jgiebas_HW2

January 29, 2018

1 46-926, Statistical Machine Learning 1: Homework 2

Author : Jordan Giebas *Due Date*: January 28th, 2018

1.1 Question 1

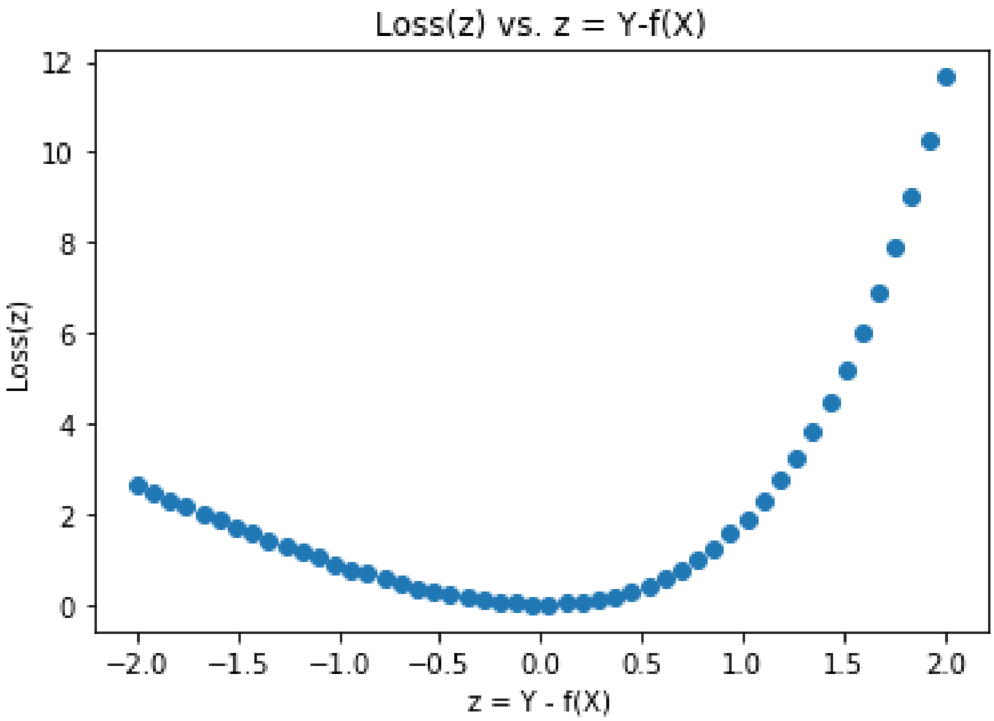
1.1.1 Part (a)

Plot the loss function given by,

$$\mathcal{L}(Y, f(X)) = b\big(e^{a(Y-f(X)} - a(Y - f(X)) - 1\big)$$

for $z = Y - f(X)$, $z \in [-2, 2]$, and where $(a, b) = (1.1, 2)$.

```
In [5]: import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline

        # Define the loss function
        def loss_f(z):

            return ( 2*(np.exp(1.1*z) - 1.1*z - 1) )

        # Vectorize the function to perform element wise
        v_loss_f = np.vectorize(loss_f)

        z_axis = np.linspace(-2, 2)
        y_axis = v_loss_f(z_axis)


        plt.xlabel("z = Y - f(X)")
        plt.ylabel("Loss(z)")
        plt.title("Loss(z) vs. z = Y-f(X)")
        plt.scatter(z_axis,y_axis)

Out[5]: <matplotlib.collections.PathCollection at 0x108adc208>
```



Description: The loss function is interesting. It is sensible that when z is close to zero, that the loss function is too. The interesting observation is the extremeties. When your prediction $f(X)$ highly underestimating Y, the loss function is much greater than when your prediction $f(X)$ is overestimating Y. **Usage**: When you want to punish the algorithm for underestimating the true value severly, but only marginally punish the algo for overestimating, this is a very good likelihood function to use.

1.1.2 Part (b)

Determine the f(X) that will minimize the expected loss, using the loss function provided above.

Following the notes from class, we first use iterated conditioning and condition on a particular value of X. Then, optimize by solving the derivative with respect to $f(x)$ equal to zero.

$$\mathbb{E}(\mathcal{L}(Y, f(X))) = \mathbb{E}\big[\mathbb{E}\big[\mathcal{L}(Y, f(X))|X = x\big]\big]$$

Focusing only on the inside conditional expectation and plugging in $\mathcal{L}(Y, f(X))$,

$$\mathbb{E}\big[b\big(e^{a(Y-f(X)} - a(Y - f(X)) - 1\big)|X = x\big]$$

$$b\mathbb{E}\big[\big(e^{a(Y-f(X)} - a(Y - f(X)) - 1\big)|X = x\big] = b\big(-a\mathbb{E}(Y|X = x) - 1 + \mathbb{E}(e^{a(Y-f(X))}|X = x) + a\mathbb{E}(f(X)|X = x)\big)$$

