Name :- Prateek Mahajan

EE511 : Homework 2

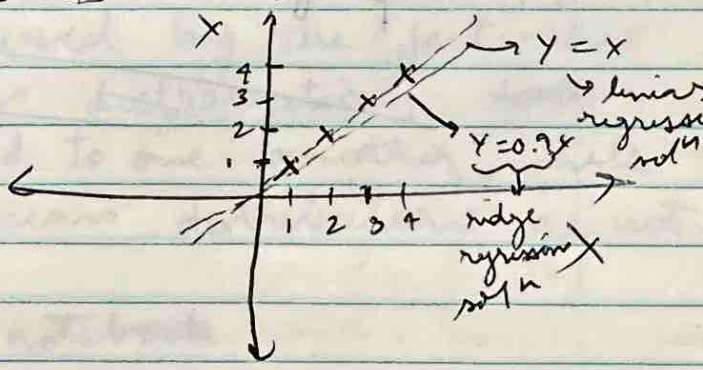
Problem 1 :-

1a) I have read and understood the general instructions ~~for HW2~~ at the top of HW2 and I formally declare that all the work I turn in for everything in this course will not contain or involve any cheating at all.

Problem 2 :-

2a) Consider a simple dataset as below :-

X	Y
1	1
2	2
3	3
4	4
⋮	⋮



For a dataset like the above, linear regression would give a sol^n of $Y = X$ with zero error

For ridge regression :- (setting $\lambda = 0.4$ / $\eta = 0.6$ & and assuming 4 datapoints

Let's assume that we get $Y = 0.9X$

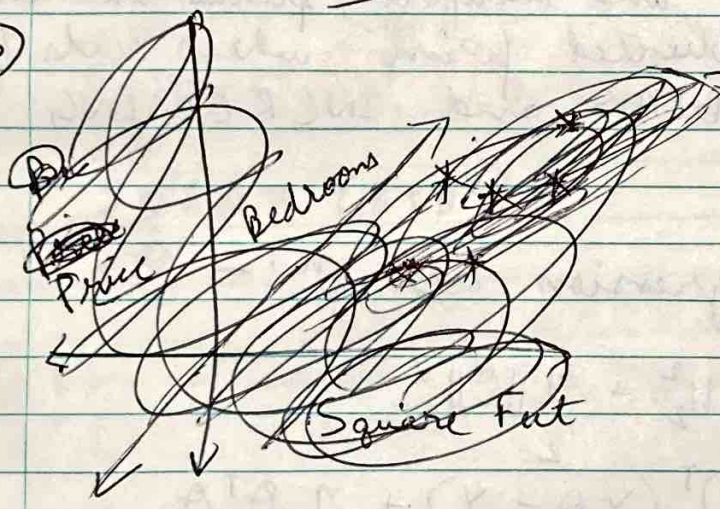
Using the ridge regression sol^n, Error $(Y = X) = 4$,
Error $(Y = 0.9X) = 3.9$

so clearly, it would be preferred despite $Y = X$

fitting the data better

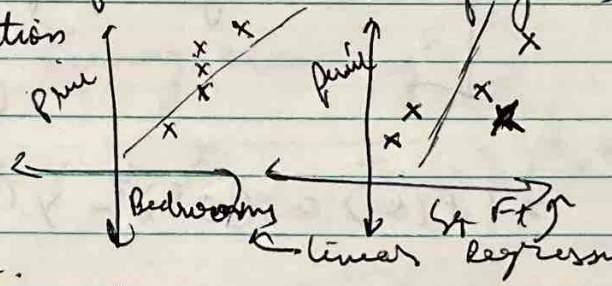
- in cases like the above, where there is only 1 feature or there is NO correlation amongst features, the linear regression is better than ridge. The reason for this is that ridge regression adds a bias that makes it good at dealing with collinearity amongst various features but makes it perform worse when the focus is on interpreting the individual effects of features & when there isn't collinearity amongst them.

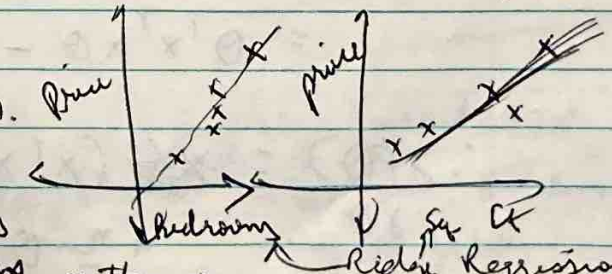
b)



Price	Bedrooms	Sq. ft.
400	3	2104
330	3	1600
369	3	2400
232	2	1416
540	4	3000
⋮	⋮	⋮

In the given example, there are 2 correlated features (no. of bedrooms & sq. feet). The addition of a regularization term adds a bias to the model that helps it better represent the correlation b/w sq. ft & no. of bedrooms than the standard linear regression. Here, we are not interested in the combined effect of the features than just their individual contributions, as they are correlated ∴ we prefer ridge regression.





b) ~~Increasing η modifies the optimization sol^n by increasing regularization and therefore, increases bias & reduces variance.~~

c) Increasing η tries to push the parameters of the optimisation problem to zero thereby simplifying the model and INCREASING BIAS and DECREASING VARIANCE by pushing the model to a more general form.

Decreasing η tries to give more importance to square error, and therefore, pushes the model to a more complicated form, which leads to DECREASING BIAS and INCREASING VARIANCE //

d) Ridge regression cost f^n :-

$$F(\theta) = \|X\theta - y\|_2^2 + \frac{\eta}{2}\|\theta\|_2^2$$

$$= (X\theta - Y)^T \cdot (X\theta - Y) + \frac{\eta}{2}\theta^T\theta$$

[in matrix form, if $X \to n \times m$ design matrix
$Y \to n \times 1$ output matrix
$\theta \to m \times 1$ parameter/feature matrix]

$$\Rightarrow F(\theta) = (\theta^T X^T - Y^T)(X\theta - Y) + \frac{\eta}{2}\theta^T\theta$$

$$= \theta^T X^T X\theta - Y^T X\theta - \theta^T X^T Y + Y^T Y + \eta/2\,\theta^T\theta$$

$$\therefore \frac{\partial F(\theta)}{\partial \theta} = (X^T X + X^T X)\theta - X^T Y - X^T Y + 0 + \frac{\eta}{2}\cdot\theta[I + I]$$

$$\Rightarrow \frac{\partial F(\theta)}{\partial \theta} = 2X^TX\theta - 2X^TY + \eta\theta$$

Setting $\frac{\partial F(\theta)}{\partial \theta} = 0 \Rightarrow 2X^TX\theta + \eta\theta = 2X^TY$

$$\Rightarrow \left(X^TX + \frac{\eta I}{2}\right).\theta = X^TY$$

$$\Rightarrow \tilde{\theta}_{optimal} = \left(X^TX + \frac{\eta I}{2}\right)^{-1}.X^TY$$

closed form soln of ridge regression

Formulas used:-

→ $\frac{\partial b^T\theta^T\theta c}{\partial \theta} = \theta(bc^T + cb^T)$

→ $\frac{\partial \theta^T a}{\partial \theta} = \frac{\partial a^T\theta}{\partial \theta} = a$

→ $\frac{\partial a^T\theta b}{\partial \theta} = ab^T$

→ $\frac{\partial \theta^T B\theta}{\partial \theta} = (B + B^T)\theta$

e) For a vanilla linear regression

1) $F(\theta) = ||X\theta - y||_2^2 = (X\theta - Y)^T.(X\theta - Y)$

$= (\theta^TX^T - Y^T).(X\theta - Y)$

$= \theta^TX^TX\theta - Y^TX\theta - \theta^TX^TY + Y^TY$

, where $X \rightarrow n \times m$ design matrix, $\theta \rightarrow m \times 1$ matrix, $Y \rightarrow n \times 1$ matrix

$\therefore \frac{\partial F}{\partial \theta} = 2X^TX\theta - 2X^TY$ (using above eqns)

$\Rightarrow \frac{\partial F}{\partial \theta} = 0 \Rightarrow X^TX\theta = X^TY \Rightarrow \tilde{\theta}_{optimal} = (X^TX)^{-1}X^TY$

$X^TX \rightarrow m \times m$ matrix

If $m > n$ and features of X are highly correlated, X^TX may NOT be invertible. (since the determinant may be zero)

X^TX is (if not invertible)

In that case, the $\tilde{\theta}$ optimal / closed form sol^n of the vanilla linear regression may not be computable.

2) Comparing the vanilla linear regression closed form sol^n to ridge regression's :-
the matrix to be inverted in ridge regression is $(X^TX + \lambda I)$, not X^TX.
∴ as long as we make sure $\lambda > 0$, $(X^TX + \lambda I)$ is always invertible ⇒ the closed form sol^n for ridge regression always exists (which is the benefit of ridge regression)

f)

(i) $F(\theta, \theta_0) = \|X\theta + \theta_0 1 - y\|_2^2 + \frac{\eta}{2}\|\theta\|_2^2$

$= (X\theta + \theta_0 1 - y)^T (X\theta + \theta_0 1 - y) + \frac{\eta}{2}\theta^T\theta$

$= (\theta^T X^T + 1^T\theta_0^T - y^T)(X\theta + \theta_0 1 - y) + \frac{\eta}{2}\theta^T\theta$

$= \theta^T X^T X\theta + \theta^T X^T \theta_0 1 - \theta^T X^T y + 1^T\theta_0^T X\theta + 1^T\theta_0^T\theta_0 1 - 1^T\theta_0^T y - y^T X\theta - y^T\theta_0 1 + y^Ty + \frac{\eta}{2}\theta^T\theta$

$\therefore \frac{\partial F(\theta, \theta_0)}{\partial \theta_0} = 0 + 1^T X\theta - 0 + 1^T\theta_0(1 + 1) - 1^T y - 0 - 1^T y + 0 + 0 + 1^T X\theta$

