Statistical and Machine Learning (01.113)
Homework 1

Adam Ilyas 1002010

February 12, 2019

1 Problem 1

Let $\theta \in \mathbb{R}^d$ be a fixed vector and θ_0 be a constant. Let $x \in \mathbb{R}^d$ be variable. Consider the hyperplane in $\mathbb{R}^d$ whose equation is given by $\langle \theta, x \rangle + \theta_0 = 0$. Given a point $y \in \mathbb{R}^d$, find the shortest distance from y to the hyperplane

Hint: Normalize θ, i.e. let $n = \frac{\theta}{||\theta||}$ and rewrite the equation of the hyperplane in terms of n.

We let $y - x_*$ be a vectore from the hyperplane to y where x_* is the point on the hyper plane that fulfills

$$\langle \theta, x \rangle + \theta_0 = 0 \;\rightarrow\; \theta^\intercal x = -\theta_0$$

Projecting the vector $y - x_*$ onto a vector ortogonal to the hyperplane will give us the shortest distance from hyperplane to y (θ is the normal of the hyperplane)

$$\begin{aligned}
(y - x^*) \cdot \frac{\theta}{||\theta||} &= (y - x_*)^\intercal \frac{\theta}{||\theta||} \\
&= \frac{1}{||\theta||}(y^\intercal \theta - x_*^\intercal \theta) \\
&= \frac{1}{||\theta||}(y^\intercal \theta - (-\theta_0)) \\
&= \frac{1}{||\theta||}(y^\intercal \theta + \theta_0)
\end{aligned}$$

Since $\theta^\intercal x = x^\intercal \theta$

2 Problem 2

A continuous random variable X is said to have the *standard normal distribution*, with mean $\mu = 0$ and variance $\sigma^2 = 1$, that is $X \sim N(0,1)$, if it has a probability density function (pdf) defined by

$$f_X(x) = \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}},\ x \in \mathbb{R}$$

Prove that

$$\int_{\mathbb{R}} f_X(x)dx = 1$$

Hint: Let $I = \int_{\mathbb{R}} e^{\frac{-x^2}{2}} dx$. Express I^2 as a double integral over $\mathbb{R}^2$ and convert to polar coordinates:
($r = \sqrt{x^2 + y^2}\ 0 \leq r \leq \infty$ and $\theta = tan^{-1}\frac{y}{x}$, $0 \leq \theta \leq 2\pi$)

$$\begin{aligned}
I^2 &= \int_{\mathbb{R}} e^{\frac{-x^2}{2}} dx \\
&= \int_{\mathbb{R}} e^{\frac{-x^2}{2}} dx \int_{\mathbb{R}} e^{\frac{-y^2}{2}} dy \\
&= \int_{\mathbb{R}} \int_{\mathbb{R}} e^{\frac{-(x^2+y^2)}{2}} dxdy \\
&= \int_{\mathbb{R}} \int_{\mathbb{R}} e^{\frac{-(x^2+y^2)}{2}} dxdy \\
&= \int_0^{2\pi} \int_0^{\infty} e^{\frac{-r^2}{2}} drd\theta \\
&= \int_0^{2\pi} -e^{\frac{-r^2}{2}} \Big|_0^{\infty} d\theta \\
&= \int_0^{2\pi} 0 - (-1) d\theta = 2\pi
\end{aligned}$$

$$\begin{aligned}
\int_{\mathbb{R}} f_x(x)dx &= \int_{\mathbb{R}} \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}} \\
&= \frac{1}{\sqrt{2\pi}} \int_{\mathbb{R}} e^{\frac{-x^2}{2}} \\
&= \frac{1}{\sqrt{2\pi}} I \\
&= \frac{1}{\sqrt{2\pi}} \sqrt{2\pi} = 1
\end{aligned}$$

3 Problem 3

Let X and Y be random variables with a joint normal distribution such that $\mathbb{E}[X] = 0 = \mathbb{E}[Y]$, $\mathbb{E}[X^2] = 1 = \mathbb{E}[Y^2]$, and the covariance $\mathbb{E}[XY] = \rho$ where $0 < |\rho| < 1$