Luckily the exponential function is a homomorphism and we can separate the arguments then condition on $e^{f(X)}|X = x$. All together we see,

$$b\big(-a\mathbb{E}(Y|X = x) - 1 + e^{-af(x)}\mathbb{E}(e^{aY}|X = x) + af(x)\big)$$

Differentiating the inside with respect to $f(X)$, setting equal to zero, and solving, yields the following result for $f(x)$:

$$f(x) = \frac{lg\left(\mathbb{E}(e^{aY}|X = x)\right)}{a}$$

1.1.3 Part(c)

If the conditional distribution $Y|X = x\ N(\beta x, \sigma^2)$, then what is the form of the optimal estimator $f(x)$?

Since we know the conditional distribtuion, we can evaluate the conditional expectation in the final result of **Part(b)**. We recognize that this is the moment generating function for Y, evaluated at $t = a$. Hence,

$$f(x) = \frac{lg\left(e^{a\beta x + \frac{1}{2}a\sigma^2}\right)}{a} = \beta x + \frac{a\sigma^2}{2}$$

1.1.4 Part(d)

For part(d), I copy and paste the code from `asymm_loss.py` into the following cell and run, for ease in having everything local in the jupyter notebook (although it is possible to call the function in a python file from within this jupyter notebook.

In [11]:
```
from scipy.stats import norm
import numpy as np

#Set some parameters
beta = 0.5
b = 2
sigma = 2
a = 1

#Define the loss function, where z = y - yhat
def loss(z):
    return b*(np.exp(a*z)-a*z-1)

#Estimation functions
#Estimation using the conditional expectation of Y|X
def f_condexp(x):
    return beta*x

# TODO: Put your function in here.  You can reference a,b,sigma, and it will just pul
# the outside namespace
def f_yours(x):
    return beta*x + a*(sigma**2)/2.0

#Simulation to see how you do
reps = 1000
```

```
    #Just generate the X variables normally.  We don't really care
    x = norm.rvs(size=reps, loc=0, scale=1)

    #Generate the Y variables from our normal model
    y = norm.rvs(size=reps, loc=x*beta, scale=sigma)

    #Calculate the fitted values for each method
    yhat_condexp = np.apply_along_axis(f_condexp, 0, x)
    yhat_yours = np.apply_along_axis(f_yours, 0, x)

    #Compute the losses
    condexp_losses = np.apply_along_axis(loss, 0, y-yhat_condexp)
    your_losses = np.apply_along_axis(loss, 0, y-yhat_yours)

    print("Average loss of the conditional expectation:",
          round(np.mean(condexp_losses),2))

    print("Average loss of your method:",
          round(np.mean(your_losses),2))
```

```
Average loss of the conditional expectation: 13.27
Average loss of your method: 3.98
```

The average loss of the estimator derived in Part(c) is substantially lower than the average loss of the conditional expectation. In terms of bias-variance tradeoff, we are introducing some bias but this is marginal relative to the amount of variance reduction we receive. Hence, we see a lower expected loss.

1.2 Question 2

1.2.1 Part (a)

The ridge regression coefficients are given by (pardon my leaving out the hat on β, it doesn't look right when I compile the latex in the jupyter notebook),

$$\beta^{ridge} = \arg\max_{\beta \in \mathbb{R}^p} ||y - X\beta||_2^2 + \lambda||\beta||_2^2$$

Where $y \in \mathbb{R}^n$ is our response vector, $X \in \mathbb{R}^{nxp}$ is our prediction matrix, and $\beta \in \mathbb{R}^p$ is our coefficient vector.

We follow the structure of the problem, we have block matrices and vectors given as $\widetilde{y} = \begin{bmatrix} y \\ 0 \end{bmatrix} \in \mathbb{R}^{n+p}$, $\widetilde{X} = \begin{bmatrix} X \\ \sqrt{\lambda} I \end{bmatrix} \in \mathbb{R}^{(n+p)xp}$ where $I \in \mathbb{R}^{pxp}$ is the identity matrix.

We know from basic linear regression (OLS) that the coefficients of regressing an observation vector onto the predicition matrix is given by

$$\beta^{OLS} = \left(X^T X\right)^{-1} X^T y$$

Regressing our block response vector on our block prediction matrix yields the following,

$$\beta = \left(\widetilde{X}^T\widetilde{X}\right)^{-1}\widetilde{X}^T\widetilde{y}$$

Performing a little matrix multiplication simplifies the result. Consider the following,

$$\widetilde{X}^T\widetilde{X} = \begin{bmatrix} X \\ \sqrt{\lambda} I \end{bmatrix}^T \begin{bmatrix} X \\ \sqrt{\lambda} I \end{bmatrix} = \begin{bmatrix} X^T & \sqrt{\lambda} I \end{bmatrix} \begin{bmatrix} X \\ \sqrt{\lambda} I \end{bmatrix} = X^T X - \lambda I$$

Also,

$$\widetilde{X}^T\widetilde{y} = \begin{bmatrix} X^T & \sqrt{\lambda} I \end{bmatrix} \begin{bmatrix} y \\ 0 \end{bmatrix} = X^T X - \lambda I\vec{0} = X^T X$$

Hence, the beta from regressing the block response vector on the block predicition vector takes the form

$$\beta = (X^T X + \lambda I)^{-1} X^T y$$

which as we saw, is precisely the solution to the ridge regression problem stated above.