$\Rightarrow \frac{\partial F(\theta, \theta_0)}{\partial \theta_0} = 2X\theta 1^T + 2\theta_0 1 1^T - 2y1^T$

$= 2\,1^T X\theta + 2\,1^T\theta_0 1 - 2\,1^T y$

Setting $\frac{\partial F(\theta, \theta_0)}{\partial \theta_0} = 0$

$\Rightarrow 1^T y = 1^T X\theta + 1\theta_0 1^T$ — (1)

$\Rightarrow y = X\theta + \theta_0 1$ — (1)

$\Rightarrow y = X\theta + \theta_0 \cdot 1$ — (1)

Now consider $\frac{\partial F(\theta, \theta_0)}{\partial \theta}$

$\therefore \frac{\partial F(\theta, \theta_0)}{\partial \theta} = 2X^T X\theta + X^T \theta_0 1 - X^T y + X^T \theta_0 1 + 0 - 0$
$- X^T y - 0 + 0 + \eta\theta$

$\Rightarrow \frac{\partial F(\theta, \theta_0)}{\partial \theta} = 2X^T X\theta + 2X^T \theta_0 1 - 2X^T y + \eta\theta$

Setting $\frac{\partial F(\theta, \theta_0)}{\partial \theta} = 0$

$\Rightarrow 2X^T y = 2X^T X\theta + 2X^T \theta_0 1 + \eta\theta$

On substituting θ_0 :—

$2X^T y = 2X^T X\theta + 2X^T y - 2X^T X\theta + \eta\theta$

$2X^T y = 2X^T X\theta + 2X^T X\theta - 2X^T y + \eta\theta$

$\Rightarrow 4X^T y = 4X^T X\theta + \eta\theta$

$\Rightarrow 2X^T y 1^T = 2X^T X\theta 1^T + 2X^T (y 1^T - X\theta 1^T) + \eta\theta 1^T$

$\Rightarrow (2X^T X + \eta I) \cdot \theta = 2X^T y - 2X^T \theta_0 \cdot 1$

$\Rightarrow \theta = (X^T X + \frac{\eta}{2} I)^{-1} (X^T)(y - \theta_0 \cdot 1)$ — (2)

Taking θ_0 :—

$y \cdot 1^T = X\theta 1^T + \theta_0 1 1^T$

However, θ_0 is a scalar bias.

$\therefore \theta_0 1 1^T = \{\theta_0\}_{n \times n}$

$\therefore$ This eqn ~~is only satisfied if~~ can be interpreted as θ_0 optimal

Taking ① :—

$$1^T y = 1^T X\theta + 1^T \theta_0 1$$

$$\Rightarrow 1^T \theta_0 1 = 1^T y - 1^T X\theta$$

$$\Rightarrow \theta_0 1^T 1 = 1^T y - 1^T X\theta \quad — ③$$

$$\Rightarrow n\theta_0 = \sum_{i=1}^{n} y_i - \sum_{i=1}^{n}\sum_{j=1}^{m} x_{i,j}\,\theta_{j,1}$$

$$\Rightarrow \theta_0 = \frac{1}{n}\left(\sum_{i=1}^{n} y_i - \sum_{j=1}^{m} x_{i,j}\,\theta_{j,1}\right)$$

$$= \underline{\text{Mean of prediction error (say } \mu_e)}$$

Taking ② :—

$$\theta = (X^T X + \frac{n}{2} I)^{-1} X^T (y - \theta_0 1)$$

$$\Rightarrow \theta = (X^T X + \frac{n}{2} I)^{-1} X^T (y - \mu_e 1)$$

If we manipulate the eqn differently (i.e. substitute θ_0 before inverse) :

And $\theta_0 = \mu_y - \frac{1^T X \theta}{n}$ (from ③)

$$(X^T X + \frac{n}{2} I)\theta = X^T y + \frac{X^T 1^T X\theta 1}{n} - X^T \mu_y 1$$

$$\left[\mu_y = \frac{1}{n}\sum_{i=1}^{n} y_i\right]$$

Note that $(1^T X\theta)$ is a 1×1 result & so can be treated as a scalar

$$\Rightarrow (X^T X + \frac{n}{2} I)\theta = X^T y + \frac{X^T 1 (1^T X\theta)}{n} - X^T \mu_y 1$$

$$\Rightarrow (X^T X + \frac{n}{2} I - \frac{X^T 1 1^T X}{n})\theta = X^T (y - \mu_y 1)$$

$$\Rightarrow \theta = (X^T X + \frac{n}{2} I - \frac{X^T 1 1^T X}{n})^{-1} \cdot X^T (y - \mu_y 1)$$

NOTE :- 2f(ii) is after Problem 3b)

sorry about the confusion

~~model, and so regularising it would~~

Problem 3:-

a) Consider a gaussian noise linear least squares regression model, where $\vec{y} = X\theta + \vec{\epsilon}$, where X is a $n \times m$ design matrix & $\vec{\epsilon}$ is a length-n vector of gaussians, $\epsilon_i \in N(0, \sigma^2)$.
MLE parameter estimate $= \tilde{\theta} = (X^T X)^{-1} X^T \vec{y}$

Let θ^* be the "true" parameter estimate for the gaussian LLS model. $(\theta^* \in \mathbb{R}^{m \times 1})$

~~$\therefore \mathbb{E}(Y|x) = x\theta^*$ (x is 1 instance of X from a~~

$\therefore$ $\mathbb{E}(Y|x)$ = true prediction of model
(as $\mathbb{E}(Y|x)$ is the best fit soln) $= x^T\theta^*$, where x ~~is~~ is the instances of X from a dataset D. — (1)
($x \in \mathbb{R}^{m \times n}$)

~~$\mathbb{E}_D[h_D(x)]$~~

Consider $h_D(x) = x^T\tilde{\theta}$
$= x^T((X^TX)^{-1}X^T\vec{y})$
$= x^T(X^TX)^{-1}X^T(X\theta^* + \epsilon)$ (since $\vec{y} = X\theta^* + \epsilon$)
$= x^T\theta^* + x^T(X^TX)^{-1}X^T\epsilon$

$\therefore \mathbb{E}_D(h_D(x)) = \mathbb{E}_D(x^T\theta^* + x^T(X^TX)^{-1}X^T\epsilon)$
$= \mathbb{E}_D(x^T\theta^*) + \mathbb{E}_D(x^T(X^TX)^{-1}X^T\epsilon)$

x, X are fixed matrices which implies this term is $x^T(X^TX)^{-1}X^T \cdot \mathbb{E}_D(\epsilon) = 0$

$\therefore E_D(h_D(x)) = \mathbb{E}_D(x^T\theta^*)$
$= x^T\theta^*$ (as x, θ^* are not variables) — (2)

$\therefore$ (1) = (2) $\Rightarrow \mathbb{E}_D(h_D(x)) = \mathbb{E}(Y|x) \Rightarrow$ this model is unbiased

b) Consider the LHS $= \mathbb{E}_D[(h_D(x) - \mathbb{E}_D(h_D(x)))^2]$

$\therefore$ LHS $= \mathbb{E}_D[(h_D(x) - x^T\theta^*)^2]$

(Since $\mathbb{E}_D(h_D(x)) = x^T\theta^*$ as per 3 a))

$h_D(x) = x^T\hat{\theta}$

$= x^T(X^TX)^{-1}X^Ty$

$= x^T(X^TX)^{-1}X^T(X\theta^* + \epsilon)$

$= x^T(X^TX)^{-1}X^TX\theta^* + x^T(X^TX)^{-1}X^T\epsilon$

$= x^T\theta^* + x^T(X^TX)^{-1}X^T\epsilon$

$\therefore$ LHS $= \mathbb{E}_D((x^T\theta^* + x^T(X^TX)^{-1}X^T\epsilon - x^T\theta^*)^2)$

~~$= \mathbb{E}_D[(x(X^TX)^{-1}X^T\epsilon)^T(x(X^TX)^{-1}X^T\epsilon)]$~~

~~$= \mathbb{E}_D(x(X^TX)^{-1}X^T\epsilon\epsilon^TX(X^TX)^{-1}x^T)$~~

~~$= \mathbb{E}\, x(X^TX)^{-1}X^T\epsilon^TX(X^TX)^T$~~

~~this is~~

~~$= \mathbb{E}\, x(X^TX)^{-1}X^T\mathbb{E}(\epsilon\epsilon^T).X(X^TX)^{-1}x^T$~~

$\Rightarrow$ LHS $= \mathbb{E}_D[(x^T(X^TX)^{-1}X^T\epsilon)^2]$

$= \mathbb{E}_D[(x^T(X^TX)^{-1}X^T\epsilon)^T(x^T(X^TX)^{-1}X^T\epsilon)^T]$

$= \mathbb{E}_D[(x^T(X^TX)^{-1}X^T\epsilon)^T(x^T(X^TX)^{-1}X^T\epsilon)^T]$

$= \mathbb{E}_D[x^T(X^TX)^{-1}X^T\epsilon\epsilon^TX^T(X^TX)^{-1}]^Tx]$

x, X are not random variables

$\Rightarrow$ LHS $= \mathbb{E}[x^T(X^TX)^{-1}X^T.\mathbb{E}(\epsilon\epsilon^T).X^T(X^TX)^{-1}x]$

[Note:- (X^TX) is symmetric as it is a covariance matrix $\Rightarrow (X^TX)^T = X^TX$

$\Rightarrow ((X^TX)^T)^T = ((X^TX)^T)^{-1} = (X^TX)^{-1}$

Similarly, $\mathbb{E}(\epsilon\epsilon^T) = \sigma^2I$ (as ϵ is a GRV & $\epsilon\epsilon^T$ is a covariance matrix)]

~~LHS~~

$$\Rightarrow LHS = x^T (X^T X)^{-1} X^T (\sigma^2 I) X (X^T X)^{-1} x$$
$$= \sigma^2 x^T (X^T X)^{-1} X^T X (X^T X)^{-1} x$$
$$= \sigma^2 x^T (X^T X)^{-1} x = RHS$$

$\therefore$ LHS = RHS $\Rightarrow$ 3b) is correct

Problem 2:- (continued)

f)

(ii) $\theta_{\text{unregularised bias ridge}} = (X^T X + \frac{\eta}{2} I - \frac{X^T 1 1^T X}{n})^{-1} X^T (y - \mu_y 1)$

$\theta_{\text{normal ridge}} = (X^T X + \frac{\eta}{2} I)^{-1} X^T y$

On comparing these clearly, it looks like when we do not penalize the bias / offset / intercept parameter by regularising it, the resultant answer seems to center the "X" & "y" terms ~~more, and may~~ by subtracting their means from them. [μ_y = mean of y, $\frac{X^T 1 1^T X}{n}$ is kind of like the mean of $X^T X$ as we are essentially ~~not~~ adding a multiplicant of $\frac{1 1^T}{n}$ between $X^T X$, which is somewhat like that the average of $X^T X$]

$\therefore$ not penalizing the offset parameter "normalizes" ~~the effect of~~ X & Y ~~not~~ by subtracting their means

~~Further, bias is an inherent prope~~

Further bias is an inherent property of the model. The offset parameter is not responsible for changing the shape of the

model and so, regularising it would not prevent it from overfitting (or have any function whatsoever).