(a) Write down the joint probability distribution $p(x, y)$ of X and Y. Let A be the covariance matrix of X and Y where

$$A = \begin{bmatrix} E[X^2] & E[XY] \\ E[YX] & E[Y^2] \end{bmatrix} = \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix}$$

$$A^{-1} = \frac{1}{1-p^2} \begin{bmatrix} 1 & -\rho \\ -\rho & 1 \end{bmatrix}$$

$$\begin{aligned} p(x,y) &= \frac{1}{\sqrt{detA}(\sqrt{2\pi})^d} \exp(-\frac{1}{2}\langle x-\mu, A^{-1}(x-\mu)\rangle) \\ &= \frac{1}{2\pi\sqrt{1-p^2}} \exp(-\frac{1}{2}\langle \begin{bmatrix} X \\ Y \end{bmatrix}, A^{-1} \begin{bmatrix} X \\ Y \end{bmatrix} \rangle) \\ &= \frac{1}{2\pi\sqrt{1-p^2}} \exp(-\frac{1}{2-2p^2}(X^2 - 2\rho XY + Y^2)) \end{aligned}$$

(b) Let B denote the inverse of the covariance matrix of $[X, Y]^T$. ($B = A^{-1}$) Perform the decomposition $B = PDP^{-1}$, where D is a diagonal matrix and P is an orthogonal matrix.

$$A^{-1} = \frac{1}{1-p^2} \begin{bmatrix} 1 & -\rho \\ -\rho & 1 \end{bmatrix} = \frac{1}{1-p^2} PDP^{-1}$$

1. **Find eigenvalues of B** ($\det(B - \lambda I) = 0$)

$$\begin{vmatrix} 1-\lambda & -\rho \\ -\rho & 1-\lambda \end{vmatrix} = 0,\ (1-\lambda)^2 - \rho^2 = 0$$

From this, $1 - \lambda_1 = \rho$ and $1 - \lambda_2 = -\rho$

Solving for $\lambda_1 = 1 - \rho$: (Ensure unit vector)

$$\begin{bmatrix} \rho & -\rho \\ -\rho & \rho \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0, \vec{v_1} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$$

Solving for $\lambda_2 = 1 + \rho$: (Ensure unit vector)

$$\begin{bmatrix} -\rho & -\rho \\ -\rho & -\rho \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0, \vec{v_2} = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \Rightarrow \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}$$

2. **Construct P from eigenvectors**
$P = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix}$, $P^{-1} = \begin{bmatrix} 1/2 & 1/2 \\ 1/2 & -1/2 \end{bmatrix}$

3. **Construct D from eigenvalues** $D = \begin{bmatrix} 1-\rho & 0 \\ 0 & 1+\rho \end{bmatrix}$

Decomposition:

$$B = \frac{1}{1-p^2} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix} \begin{bmatrix} 1-\rho & 0 \\ 0 & 1+\rho \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix}$$

(c) Use the result above to transform $(x, y) \to (u, v)$ such that under the new coordinates, the joint distribution can be factorized; i.e. $q(u, v) = q_1(u)q_2(v)$.

$$p(x,y) = \frac{1}{2\pi\sqrt{1-p^2}} \exp(-\frac{1}{2}\langle \begin{bmatrix} X \\ Y \end{bmatrix}, PDP^{-1} \begin{bmatrix} X \\ Y \end{bmatrix} \rangle)$$

$$k \begin{bmatrix} x \\ y \end{bmatrix} = P^{-1} \begin{bmatrix} u \\ v \end{bmatrix} \Rightarrow \begin{bmatrix} u \\ v \end{bmatrix} = kP \begin{bmatrix} x \\ y \end{bmatrix} \text{ k is any constant}$$

$$\begin{bmatrix} u \\ v \end{bmatrix} = k \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Sub the following into $p(x, y)$