1.2.2 Part (b)

We assume that $\lambda > 0$ so that $\sqrt{\lambda}$ is well defined. We would like to prove that $\widetilde{X}$ must have full column-rank. It suffices to show that each of the columns of $\widetilde{X}$ are linearly independent. Firstly, let's write out the entries of $\widetilde{X}$.

$$\widetilde{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1p} \\ x_{21} & x_{22} & \dots & x_{2p} \\ \vdots & \vdots & \dots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{np} \\ \sqrt{\lambda} & 0 & \dots & 0 \\ 0 & \sqrt{\lambda} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & \sqrt{\lambda} \end{bmatrix}$$

Now define the collection of vectors $\{\vec{v}_i\}_{i=1}^p$ as the columns of the $\widetilde{X}$, where each $\vec{v}_i \in \mathbb{R}^{n+p}$.

We would like to show that this collection of vectors is linearly independent. Hence, we must show that for any linear combination equal to zero, $\sum_{i=1}^p \alpha_i \vec{v}_i = \vec{0}$, that this implies $\alpha_i = \vec{0}, \forall i$.

Consider the following linear combination set equal to zero,

$$\sum_{i=1}^p \alpha_i \vec{v}_i = \alpha_1 \begin{bmatrix} x_{11} \\ \vdots \\ x_{n1} \\ \vdots \\ \sqrt{\lambda} \\ \vdots \\ 0 \\ 0 \end{bmatrix} + \alpha_2 \begin{bmatrix} x_{12} \\ \vdots \\ x_{n2} \\ \vdots \\ 0 \\ \sqrt{\lambda} \\ \vdots \\ 0 \end{bmatrix} + \cdots + \alpha_p \begin{bmatrix} x_{1p} \\ \vdots \\ x_{np} \\ \vdots \\ 0 \\ 0 \\ \vdots \\ \sqrt{\lambda} \end{bmatrix} = \begin{bmatrix} \alpha_1 x_{11} \\ \vdots \\ \alpha_1 x_{n1} \\ \vdots \\ \alpha_1\sqrt{\lambda} \\ \vdots \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} \alpha_2 x_{12} \\ \vdots \\ \alpha_2 x_{n2} \\ \vdots \\ 0 \\ \alpha_2\sqrt{\lambda} \\ \vdots \\ 0 \end{bmatrix} + \cdots + \begin{bmatrix} \alpha_p x_{1p} \\ \vdots \\ \alpha_p x_{np} \\ \vdots \\ 0 \\ 0 \\ \vdots \\ \alpha_p\sqrt{\lambda} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

This is clearly a system of equations, but let's focus our attention on the $p^{th}, (p+1)^{th}, ..., (n+p)^{th}$ rows. Solving each of these equations clearly shows that $\{\alpha_i\}_{i=1}^{p} = 0, \forall i$. Hence, we conclude that the columns of $\widetilde{X}$ are linearly independent, and it follows immediately that $\widetilde{X}$ has full column-rank.

1.2.3 Part (c)

In Part(a) we confirmed that for $I \in \mathbb{R}^{pxp}$, the β^{ridge} is given by (again pardon the omissino of the hat),

$$\beta^{ridge} = (X^T X + \lambda I)^{-1} X^T y$$

Hence,

$$a^T \beta^{ridge} = a^T \left[(X^T X + \lambda I)^{-1} X^T y \right]$$

By associativity,

$$a^T \beta^{ridge} = \left[a^T (X^T X + \lambda I)^{-1} X^T \right] y$$

I'm not sure how much detail is needed, I see it two ways. The first is that this is clearly a linear system in y because matrix multiplication is a linear transformation. The second is that, we're told that $a \in \mathbb{R}^p$ as well. Hence, $a^T \in \mathbb{R}^{1xp}$. It follows that, $\left[a^T (X^T X + \lambda I)^{-1} X^T \right] \in \mathbb{R}^{1xn}$. Hence, when this quantity is multiplied by $y \in \mathbb{R}^n$, we will get a scalar quantity. I'm not sure exactly what more is being asked, hopefully this is sufficient.

1.3 Question 3

Submitted as a separate .Rmd file - there is a problem installing `rpy2` with the new OS "macOS High Sierra" if you have R –version 3.4+, so I can't simply put this all here. (It's a problem regarding clang and g++ compilers from the C-API underneath the hood for Python to R).