Problem 4:—

a) According to the scatter plot, LSTAT & RM are the most correlated to MEDV

b) According to the correlation matrix RM, LSTAT & RATIO are the most correlated to MEDV. The answer is different from 4a, as ~~we are trying to correlate house prices from the same dataset~~ the 3rd most correlated feature is not clear in the graphs.

c) Linear Regression:—
Closed form soln:— $\tilde{\theta} = (X^T X)^{-1} X^T y$

Ridge Regression:—
Closed form soln:— $\tilde{\theta} = (X^T X + \frac{\eta I}{2})^{-1} X^T y$

Please refer to the end of this pdf of the ipynb files for coefficient values (note that I used $\eta = 20.0$ for this problem)

d) Please refer to the .ipynb file attached at the end of this pdf for the final RMSE outputs I got for the linear & ridge regression.

Training RMSE:—
Linear < Ridge (linear is better)
This is because the ridge regression performs regularisation to reduce the tendency of the model to overfit to the training data and generalise it more (i.e. increase bias & reduce variance)

∴ since the linear regression model is trained to fit the training data better, it performs better on that data

Testing RMSE :-

Linear > Ridge (ridge is better)

The linear model does not regularise & so has higher variance than the ridge model, which regularises the cost fn to reduce the tendency of the model to overfit the training data. For this reason, the ridge model is more generalised & performs better on another dataset (i.e. the testing one)

e) Please refer to the .ipynb file for my RMSE outputs (NOTE $\eta = 20$)

On comparing the RMSE outputs of Top 3 features vs that of all 13, we can see that the RMSE is higher when we use just 3 features (albeit not significantly)

This indicates that while a large chunk of correlation with MEDV originates in these features, there are some other features in the model that have significance & should be ignored.

Problem 5:-

a) $F(\theta) = \frac{1}{n}\sum_{i=1}^{n} -\log\left[P(y=y_i \mid x=x_i, \theta)\right] + \frac{\eta}{2}\|\theta\|_2^2$

~~First let us consider~~

, where $x \in \mathbb{R}^{n \times m}$, $\theta \in \mathbb{R}^{m \times c}$

First, let us consider

$$P(y=y_i \mid x_i, \theta) = \frac{\exp(x \cdot \theta^{(k)})}{\sum_{j=1}^{c} \exp(x \cdot \theta^{(j)})}$$

(where $\theta^{(k)}$ is the k^{th} column of θ)

$$\Rightarrow \log\left(P(y=y_i \mid x_i, \theta)\right) = \underbrace{x\theta^{(k)}}_{\text{take this as } z^{(k)}} - \log\Big(\underbrace{\sum_{j=1}^{c} \exp(x \cdot \theta^{(j)})}_{\text{which implies that this is } z^{(j)}}\Big)$$

$$\therefore \frac{\partial}{\partial z^{(l)}}\left(\log(P(y=y_i \mid x_i, \theta))\right) = \frac{\partial(x\theta^{(k)})}{\partial z^{(l)}} - \frac{\partial}{\partial z^{(l)}}\left(\sum_{j=1}^{c} \exp(x \cdot \theta^{(j)})\right)$$

$$= \frac{\partial(z^{(k)})}{\partial z^{(l)}} - \frac{\partial}{\partial z^{(l)}}\left(\sum_{j=1}^{c} \exp(z^{(j)})\right)$$

(NOTE:- l is any value b/w 1 & c]

$$\frac{\partial z^{(k)}}{\partial z^{(l)}} = \begin{cases} 1, & \text{if } k = l \\ 0, & \text{otherwise} \end{cases}$$

$$\therefore \frac{\partial}{\partial z^{(l)}}\left(\log(P(y=y_i \mid x_i, \theta))\right) = 1\{k==l\} - \frac{1}{\sum_{j=1}^{c} \exp(z^{(j)})} \cdot \frac{\partial \sum_{j=1}^{c} e^{z^{(j)}}}{\partial z^{(l)}}$$

But $\frac{\partial}{\partial z^{(l)}} \sum_{j=1}^{c} e^{z^{(j)}} \simeq \frac{\partial}{\partial z^{(l)}} e^{z^{(l)}} = e^{z^{(l)}}$

(as all other terms will give zero)

(cs cont.)

$$\therefore \frac{\partial(\log(P(y=y_i|x_i,\theta)))}{\partial z^{(i)}} = 1\{k==l\} - \frac{e^{z^{(i)}}}{\sum_{j=1}^{c} \exp(z^{(i)})}$$

$$\Rightarrow \frac{\partial(\log(P(y=y_i|x_i,\theta)))}{\partial z^{(i)}} = 1\{k==l\} - p(y=l \mid x_i,\theta) \quad \text{---} (1)$$

$$\therefore \frac{\partial}{\partial\theta}(\log(P(y=y_i|x_i,\theta))) = \frac{\partial z^{(i)}}{\partial\theta} \cdot \frac{\partial}{\partial z^{(i)}}(\log(P(y=y_i|x_i,\theta)))$$

Consider $\frac{\partial z^{(i)}}{\partial\theta} = \frac{\partial(x\cdot\theta^{(i)})}{\partial\theta} = \frac{\partial \sum_{j=1}^{m} x_j \theta_j^{(i)}}{\partial\theta}$

$$= \begin{bmatrix} 0 & 0 & \cdots & 0 \\ x_1 & x_2 & \cdots & x_m \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

(where x is a single column)

$$\therefore \frac{\partial}{\partial\theta}(\log(P(y=y_i|x_i,\theta))) = \frac{\partial z^{(i)}}{\partial\theta} \cdot \frac{\partial}{\partial z^{(i)}}(\log(P(y=y_i|x_i,\theta)))$$

Consider $\frac{\partial z^{(i)}}{\partial\theta} = \frac{\partial(x\cdot\theta^{(i)})}{\partial\theta} = x$ (for a single column)

$$\therefore \frac{\partial}{\partial\theta}(\log(P(y=y_i|x_i,\theta))) = x_i\left(1\{k==l\} - p(y=l|x_i,\theta)\right)$$

For $F(\theta)$ we ~~[illegible]~~ sum this log probability for all n & classes.

$$\Rightarrow \frac{\partial F(\theta)}{\partial\theta} = \frac{\partial}{\partial\theta}\left[\frac{1}{n}\sum_{i=1}^{n} -\log(Pr(y=y_i|x_i,\theta)] + \frac{\eta}{2}\|\theta\|_2^2\right\}$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{c} x_i\left(1_{\{k==l\}} - Pr(y_i=j|x_i,\theta)\right)$$

$$+\frac{\eta\partial\theta^T\theta}{2\partial\theta}$$

$$\Rightarrow \frac{\partial F(\theta)}{\partial\theta} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{c} x_i\left(1_{\{k=l\}} - Pr(y_i=j|x_i,\theta)\right)$$

$$+\eta\theta$$

Gradient Descent Rule for θ :-

$$\theta \Leftarrow \theta - \alpha \nabla . \frac{\partial F(\theta)}{\partial \theta}$$

$$\Rightarrow \theta \Leftarrow \theta + \frac{\alpha}{n} \sum_{i=1}^{n} \sum_{j=1}^{c} x_i \left(1_{\{k=j\}} - P_\theta(y_i = j \mid x_i, \theta) \right)$$

$$- \alpha \eta \theta \quad \text{---} \quad (2)$$

For a matrix formulation :-

revert back to

$$\frac{\partial}{\partial \theta} \Rightarrow \frac{\partial (\log(P(y=y_i \mid x_i, \theta)))}{\partial \theta} = \frac{\partial z^{(i)}}{\partial \theta} \cdot \frac{\partial (\log(P(y=y_i \mid x_i \theta)))}{\partial z^{(i)}}$$

(consider $\frac{\partial z^{(i)}}{\partial \theta}$)

Branching it out further (to ∂z instead of $\partial z^{(i)}$)

$$\frac{\partial (\log(P(y=y_i \mid x_i, \theta)))}{\partial \theta} = \frac{\partial z}{\partial \theta} \cdot \frac{\partial}{\partial z} (\log(P(y=y_i \mid x_i, \theta)))$$

$$\frac{\partial z}{\partial \theta} \approx X^T \theta \quad (\text{where } z = x\theta)$$

Using θ (extrapolating θ) :-

$$\frac{\partial}{\partial z} (\log(P(y=y_i \mid x_i, \theta))) = Y - P,$$

where P is a $n \times c$ matrix, ~~containing the~~ where

~~$P_{i,j} = \exp(x_{i,j} \cdot \theta_{j,c})$~~

$$P_{i,k} = \frac{\exp(x_{i,j} \cdot \theta_{j,c})}{\sum_{l=1}^{c} \exp(x_{i,j} \cdot \theta_{j,c})}$$ and assuming Y is a 1 hot encoding matrix of size $n \times c$

~~the matrix for~~

$$\therefore \frac{\partial}{\partial \theta} (\log(P(y=y_i \mid x_i, \theta))) = X^T . (Y - P)$$

$$\therefore \frac{\partial F}{\partial \theta} \text{ (in matrix form)} = -\frac{1}{N} X^T (Y - P) + \eta \theta$$

The least mean squares gradient descent rule is :-

$$\theta \leftarrow \theta + \alpha \left(y^{(i)} - h_\theta(x^{(i)})\right)(x^{(i)})$$

On comparing ② to the above :-

clearly both are of the form
$\theta \leftarrow \theta' + \alpha \times$ error, where error is the deviation of the prediction from the original output in the data.

b) Please refer to the .ipynb file at the end of this pdf for this qn.

c) Based on the convergence curve, clearly as we increase the learning rate, the model gets trained faster / requires fewer training epochs.
However we also see that the final cost fn is lower & training / testing precision is higher to a certain point as we reduce the learning rate.
The above are the tradeoffs to consider.
However, clearly, 0.01 is the sweet spot of this model as it is the point where cost fn is lowest, precision is highest and training time is acceptable / not too long

d) Please refer to the .ipynb file attached at the end of the pdf for this qn.

e) No, different batch sizes show different convergence rates with the same learning

rate. From the fig. convergence speed reduces as batch size increases.