$$\begin{bmatrix} x+y \\ x-y \end{bmatrix} = \begin{bmatrix} u \\ v \end{bmatrix}$$

$$\begin{aligned}
p(x,y) &= \frac{1}{2\pi\sqrt{1-p^2}} \exp(-\frac{1}{2}\langle \begin{bmatrix} x \\ y \end{bmatrix}, PDP^{-1} \begin{bmatrix} x \\ y \end{bmatrix} \rangle) \\
&= \frac{1}{2\pi\sqrt{1-p^2}} \exp(-\frac{1}{2}\langle \begin{bmatrix} u-v \\ u+v \end{bmatrix}, \frac{1}{1-p^2} \begin{bmatrix} 1 & -\rho \\ -\rho & 1 \end{bmatrix} \begin{bmatrix} u-v \\ u+v \end{bmatrix} \rangle) \\
&= \frac{1}{2\pi\sqrt{1-p^2}} \exp(-\frac{1}{2-p^2}\langle \begin{bmatrix} u-v \\ u+v \end{bmatrix}, \begin{bmatrix} u-v-up-vp \\ u+v-up+vp \end{bmatrix} \rangle) \\
&= \frac{1}{2\pi\sqrt{1-p^2}} \exp(-u^2+u)\exp(-v^2-v) = q(u,v)
\end{aligned}$$

$$q_1(u) = a \cdot q(u)$$

$$q_2(u) = b \cdot q(v) \text{ where } ab = \frac{1}{2\pi\sqrt{1-p^2}}$$

4 Problem 4

We will now use PyTorch to perform linear regression using gradient descent. Import the Boston data from *sklearn* datasets to generate a linear model that predicts the prices of houses (MEDV) using three inputs:

(i) average number of rooms per dwelling (RM);1

(i) index of accessibility to radial highways (RAD);

(i) per capita crime rate by town (CRIM).

The data contains 506 observations on housing prices for Boston suburbs. The first three columns corresponds to the inputs RM, RAD and CRIM , respectively. The last column is the target MEDV .

(a) Write the code to generate (random) weights w_{RM} , w_{RAD} , w_{CRIM} and bias b.

```
theta = torch.randn((d+1,1))
```

After that, write a function to compute the linear model. Recall that Whenever the matrix $\mathbf{X}^T\mathbf{X}$ is invertible, the optimal weight which minimizes the empirical risk (MSE), is obtained as

$$\theta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

```
def optimal_weight(X,y, bias=True):
    """
    Using invertible matrix method
    X: (N, d) matrix
    y: (d, 1) column vector
    Returns:
        tensor of bias and weigths
    """
    X, y = make_tensor(X,y)
    if bias:
        X = add_ones(X)

    inv = torch.inverse(X.t()@X)
    w_opt = inv @ X.t() @ y
    return w_opt.reshape(-1,1)
```

Optimal Values:
$Bias$ = -27.1 $w_{RM} = 8.24$, $w_{RAD} = -0.17$, $w_{CRIM} = -0.16$

(b) Write a function that computes the mean squared error (MSE).

```
def calc_cost(X, y, theta):
    """
    X: (N, d) tensor
    y: (N, 1) tensor
    theta: (d, 1) or (d+1, 1) tensor

    Returns Mean Squared Error
    """
    if theta.shape[0]-1 == X.shape[1]:
        X = add_ones(X) # concat a column of ones
    y_pred = X@theta
    return ((y_pred - y)**2).sum()/N
```

(c) Complete the loop below to update the weights and bias using a fixed learning rate (try different values from 0.01 to 0.0001) over 200 iterations/epochs.

alpha	0.008	0.005	0.001	0.0005	0.0001
Mean Squared Error	**46.429**	46.543	46.91	50.64	156.564
Bias	1.215	1.399	1.63	1.609	1.408
Theta 1	3.941	3.913	3.776	3.426	1.928
Theta 2	-0.255	-0.255	-0.199	-0.022	0.112
Theta 3	-0.197	-0.197	-0.231	-0.312	0.396

