSS(\theta) = (Y - X\theta)^T (Y - X\theta)$$

Find a **critical point** for $RSS(\theta)$ by calculate $\frac{\partial}{\partial \theta} RSS(\theta) = 0$.

b) **Ridge regression** changes the loss function to add in a term penalizing the θ if they get to large: For any positive number λ, the Ridge loss function

$$\text{Ridge}_\lambda(\theta) = (Y - X\theta)^T (Y - X\theta) + \lambda \theta^T \theta$$

Find an expression for the location of the critical point of $\text{Ridge}_\lambda(\theta)$.

$\beta = \theta$ in solution.

a) $RSS(\beta) = (Y - X\beta)^T (Y - X\beta) = \beta^T X^T X \beta - 2Y^T X\beta + Y^T Y$

$\frac{\partial}{\partial \beta} RSS(\beta) = 2X^T X\beta - 2X^T Y = 2X^T (X\beta - Y)$

$\therefore 0 = \frac{\partial}{\partial \beta} RSS(\beta)$ when $X^T (X\beta - Y) = 0$

$\therefore X^T Y - X^T X\beta = 0, \beta = (X^T X)^{-1} X^T Y$

$\therefore RSS(\beta)$ has an critical point when $\beta = (X^T X)^{-1} X^T Y$

b) $\because Ridge_\lambda(\beta) = (y - X\beta)^T (y - X\beta) + \lambda\ \beta^T \beta$

$\therefore \frac{\partial}{\partial \beta} Ridge_\lambda(\beta) = 2X^T (X\beta - y) + 2\lambda^2 \beta = 2(X^T X + \lambda\ I)\beta - 2X^T y$

$\therefore 0 = \frac{\partial}{\partial \beta} Ridge_\lambda(\beta)$ when $\beta = (X^T X + \lambda I)^{-1} X^T y$

$\therefore$ The location of the critical point of $Ridge_\lambda(\beta)$ is $\beta = (X^T X + \lambda\ I)^{-1} X^T y$

Problem 2 - Computing Linear Regression:

Consider the points

$x^{(i)}$	1.2	3.2	5.1	3.5	2.6
$y^{(i)}$	7.8	1.2	6.4	2.6	8.1

a). Fit a linear function to this dataset when the loss is RSS. You may use a computer to solve the matrix equation but you should report the best fit function.

b). Fit a linear function to this dataset when the loss is the Ridge Loss from Problem 1.b) with $\lambda = 1$ and with $\lambda = 10$. What specifically explains the difference in values between the three fits.

(1) Linear function: $y(x) = \beta_0 + \beta_1 x$.

$$X = \begin{pmatrix} 1 & 1.2 \\ 1 & 3.2 \\ 1 & 5.1 \\ 1 & 3.5 \\ 1 & 2.6 \end{pmatrix}, Y = \begin{pmatrix} 7.8 \\ 1.2 \\ 6.4 \\ 2.6 \\ 8.1 \end{pmatrix}.$$

By Problem 2, $RSS(\beta)$ has an critical point when $\beta = (X^T X)^{-1} X^T Y = \begin{pmatrix} 7.3311 \\ -0.6766 \end{pmatrix}$.

Therefore, the best fit function is $y(x) = 7.3311 - 0.6766x$.

(2) We will use the formula $\beta = (X^T X + \lambda I)^{-1} X^T \vec{y}$ to do the calculation. However, we don't want to put penalty on β_0.
We need to centralize our data by calculate $x' := x^{(i)} - \overline{x}$ and $y' := y^i - \overline{y}$
In this new centralized data, we have the linear model $y' = \beta_1 x'$ with no intersection.
So, now, we can use the formula to the new data (X, y), and calculate the coefficient $\beta = (X^T X + \lambda I)^{-1} X^T \vec{y}$
$\overline{y} = 5.22$ and
So $y - \overline{y} = \beta_1 (x - \overline{x})$.
When $\lambda = 1, \beta_1 = -0.60168365$.
When $\lambda = 10, \beta_1 = -0.30130907$.
When $\lambda = 100, \beta_1 = -0.05028326$.

Actually, when you do the calculation in the formula, you can keep the the original y since $X^T \overline{y} = 0$ for cauterized X.

The following answer will be **wrong** since it directly used the formula without centralize data and that means the penalty is also on the intersection.

For $\lambda = 1, \beta = \begin{pmatrix} 3.1152 \\ 0.4749 \end{pmatrix}$. The function is $y(x) = 3.1152 + 0.4749x$.