Pleased find the curves with the tuned learning rates at the end of the .ipynb notebook.

Since the new curves are tuned better they give a better precision o/p than the earlier ones.

A higher learning rate worked better for smaller batches sizes because increasing λ reduces the impact of regularisation, which can become overpowering when you reduce your batch size.

A high learning rate of 0.3 yielded the overall fastest convergence in terms of wall clock time

imph6h73h

February 9, 2024

[151]:
```
# load data
from sklearn import datasets
boston = datasets.load_boston()
print(boston.keys())
```

```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename',
'data_module'])
```

```
/Users/prateek/anaconda3/envs/AnacondaTest/lib/python3.11/site-
packages/sklearn/utils/deprecation.py:87: FutureWarning: Function load_boston is
deprecated; `load_boston` is deprecated in 1.0 and will be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np

        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::
```

```
        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.
  warnings.warn(msg, category=FutureWarning)
```

[152]:
```
# a form of summary of the data
feature = boston.data
price = boston.target
print('data size = ', feature.shape)
print('target size = ', price.shape)
print('feature attributes: ', boston.feature_names)
print(boston.DESCR)
```

```
data size =  (506, 13)
target size =  (506,)
feature attributes:  ['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD'
'TAX' 'PTRATIO'
 'B' 'LSTAT']
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value
(attribute 14) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000
sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0
otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of black people
by town
```

```
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/


This dataset was taken from the StatLib library which is maintained at Carnegie
Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
…', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that
address regression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential
Data and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In
Proceedings on the Tenth International Conference of Machine Learning, 236-243,
University of Massachusetts, Amherst. Morgan Kaufmann.
```

[153]:
```
# more details of the data
import pandas as pd
df_feature = pd.DataFrame(feature, columns = boston.feature_names)
df_target = pd.DataFrame(price, columns =['MEDV'])
df_boston = pd.concat([df_feature, df_target,], axis = 1)
```

[154]:
```
df_boston.head()
```

[154]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	\
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	

```
   PTRATIO       B  LSTAT  MEDV
0     15.3  396.90   4.98  24.0
1     17.8  396.90   9.14  21.6
2     17.8  392.83   4.03  34.7
3     18.7  394.63   2.94  33.4
4     18.7  396.90   5.33  36.2
```

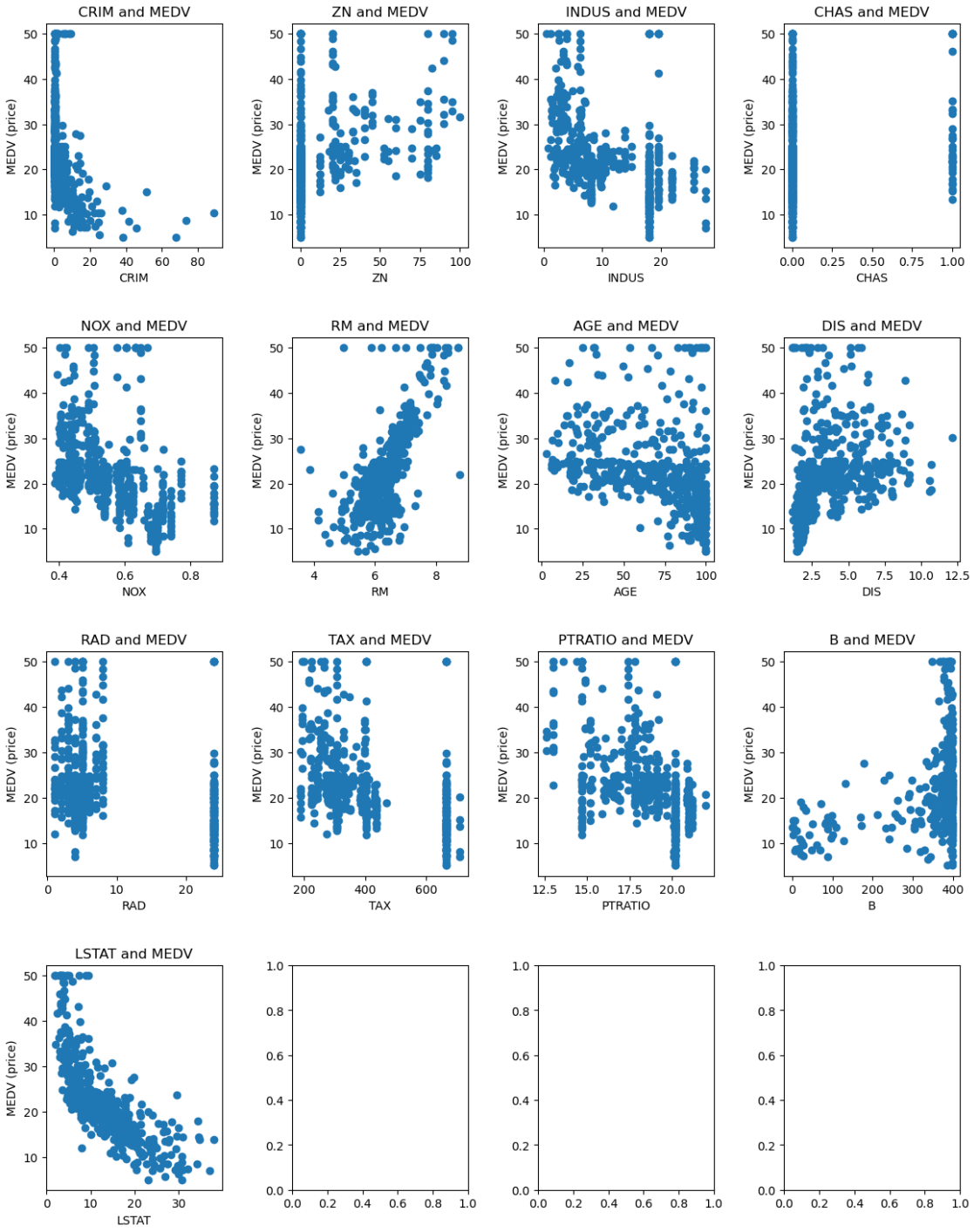
[155]:
```
df_boston.describe()
```

[155]:
```
              CRIM          ZN       INDUS        CHAS         NOX          RM  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean     3.613524   11.363636   11.136779    0.069170    0.554695    6.284634
std      8.601545   23.322453    6.860353    0.253994    0.115878    0.702617
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000
25%      0.082045    0.000000    5.190000    0.000000    0.449000    5.885500
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.208500
75%      3.677083   12.500000   18.100000    0.000000    0.624000    6.623500
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000

              AGE         DIS         RAD         TAX     PTRATIO           B  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean    68.574901    3.795043    9.549407  408.237154   18.455534  356.674032
std     28.148861    2.105710    8.707259  168.537116    2.164946   91.294864
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320000
25%     45.025000    2.100175    4.000000  279.000000   17.400000  375.377500
50%     77.500000    3.207450    5.000000  330.000000   19.050000  391.440000
75%     94.075000    5.188425   24.000000  666.000000   20.200000  396.225000
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.900000

            LSTAT        MEDV
count  506.000000  506.000000
mean    12.653063   22.532806
std      7.141062    9.197104
min      1.730000    5.000000
25%      6.950000   17.025000
50%     11.360000   21.200000
75%     16.955000   25.000000
max     37.970000   50.000000
```

[156]:
```
# 2.1 how does each feature relate to the price
import matplotlib.pyplot as plt
plt.figure()
fig,axes = plt.subplots(4, 4, figsize=(14,18))
fig.subplots_adjust(wspace=.4, hspace=.4)
img_index = 0
for i in range(boston.feature_names.size):
```