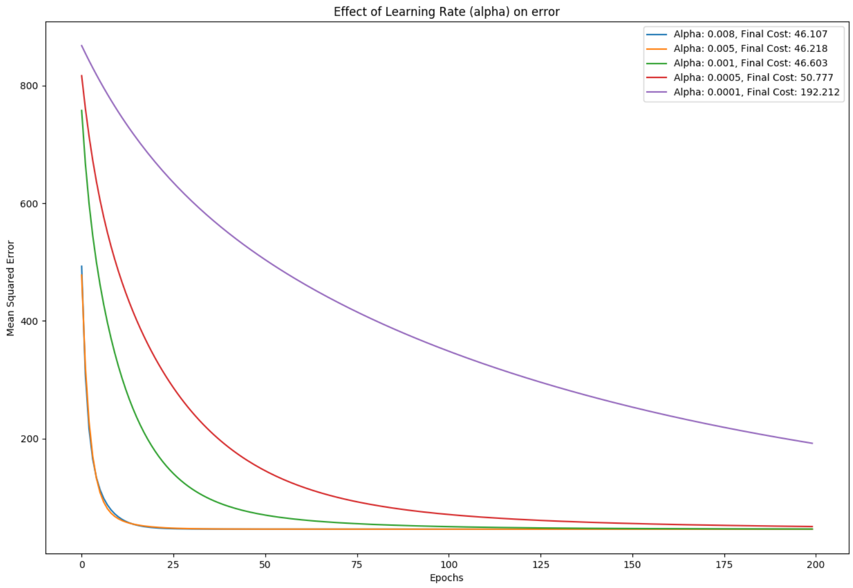
Below we observed that 0.008 is the best learning rate. However, any larger, the algorithm would not converge and we would get further and further away from a minima. A smaller rate means that we would reach the minima slower, but we would reach nonetheless (with more iterations/epochs)

Note The update theta formula that was used is the following

$$\theta_j(t+1) = \theta_j(t) - \alpha \frac{1}{N} \sum_{i=1}^{N} (X^\intercal X\theta(t) - X^\intercal y)$$

where N is the number of data points, hence effective scaling down the learning rate (alpha) to allow for convergence.

Plot of the evolution of MSE at every iteration below




```
def update_theta(X,y,theta, alpha):
    gradient = X.t() @ (X@theta - y)
    theta_new = (theta - alpha*(gradient)/N)
    return theta_new

def batch_gradient_descent(X,y,theta,alpha=0.1,max_iter=200):
    """
    X: (N, d) matrix (iterable)
    y: (N, 1) column vector (iterable)
    theta: (d,1) or (d+1,1) column vector (iterable)
    alpha: learning rate (float)
    max_iter: no. of epoch (int)

    Returns:
        theta: calculated bias and weights
        cost_history: list containing losses over each epoch
        theta_history: list containing theta over each epoch
    """
    X, y = tensor(X), tensor(y)
    assert X.shape[0] == y.shape[0], "Dimensions␣must␣fit"
    if theta.shape[0]-1 == X.shape[1]:
        X = add_ones(X)
```

```
    N, d = X.shape
    theta_history = []
    cost_history = []
    for i in range(max_iter):
        print(f"Epoch:␣{i}")
        theta = update_theta(X,y,theta, alpha)
        cost = calc_cost(X,y,theta)
        print (f"Loss=␣{cost}")
        theta_history.append(theta)
        cost_history.append(cost)

    return theta, cost_history, theta_history

def add_ones(X):
    """
    Add a column of ones at the left hand side of matrix X
    X: (N, d) tensor
    Returns
        (N, d+1) tensor
    """
    ones = torch.ones((X.shape[0],1), dtype=torch.float32)
    X = torch.cat((ones, X), dim=-1)
    return X

def make_tensor(*args):
    """
    Check if arguments are tensor, converts arguments to
        tensor
    accepts and returns Iterables
    """
    for el in args:
        if not torch.is_tensor(el):
            el = tensor(el)
    return args
```