For $\lambda = 10, \beta = \begin{pmatrix} 0.1792 \\ 0.4672 \end{pmatrix}$. The function is $y(x) = 0.1792 + 0.4672x$.

code:

```
In [1]: import numpy as np
```

```
In [2]: def normal_equation(x, y, c=None):
            if c is None:
                return np.linalg.inv(x.T.dot(x)).dot(x.T).dot(y)  #Linear Regression
            else:
                return np.linalg.inv(x.T.dot(x)+c*np.identity(1)).dot((x.T).dot(y)) #Ridge Regression
```

```
In [3]: x=np.array([1.2, 3.2, 5.1, 3.5, 2.6])
        y=np.array([7.8, 1.2, 6.4, 2.6, 8.1])
```

Linear Regression: ¶

```
In [4]: xa=np.append(np.ones(x.shape[0]), x)
        xc = xa.reshape(2,-1).T
```

```
In [5]: coeff=normal_equation(xc,y)
        coeff
Out[5]: array([ 7.33109118, -0.67663179])
```

Ridge Regression

```
In [6]: ybar=np.mean(y)
        ybar
Out[6]: 5.220000000000001
```

```
In [7]: x_normal=x-np.mean(x)
```

```
In [8]: RidgeCoeff1=normal_equation(x_normal,y,1)
        RidgeCoeff1
Out[8]: array([[-0.60168365]])
```

```
In [9]: RidgeCoeff10=normal_equation(x_normal,y,10)
        RidgeCoeff10
Out[9]: array([[-0.30130907]])
```

Problem 3 - Gradient Decent and Newton's method

Consider solving the problem of locally weighted linear regression using gradient descent and Newton's method. Given data $\{\vec{x}^{(i)}, y^{(i)}\}$ for $i = 1, 2, ..., n$ and and a query point $\vec{x}$, we choose a parameter vector θ to minimize the loss function

$$J(\vec{\theta}; \vec{x}) = \sum_{i=1}^{n} w^{(i)} (\vec{\theta}^T \vec{x}^{(i)} - y^{(i)})^2$$

Here the weight function is $w^{(i)} = \exp\left(-\frac{\|\vec{x}^{(i)} - \vec{x}\|^2}{2\tau^2}\right)$ where τ is a hyper-parameter that must be tuned. Note that whenever we receive a new query point $\vec{x}$, we must solve the entire problem again with these new weights $w^{(i)}$.

(a) Given a data point $\vec{x}$, derive the gradient of $J(\vec{\theta}; \vec{x})$ with respect to $\vec{\theta}$.

Write $J(\vec{\theta}; \vec{x})$ using matrix multiplication as

$$J(\vec{\theta}; \vec{x}) = (X\vec{\theta} - \vec{y})^T W(X\vec{\theta} - \vec{y})$$

where $X = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & ... & x_d^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & ... & x_d^{(2)} \\ \vdots & \vdots & \vdots & ... & \vdots \\ 1 & x_1^{(n)} & x_2^{(n)} & ... & x_d^{(n)} \end{bmatrix}$, $\vec{y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix}$ and $W = \begin{bmatrix} w^{(1)} & & \\ & \ddots & \\ & & w^{(n)} \end{bmatrix}$

Using matrix calculus, we can calculate that (Write details.)

$$\nabla J = 2X^T W(X\vec{\theta} - \vec{y})$$

(b) Given a data point $\vec{x}$, derive the Hessian of $J(\vec{\theta}; \vec{x})$ with respect to $\vec{\theta}$.

The Hessian of $J(\vec{\theta}; \vec{x})$ is

$$H(J) = 2X^T WX$$

(c) Given a data point $\vec{x}$, write the update formula for gradient descent. Use the symbol η for an arbitrary step size.

$$\vec{\theta}^{(t+1)} = \vec{\theta}^{(t)} - \eta\nabla J = \theta^{(t)} + 2\eta X^T W(X\vec{\theta}^{(t)} - \vec{y})$$

(d) Given a data point $\vec{x}$, write the update formula for Newtons method.

$$\vec{\theta}^{(t+1)} = \vec{\theta}^{(t)} - H^{-1}\nabla J = \theta^{(t)} - (X^T WX)^{-1} X^T W(X\vec{\theta}^{(t)} - \vec{y})$$

Problem 4 - (stochastic) Gradient Decent

(1) The data file $\{\vec{x}^{(i)}, y^{(i)}\}$ for $i = 1, 2, ..., n$ is drawn (with noise) from

$$f(x) = \beta_0 + \beta_1 \sin(x) + \beta_2 \cos(x)$$

Can you solve the parameters use the least squares method? Find a closed formula and explain the matrices clearly in your formula.