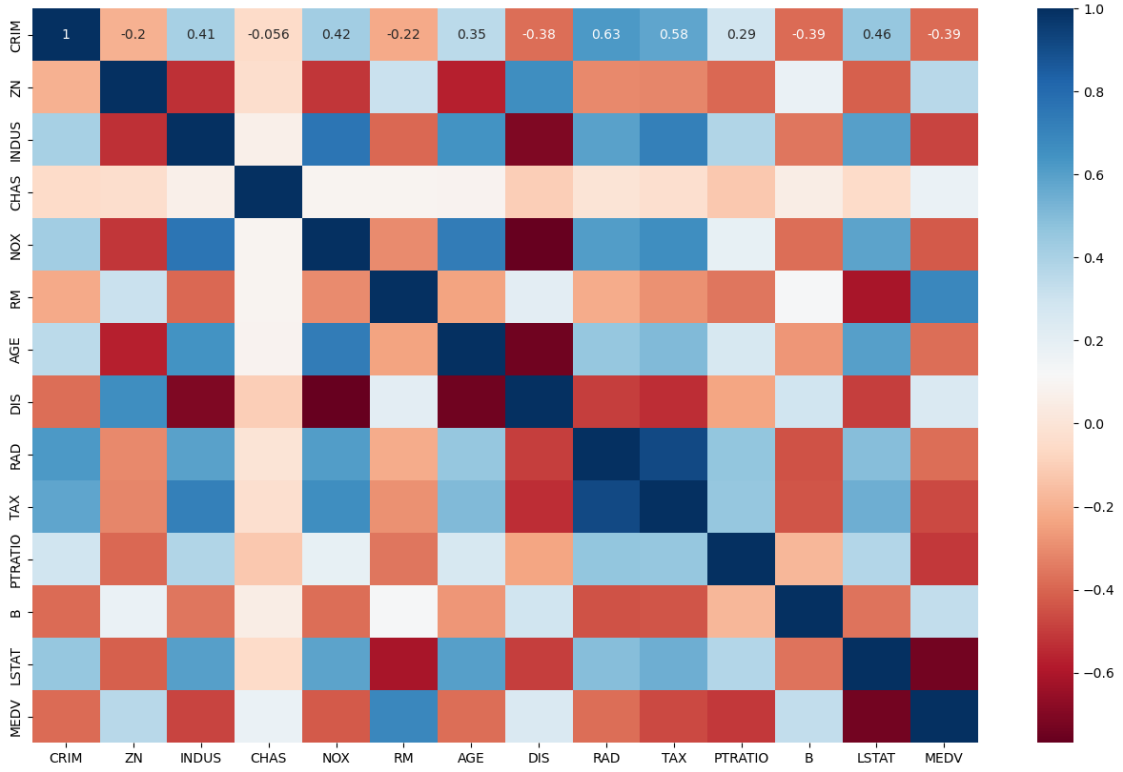
```
    row, col = i // 4, i % 4
    axes[row][col].scatter(feature[:,i], price)
    axes[row][col].set_title(boston.feature_names[i] + ' and MEDV')
    axes[row][col].set_xlabel(boston.feature_names[i])
    axes[row][col].set_ylabel('MEDV (price)')
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```



[157]:
```
# 2.2 correlation matrix
import seaborn as sns
fig, ax = plt.subplots(figsize=(16, 10))
correlation = df_boston.corr()
sns.heatmap(correlation, annot = True, cmap = 'RdBu')
```

```
plt.show()
correlation
```



[157]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE
CRIM	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734
ZN	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537
INDUS	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779
CHAS	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518
NOX	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470
RM	-0.219247	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.240265
AGE	0.352734	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.000000
DIS	-0.379670	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.747881
RAD	0.625505	-0.311948	0.595129	-0.007368	0.611441	-0.209847	0.456022
TAX	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456
PTRATIO	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515
B	-0.385064	0.175520	-0.356977	0.048788	-0.380051	0.128069	-0.273534
LSTAT	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339
MEDV	-0.388305	0.360445	-0.483725	0.175260	-0.427321	0.695360	-0.376955

	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
CRIM	-0.379670	0.625505	0.582764	0.289946	-0.385064	0.455621	-0.388305
ZN	0.664408	-0.311948	-0.314563	-0.391679	0.175520	-0.412995	0.360445

```
INDUS   -0.708027  0.595129  0.720760  0.383248 -0.356977  0.603800 -0.483725
CHAS    -0.099176 -0.007368 -0.035587 -0.121515  0.048788 -0.053929  0.175260
NOX     -0.769230  0.611441  0.668023  0.188933 -0.380051  0.590879 -0.427321
RM       0.205246 -0.209847 -0.292048 -0.355501  0.128069 -0.613808  0.695360
AGE     -0.747881  0.456022  0.506456  0.261515 -0.273534  0.602339 -0.376955
DIS      1.000000 -0.494588 -0.534432 -0.232471  0.291512 -0.496996  0.249929
RAD     -0.494588  1.000000  0.910228  0.464741 -0.444413  0.488676 -0.381626
TAX     -0.534432  0.910228  1.000000  0.460853 -0.441808  0.543993 -0.468536
PTRATIO -0.232471  0.464741  0.460853  1.000000 -0.177383  0.374044 -0.507787
B        0.291512 -0.444413 -0.441808 -0.177383  1.000000 -0.366087  0.333461
LSTAT   -0.496996  0.488676  0.543993  0.374044 -0.366087  1.000000 -0.737663
MEDV     0.249929 -0.381626 -0.468536 -0.507787  0.333461 -0.737663  1.000000
```

[158]:
```
# train-test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(feature, price, test_size=0.
 ↪3, random_state=8)
```

[159]:
```
# 2.3 linear regression and ridge regression
import numpy as np

def least_square(X, y):
    #TODO
    theta = np.matmul(X.transpose(),X)
    theta = np.linalg.inv(theta)
    theta = np.matmul(theta,X.transpose())
    theta = np.matmul(theta, y)
    return theta

def ridge_reg(X, y, eta):
    #TODO
    theta = np.matmul(X.transpose(),X)
    num_rows, num_columns = theta.shape
    theta = theta + (eta/2)*np.identity(num_rows)
    theta = np.linalg.inv(theta)
    theta = np.matmul(theta,X.transpose())
    theta = np.matmul(theta, y)
    return theta

# apply linear regression
theta = least_square(X_train, y_train)
df_theta = pd.DataFrame(zip(boston.feature_names,␣
 ↪theta),columns=['Feature','Coeff'])
print("Linear Regression Output")
display(df_theta)

df_theta_r_temp = df_theta
```

```
# apply ridge regression
for i in range(25):
    theta_r = ridge_reg(X_train, y_train, i)
    df_theta_r = pd.DataFrame(zip(boston.feature_names,␣
 ↪theta_r),columns=['Feature','Coeff'])
    print("Ridge Regression Output with eta = ", i)
    if(i == 20):
        df_theta_r_temp = df_theta_r
    display(df_theta_r)

# Take eta as 20, for the right balance of training vs testing dataset accuracy
df_theta_r = df_theta_r_temp
```

```
Linear Regression Output
```

	Feature	Coeff
0	CRIM	-0.099324
1	ZN	0.052251
2	INDUS	0.004516
3	CHAS	2.957261
4	NOX	1.127938
5	RM	5.854198
6	AGE	-0.014957
7	DIS	-0.920844
8	RAD	0.159519
9	TAX	-0.008934
10	PTRATIO	-0.435674
11	B	0.014905
12	LSTAT	-0.474751

```
Ridge Regression Output with eta =  0
```

	Feature	Coeff
0	CRIM	-0.099324
1	ZN	0.052251
2	INDUS	0.004516
3	CHAS	2.957261
4	NOX	1.127938
5	RM	5.854198
6	AGE	-0.014957
7	DIS	-0.920844
8	RAD	0.159519
9	TAX	-0.008934
10	PTRATIO	-0.435674
11	B	0.014905
12	LSTAT	-0.474751

```
Ridge Regression Output with eta =  1
```

	Feature	Coeff

```
0      CRIM -0.099514
1        ZN  0.052396
2     INDUS  0.005971
3      CHAS  2.901961
4       NOX  0.928628
5        RM  5.852968
6       AGE -0.014481
7       DIS -0.920441
8       RAD  0.159970
9       TAX -0.008921
10  PTRATIO -0.433651
11        B  0.014964
12    LSTAT -0.474664
```

```
Ridge Regression Output with eta =  2
```

```
    Feature     Coeff
0      CRIM -0.099660
1        ZN  0.052550
2     INDUS  0.006999
3      CHAS  2.846734
4       NOX  0.810144
5        RM  5.848060
6       AGE -0.014077
7       DIS -0.919409
8       RAD  0.160346
9       TAX -0.008919
10  PTRATIO -0.431701
11        B  0.015015
12    LSTAT -0.474969
```

```
Ridge Regression Output with eta =  3
```

```
    Feature     Coeff
0      CRIM -0.099782
1        ZN  0.052709
2     INDUS  0.007809
3      CHAS  2.792744
4       NOX  0.731256
5        RM  5.841327
6       AGE -0.013709
7       DIS -0.918069
8       RAD  0.160677
9       TAX -0.008922
10  PTRATIO -0.429775
11        B  0.015062
12    LSTAT -0.475464
```

```
Ridge Regression Output with eta =  4
```

```
    Feature     Coeff
```

```
0      CRIM -0.099889
1        ZN  0.052870
2     INDUS  0.008487
3      CHAS  2.740394
4       NOX  0.674718
5        RM  5.833546
6       AGE -0.013363
7       DIS -0.916557
8       RAD  0.160976
9       TAX -0.008926
10  PTRATIO -0.427853
11        B  0.015106
12    LSTAT -0.476063
```

```
Ridge Regression Output with eta =  5
```

```
    Feature     Coeff
0      CRIM -0.099986
1        ZN  0.053032
2     INDUS  0.009079
3      CHAS  2.689813
4       NOX  0.632042
5        RM  5.825104
6       AGE -0.013033
7       DIS -0.914939
8       RAD  0.161248
9       TAX -0.008932
10  PTRATIO -0.425927
11        B  0.015148
12    LSTAT -0.476724
```

```
Ridge Regression Output with eta =  6
```

```
    Feature     Coeff
0      CRIM -0.100074
1        ZN  0.053194
2     INDUS  0.009608
3      CHAS  2.641017
4       NOX  0.598560
5        RM  5.816216
6       AGE -0.012713
7       DIS -0.913252
8       RAD  0.161499
9       TAX -0.008938
10  PTRATIO -0.423992
11        B  0.015189
12    LSTAT -0.477424
```

```
Ridge Regression Output with eta =  7
```

```
    Feature     Coeff
```

```
0      CRIM -0.100155
1        ZN  0.053356
2     INDUS  0.010089
3      CHAS  2.593974
4       NOX  0.571497
5        RM  5.807010
6       AGE -0.012402
7       DIS -0.911518
8       RAD  0.161731
9       TAX -0.008945
10  PTRATIO -0.422048
11        B  0.015228
12    LSTAT -0.478151
```

```
Ridge Regression Output with eta =  8
```

```
    Feature     Coeff
0      CRIM -0.100231
1        ZN  0.053517
2     INDUS  0.010532
3      CHAS  2.548625
4       NOX  0.549095
5        RM  5.797569
6       AGE -0.012097
7       DIS -0.909751
8       RAD  0.161946
9       TAX -0.008951
10  PTRATIO -0.420093
11        B  0.015267
12    LSTAT -0.478894
```

```
Ridge Regression Output with eta =  9
```

```
    Feature     Coeff
0      CRIM -0.100302
1        ZN  0.053679
2     INDUS  0.010943
3      CHAS  2.504903
4       NOX  0.530189
5        RM  5.787950
6       AGE -0.011799
7       DIS -0.907961
8       RAD  0.162146
9       TAX -0.008956
10  PTRATIO -0.418128
11        B  0.015304
12    LSTAT -0.479650
```

```
Ridge Regression Output with eta =  10
```

```
    Feature     Coeff
```

```
0      CRIM -0.100368
1        ZN  0.053839
2     INDUS  0.011329
3      CHAS  2.462736
4       NOX  0.513974
5        RM  5.778194
6       AGE -0.011505
7       DIS -0.906153
8       RAD  0.162332
9       TAX -0.008962
10  PTRATIO -0.416153
11        B  0.015341
12    LSTAT -0.480413
```

```
Ridge Regression Output with eta =  11
```

```
    Feature     Coeff
0      CRIM -0.100431
1        ZN  0.053999
2     INDUS  0.011691
3      CHAS  2.422052
4       NOX  0.499876
5        RM  5.768328
6       AGE -0.011216
7       DIS -0.904333
8       RAD  0.162505
9       TAX -0.008967
10  PTRATIO -0.414168
11        B  0.015378
12    LSTAT -0.481181
```

```
Ridge Regression Output with eta =  12
```

```
    Feature     Coeff
0      CRIM -0.100490
1        ZN  0.054158
2     INDUS  0.012033
3      CHAS  2.382781
4       NOX  0.487477
5        RM  5.758376
6       AGE -0.010930
7       DIS -0.902504
8       RAD  0.162666
9       TAX -0.008971
10  PTRATIO -0.412174
11        B  0.015414
12    LSTAT -0.481953
```

```
Ridge Regression Output with eta =  13
```

```
    Feature     Coeff
```

```
0      CRIM -0.100545
1        ZN  0.054317
2     INDUS  0.012357
3      CHAS  2.344856
4       NOX  0.476462
5        RM  5.748354
6       AGE -0.010648
7       DIS -0.900668
8       RAD  0.162816
9       TAX -0.008975
10  PTRATIO -0.410173
11        B  0.015449
12    LSTAT -0.482726
```

```
Ridge Regression Output with eta =  14
```

```
    Feature     Coeff
0      CRIM -0.100598
1        ZN  0.054475
2     INDUS  0.012665
3      CHAS  2.308210
4       NOX  0.466590
5        RM  5.738275
6       AGE -0.010370
7       DIS -0.898828
8       RAD  0.162955
9       TAX -0.008979
10  PTRATIO -0.408164
11        B  0.015484
12    LSTAT -0.483500
```

```
Ridge Regression Output with eta =  15
```

```
    Feature     Coeff
0      CRIM -0.100648
1        ZN  0.054632
2     INDUS  0.012958
3      CHAS  2.272783
4       NOX  0.457674
5        RM  5.728152
6       AGE -0.010094
7       DIS -0.896985
8       RAD  0.163084
9       TAX -0.008982
10  PTRATIO -0.406149
11        B  0.015518
12    LSTAT -0.484274
```

```
Ridge Regression Output with eta =  16
```

```
    Feature     Coeff
```

```
0      CRIM -0.100695
1        ZN  0.054789
2     INDUS  0.013238
3      CHAS  2.238517
4       NOX  0.449568
5        RM  5.717993
6       AGE -0.009822
7       DIS -0.895141
8       RAD  0.163205
9       TAX -0.008985
10  PTRATIO -0.404128
11        B  0.015552
12    LSTAT -0.485046
```

```
Ridge Regression Output with eta =  17
```

```
    Feature     Coeff
0      CRIM -0.100740
1        ZN  0.054945
2     INDUS  0.013505
3      CHAS  2.205356
4       NOX  0.442153
5        RM  5.707805
6       AGE -0.009552
7       DIS -0.893296
8       RAD  0.163316
9       TAX -0.008988
10  PTRATIO -0.402102
11        B  0.015585
12    LSTAT -0.485818
```

```
Ridge Regression Output with eta =  18
```

```
    Feature     Coeff
0      CRIM -0.100782
1        ZN  0.055100
2     INDUS  0.013761
3      CHAS  2.173249
4       NOX  0.435333
5        RM  5.697594
6       AGE -0.009285
7       DIS -0.891451
8       RAD  0.163420
9       TAX -0.008990
10  PTRATIO -0.400072
11        B  0.015618
12    LSTAT -0.486588
```

```
Ridge Regression Output with eta =  19
```

```
    Feature     Coeff
```

```
0      CRIM -0.100823
1        ZN  0.055254
2     INDUS  0.014006
3      CHAS  2.142145
4       NOX  0.429030
5        RM  5.687367
6       AGE -0.009020
7       DIS -0.889607
8       RAD  0.163516
9       TAX -0.008991
10  PTRATIO -0.398037
11        B  0.015651
12    LSTAT -0.487356
```

```
Ridge Regression Output with eta =  20
```

```
    Feature     Coeff
0      CRIM -0.100861
1        ZN  0.055408
2     INDUS  0.014242
3      CHAS  2.112001
4       NOX  0.423179
5        RM  5.677128
6       AGE -0.008757
7       DIS -0.887765
8       RAD  0.163604
9       TAX -0.008993
10  PTRATIO -0.396000
11        B  0.015683
12    LSTAT -0.488121
```

```
Ridge Regression Output with eta =  21
```

```
    Feature     Coeff
0      CRIM -0.100898
1        ZN  0.055561
2     INDUS  0.014468
3      CHAS  2.082771
4       NOX  0.417726
5        RM  5.666881
6       AGE -0.008497
7       DIS -0.885925
8       RAD  0.163685
9       TAX -0.008994
10  PTRATIO -0.393960
11        B  0.015715
12    LSTAT -0.488884
```

```
Ridge Regression Output with eta =  22
```

```
    Feature     Coeff
```

```
0      CRIM -0.100933
1        ZN  0.055713
2     INDUS  0.014685
3      CHAS  2.054414
4       NOX  0.412625
5        RM  5.656629
6       AGE -0.008239
7       DIS -0.884088
8       RAD  0.163760
9       TAX -0.008994
10  PTRATIO -0.391918
11        B  0.015747
12    LSTAT -0.489645
```

Ridge Regression Output with eta = 23

```
    Feature     Coeff
0      CRIM -0.100966
1        ZN  0.055865
2     INDUS  0.014894
3      CHAS  2.026893
4       NOX  0.407837
5        RM  5.646376
6       AGE -0.007983
7       DIS -0.882254
8       RAD  0.163829
9       TAX -0.008994
10  PTRATIO -0.389874
11        B  0.015778
12    LSTAT -0.490402
```

Ridge Regression Output with eta = 24

```
    Feature     Coeff
0      CRIM -0.100997
1        ZN  0.056015
2     INDUS  0.015095
3      CHAS  2.000170
4       NOX  0.403330
5        RM  5.636124
6       AGE -0.007729
7       DIS -0.880422
8       RAD  0.163892
9       TAX -0.008994
10  PTRATIO -0.387829
11        B  0.015809
12    LSTAT -0.491157
```

[160]:
```
# 2.4 evaluation
def pred_fn(X, theta):
    #TODO
    pred = np.matmul(X, theta)
    return pred

def root_mean_square_error(pred, y):
    #TODO
    diff_matrix = y - pred
    rmse = diff_matrix**2
    rmse = rmse.sum()
    rmse = rmse/np.size(y)
    rmse = np.sqrt(rmse)
    return rmse

pred_linear_train = pred_fn(X_train,df_theta.loc[:,"Coeff"])
pred_linear_test = pred_fn(X_test,df_theta.loc[:,"Coeff"])
rmse_linear_train = root_mean_square_error(pred_linear_train, y_train)
rmse_linear_test = root_mean_square_error(pred_linear_test, y_test)
print("Training Set RMSE of Linear Regression : ", rmse_linear_train)
print("Testing Set RMSE of Linear Regression : ", rmse_linear_test)

pred_ridge_train = pred_fn(X_train,df_theta_r.loc[:,"Coeff"])
pred_ridge_test = pred_fn(X_test,df_theta_r.loc[:,"Coeff"])
rmse_ridge_train = root_mean_square_error(pred_ridge_train, y_train)
rmse_ridge_test = root_mean_square_error(pred_ridge_test, y_test)
print("Training Set RMSE of Ridge Regression : ", rmse_ridge_train)
print("Testing Set RMSE of Ridge Regression : ", rmse_ridge_test)
```