The least squares solution is

$$\vec{\beta} = (X^T X)^{-1} X^T \vec{y}$$

where $X = \begin{bmatrix} 1 & \sin(\vec{x}^{(1)}) & \cos(\vec{x}^{(1)}) \\ 1 & \sin(\vec{x}^{(2)}) & \cos(\vec{x}^{(2)}) \\ \vdots & \vdots & \vdots \\ 1 & \sin(\vec{x}^{(n)}) & \cos(\vec{x}^{(n)}) \end{bmatrix}$ and $\vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}$

(2) The data file $\{\vec{x}^{(i)}, y^{(i)}\}$ for $i = 1, 2, ..., n = 10$ is drawn (with noise) from the function:

$$g(x) = \beta_0 + \sin(\beta_1 x) + \cos(\beta_2 x)$$

$x^{(i)}$	0	2	4	6	8	10	12	14	16	18
$y^{(i)}$	2.85	1.5	0.49	1.57	1.9	0.6	0.38	2.33	1.65	0.3

Use gradient decent(GD) or stochastic gradient decent (SGD) to fit the data to the function $g(x)$ by minimizing the RSS loss

$$RSS = \sum_{i=1}^{n} (y^{(i)} - g(x^{(i)}))^2$$

Turn in any associated computations, your learning rate, and the parameters.

We will use gradient descent to fit the data by minimizing RSS loss:

$$\frac{\partial}{\partial \beta_j} RSS = \frac{\partial}{\partial \beta_j} \left(\sum_{i=1}^{n} (y^{(i)} - g(x^{(i)}))^2 \right) \text{ for each } j = 0, 1, 2$$

$$= \sum_{i=1}^{n} \frac{\partial}{\partial \beta_j} (y^{(i)} - g(x^{(i)}))^2$$

$$= \sum_{i=1}^{n} \left(2(y^{(i)} - g(x^{(i)})) \cdot \frac{\partial}{\partial \beta_j} (y^{(i)} - g(x^{(i)})) \right) \text{ (by the chain rule)}$$

$$= 2 \sum_{i=1}^{n} \left((y^{(i)} - g(x^{(i)})) \cdot \frac{\partial}{\partial \beta_j} (y^{(i)} - \beta_0 - sin(\beta_1 x^{(i)}) - cos(\beta_2 x^{(i)}) \right)$$

$$\frac{\partial}{\partial \beta_0} RSS = 2 \sum_{i=1}^{n} \left((y^{(i)} - g(x^{(i)})) \cdot (-1) \right) = 2 \sum_{i=1}^{n} \left(g(x^{(i)}) - y^{(i)} \right)$$

$$\frac{\partial}{\partial \beta_1} RSS = 2 \sum_{i=1}^{n} \left((y^{(i)} - g(x^{(i)})) \cdot -\beta_1 cos(\beta_1 x^{(i)}) \right)$$

$$\frac{\partial}{\partial \beta_2} RSS = 2 \sum_{i=1}^{n} \left((y^{(i)} - g(x^{(i)})) \cdot \beta_2 sin(\beta_2 x^{(i)}) \right)$$

Plug into the update formula for gradient descent:

$$\beta_0^{k+1} = \beta_0^k - \alpha \left(2 \sum_{i=1}^{n} \left(g(x^{(i)}) - y^{(i)} \right) \right)$$

$$\beta_1^{k+1} = \beta_1^k + \alpha \left(2\beta_1^k \sum_{i=1}^{n} \left((y^{(i)} - g(x^{(i)})) \cdot cos(\beta_1^k x^{(i)}) \right) \right)$$

$$\beta_2^{k+1} = \beta_2^k - \alpha \left(2\beta_2^k \sum_{i=1}^{n} \left((y^{(i)} - g(x^{(i)})) \cdot sin(\beta_2^k x^{(i)}) \right) \right)$$

Using different initial value, different learning rate α, different number iterations, you may get different result. You should try different learning late (first, 1000, 100, 10, 1, 0.1, 0.01, 0.001, ..., and recording the cost function value to see what is the best α. Once you determined a good one say 0.1, then try different scales like 0.01, 0.02, 0.03, 0.04, ..., 0.09, 0.1, 0.11, ...

The true model for the data is z = 2+np.sin(3*x1) + np.cos(4*x1), so your result should close to $(\beta_0, \beta_1, \beta_2)$ = (2, 1, 1)