```
Training Set RMSE of Linear Regression :  4.820626531838223
Testing Set RMSE of Linear Regression :  5.209217510530916
Training Set RMSE of Ridge Regression :  4.829777333975097
Testing Set RMSE of Ridge Regression :  5.189347305423606
```

[161]:
```
# 2.5 linear models of top-3 features
X_train_top3 = np.stack((X_train[:,10],X_train[:,5],X_train[:,12])).transpose()
X_test_top3 = np.stack((X_test[:,10],X_test[:,5],X_test[:,12])).transpose()
# linear regression using top-3 features
theta_top3 = least_square(X_train_top3, y_train)
df_theta_top3 = pd.DataFrame(zip(boston.feature_names,␣
 ↪theta_top3),columns=['Feature','Coeff'])
pred_linear_train_top3 = pred_fn(X_train_top3,df_theta_top3.loc[:,"Coeff"])
pred_linear_test_top3 = pred_fn(X_test_top3,df_theta_top3.loc[:,"Coeff"])
rmse_linear_train_top3 = root_mean_square_error(pred_linear_train_top3, y_train)
rmse_linear_test_top3 = root_mean_square_error(pred_linear_test_top3, y_test)
print("Training Set RMSE of Linear Regression for top 3 features : ",␣
 ↪rmse_linear_train_top3)
```

```
print("Testing Set RMSE of Linear Regression for top 3 features : ",␣
 ↪rmse_linear_test_top3)
# ridge regression using top-3 features
theta_r_top3 = ridge_reg(X_train_top3, y_train, 20.0)
df_theta_r_top3 = pd.DataFrame(zip(boston.feature_names,␣
 ↪theta_r_top3),columns=['Feature','Coeff'])
pred_ridge_train_top3 = pred_fn(X_train_top3,df_theta_r_top3.loc[:,"Coeff"])
pred_ridge_test_top3 = pred_fn(X_test_top3,df_theta_r_top3.loc[:,"Coeff"])
rmse_ridge_train_top3 = root_mean_square_error(pred_ridge_train_top3, y_train)
rmse_ridge_test_top3 = root_mean_square_error(pred_ridge_test_top3, y_test)
print("Training Set RMSE of Ridge Regression for top 3 features and eta = 20.0 :
 ↪ ", rmse_ridge_train_top3)
print("Testing Set RMSE of Ridge Regression for top 3 features and eta = 20.0 :␣
 ↪", rmse_ridge_test_top3)
```

```
Training Set RMSE of Linear Regression for top 3 features :  5.273361751695365
Testing Set RMSE of Linear Regression for top 3 features :  5.494723646664577
Training Set RMSE of Ridge Regression for top 3 features and eta = 20.0 :
5.276310228536866
Testing Set RMSE of Ridge Regression for top 3 features and eta = 20.0 :
5.477573443118745
```

[ ]:

qrnmj2fvp

February 9, 2024

[489]:
```
from sklearn import datasets
digits = datasets.load_digits()
print(digits.keys())
```

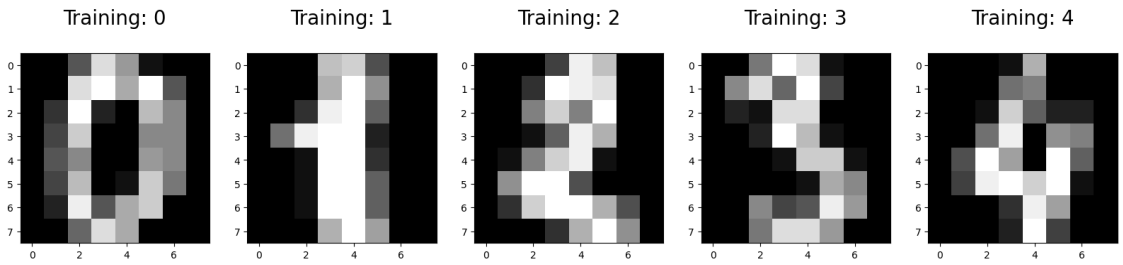
```
dict_keys(['data', 'target', 'frame', 'feature_names', 'target_names', 'images',
'DESCR'])
```

[490]:
```
# a form of summary of the data
print('data size = ', digits.data.shape)
print('target size = ', digits.target.shape)
print(digits.DESCR)
```

```
data size =  (1797, 64)
target size =  (1797,)
.. _digits_dataset:

Optical recognition of handwritten digits dataset
--------------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 1797
    :Number of Attributes: 64
    :Attribute Information: 8x8 image of integer pixels in the range 0..16.
    :Missing Attribute Values: None
    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)
    :Date: July; 1998

This is a copy of the test set of the UCI ML hand-written digits datasets
https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digit
s

The data set contains images of hand-written digits: 10 classes where
each class refers to a digit.

Preprocessing programs made available by NIST were used to extract
normalized bitmaps of handwritten digits from a preprinted form. From a
total of 43 people, 30 contributed to the training set and different 13
```

```
to the test set. 32x32 bitmaps are divided into nonoverlapping blocks of
4x4 and the number of on pixels are counted in each block. This generates
an input matrix of 8x8 where each element is an integer in the range
0..16. This reduces dimensionality and gives invariance to small
distortions.

For info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G.
T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C.
L. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469,
1994.

.. topic:: References

  - C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their
    Applications to Handwritten Digit Recognition, MSc Thesis, Institute of
    Graduate Studies in Science and Engineering, Bogazici University.
  - E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.
  - Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin.
    Linear dimensionalityreduction using relevance weighted LDA. School of
    Electrical and Electronic Engineering Nanyang Technological University.
    2005.
  - Claudio Gentile. A New Approximate Maximal Margin Classification
    Algorithm. NIPS. 2000.
```

[491]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

# show examples of dataset
plt.figure(figsize=(20,4))
for index, (image, label) in enumerate(zip(digits.data[0:5], digits.target[0:
 ↪5])):
    plt.subplot(1, 5, index + 1)
    plt.imshow(np.reshape(image, (8,8)), cmap=plt.cm.gray)
    plt.title('Training: %i\n' % label, fontsize = 20)
```



[492]:
```
# train-test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target,␣
 ↪test_size=0.25, random_state=8)
print(X_train[256], y_train[256])
```

```
[ 0.  0.  2. 16. 10.  0.  0.  0.  0.  0.  4. 16. 16.  5.  0.  0.  0.  0.
  8. 16. 16.  3.  0.  0.  0.  0.  9. 16. 16.  3.  0.  0.  0.  0.  8. 16.
 16.  3.  0.  0.  0.  0.  8. 16. 16.  1.  0.  0.  0.  0.  5. 16. 14.  0.
  0.  0.  0.  0.  1. 12. 16.  3.  0.  0.] 1
```

[493]:
```
# 3.2 batch gradient descent (GD) for Logistic regression
def LogisticRegression_GD(X_train, y_train, learning_rate):
    loss = []
    dataset_size, data_attributes = X_train.shape
    print("Dataset Size : ", dataset_size)
    epsilon = 1.0*np.exp(-4)
    eta = 0.1
    c = 10
    W = np.random.normal(0, 1, (data_attributes*c))
    b = np.zeros(c)
    W.resize(data_attributes, c)
    while 1 :
        Z = np.matmul(X_train, W)
        for i in range(dataset_size):
            Z[i] = Z[i] + b
        k=0
        for i in range(dataset_size):
            for j in range(c):
                if(Z[i][j] > k) :
                    k = Z[i][j]
        Z = Z - k*np.ones(Z.shape)
        Z = np.exp(Z)
        Z_sum = np.sum(Z, axis = 1)
        for i in range(dataset_size):
            for j in range(c):
                Z[i][j] = Z[i][j]/Z_sum[i]
        Zlog = -np.log(Z)
        F = Zlog.sum()
        F = F/dataset_size
        W_2 = W**2
        W_2_reg = eta*W_2.sum()/2
        F = F + W_2_reg
        loss.append(F)
        if len(loss) >= 2 :
            if np.abs(loss[-1] - loss[-2]) < epsilon :
                break
```

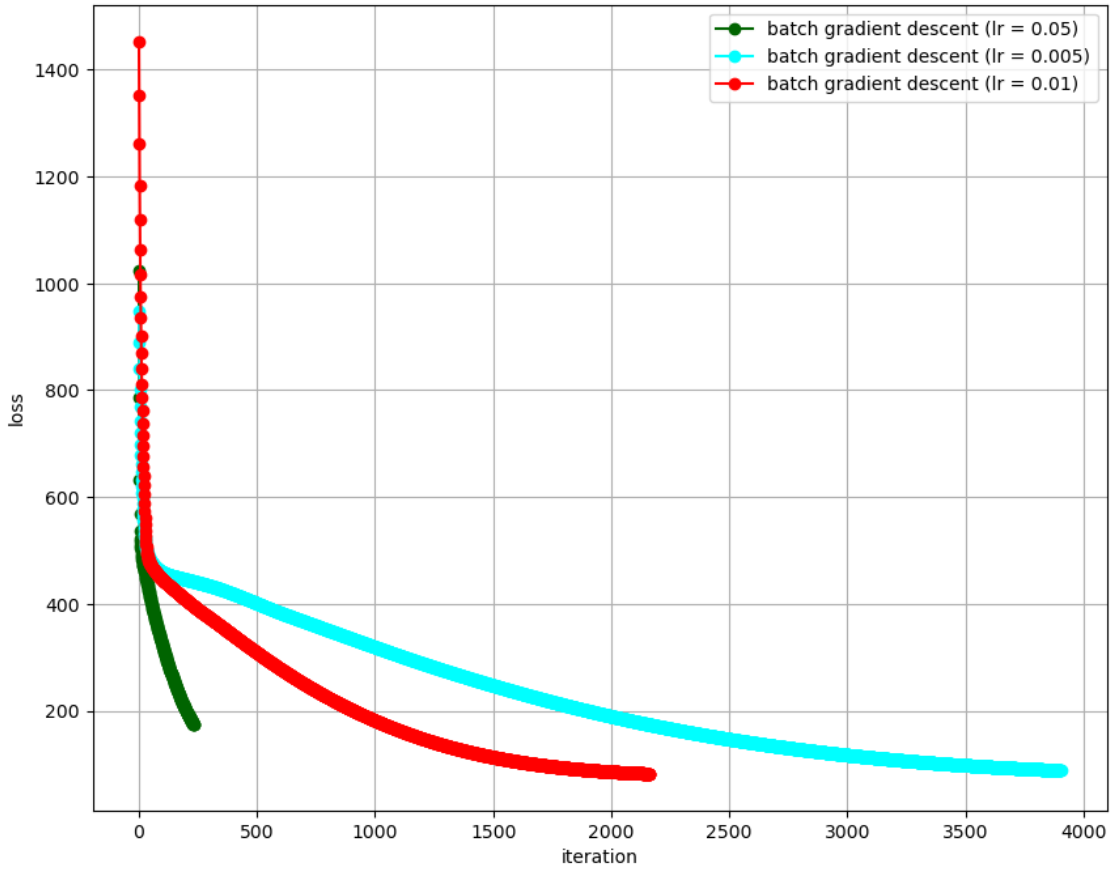
```
        #Gradient Descent Stuff
        gradient_desc_sum = 0;
        for i in range(dataset_size):
            error = np.zeros(c)
            for j in range(c):
                if(y_train[i] == j+1):
                    error[j] = (1 - Z[i][j])
                else :
                    error[j] = (-Z[i][j])
            temp = np.zeros((data_attributes, c))
            for j in range(data_attributes):
                for k in range(c):
                    temp[j][k] = X_train[i][j]*error[k]
            W = W + (learning_rate*temp/dataset_size)
        W = W - (learning_rate*eta*W)
    return W, b, np.array(loss)
```

[495]:
```
def calculate_precision(Y, W, X):
    Z = np.matmul(X, W)
    Z = np.exp(Z)
    c = 10
    dataset_size, data_attributes = X.shape
    Z_sum = np.sum(Z, axis = 1)
    for i in range(dataset_size):
            for j in range(c):
                Z[i][j] = Z[i][j]/Z_sum[i]
    argmax = np.argmax(Z, axis = 1)

    correct_predictions = 0
    for i in range(dataset_size):
        if Y[i] == (argmax[i] + 1):
            correct_predictions += 1
    precision = (correct_predictions/dataset_size)*100
    return precision

# evaluation of different learning rate
learning_rate = [5.0e-2, 5.0e-3, 1.0e-2]
cl = ['darkgreen', 'cyan', 'red']
fig, ax = plt.subplots(figsize=(10, 8))
for i in range(len(learning_rate)):

    print('---------------------------------------')
    print('learning rate =', learning_rate[i])

    W, b, loss_GD = LogisticRegression_GD(X_train, y_train, learning_rate[i])

    print("Final Loss Value : ", loss_GD[-1])
```

```
    prec_train = calculate_precision(y_train, W, X_train)
    print('training precision =', prec_train)

    prec_test = calculate_precision(y_test, W, X_test)
    print('test precision =', prec_test)

    plt.plot(loss_GD, c = cl[i], ls = '-', marker = 'o', label = 'batch␣
 ↪gradient descent (lr = ' + str(learning_rate[i]) + ')')

plt.grid()
plt.legend()
plt.xlabel('iteration')
plt.ylabel('loss')
```

```
---------------------------------------
learning rate = 0.05
Dataset Size :  1347
Final Loss Value :  174.10307939002357
training precision = 80.69784706755753
test precision = 78.88888888888889
---------------------------------------
learning rate = 0.005
Dataset Size :  1347
Final Loss Value :  88.26890279017098
training precision = 88.2702301410542
test precision = 86.22222222222223
---------------------------------------
learning rate = 0.01
Dataset Size :  1347
Final Loss Value :  81.93692451199382
training precision = 88.71566443949519
test precision = 86.66666666666667
```

[495]: Text(0, 0.5, 'loss')



[509]:
```
# 3.3 stochastic gradient descent (SGD) for Logistic regression

def LogisticRegression_SGD(X, y, batch_size, lr=1.0e-2, eta=2.0e-1, eps = 1.
↪0e-4, max_epoch=500):

    loss = []
    dataset_size, data_attributes = X_train.shape
    print("Dataset Size : ", dataset_size)
    epsilon = 1.0*np.exp(-4)
    eta = 0.1
    c = 10
    notstop = True
    W = np.random.normal(0, 1, (data_attributes*c))
    b = np.zeros(c)
    W.resize(data_attributes, c)
    batch_starting_point = 0
    epoch = 0
    # optimization loop
    while notstop and epoch < max_epoch:
```

```
        batch_ending_point = 0
        if batch_starting_point + batch_size <= dataset_size:
            batch_starting_point = 0

        Z = np.matmul(X[batch_starting_point:batch_starting_point+100], W)
        batch_starting_point = batch_starting_point+100
        for i in range(batch_size):
            Z[i] = Z[i] + b
        k=0
        for i in range(batch_size):
            for j in range(c):
                if(Z[i][j] > k) :
                    k = Z[i][j]
        Z = Z - k*np.ones(Z.shape)
        Z = np.exp(Z)
        Z_sum = np.sum(Z, axis = 1)
        for i in range(batch_size):
            for j in range(c):
                Z[i][j] = Z[i][j]/Z_sum[i]
        Zlog = -np.log(Z)
        F = Zlog.sum()
        F = F/batch_size
        W_2 = W**2
        W_2_reg = eta*W_2.sum()/2
        F = F + W_2_reg
        loss.append(F)
        if len(loss) >= 2 :
            if np.abs(loss[-1] - loss[-2]) < epsilon :
                break

        # half lr if not improving in 10 epochs
        if epoch > 10:
            if loss[epoch - 10] <= loss[epoch] - eps:
                lr *= 0.5
                print('reduce learning rate to', lr)

        # stop if not improving in 20 epochs
        if epoch > 20:
            if loss[epoch - 20] <= loss[epoch] - eps or abs(loss[epoch] -␣
↪loss[epoch-1]) <= eps:
                notstop = False
                break

        #Gradient Descent Stuff
        gradient_desc_sum = 0;
        for i in range(batch_size):
            error = np.zeros(c)
```

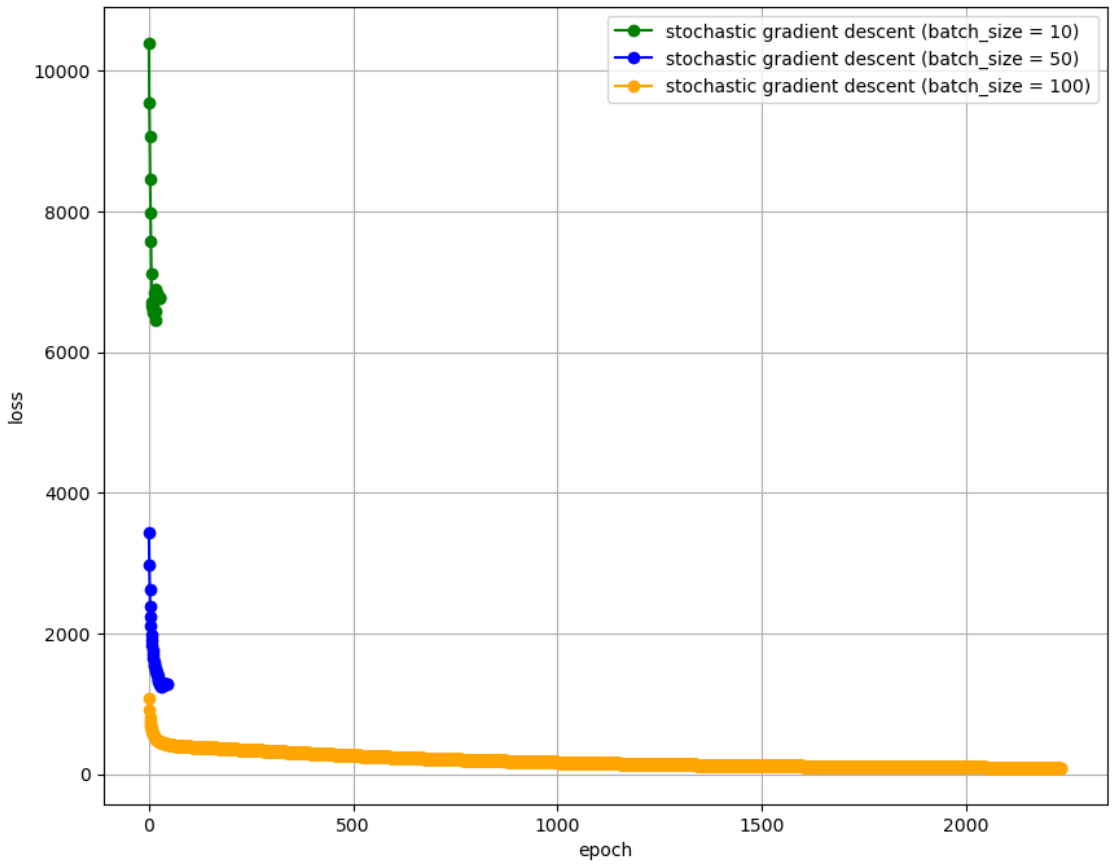
```
            for j in range(c):
                if(y_train[i] == j+1):
                    error[j] = (1 - Z[i][j])
                else :
                    error[j] = (-Z[i][j])
            temp = np.zeros((data_attributes, c))
            for j in range(data_attributes):
                for k in range(c):
                    temp[j][k] = X_train[i][j]*error[k]
            W = W + (lr*temp/batch_size)
        W = W - (lr*eta*W)
        epoch += 1

    return (W, b, np.array(loss))
```

[527]:
```
# evaluation of different batch size
bs = [10, 50, 100]
cl = ['green', 'blue', 'orange']
# Standard Learning Rate
lr = [1.0e-2, 1.0e-2, 1.0e-2]
fig, ax = plt.subplots(figsize=(10, 8))

for i in range(len(bs)):

    print('---------------------------------------')
    print('batch_size =', bs[i])
    W, b, loss_SGD = LogisticRegression_SGD(X_train, y_train, bs[i], lr[i], eta␣
 ↪= 2.0e-1, eps = 1.0e-4, max_epoch = 2500)

    print("Final Loss Value : ", loss_SGD[-1])
    prec_train = calculate_precision(y_train, W, X_train)
    print('training precision =', prec_train)

    prec_test = calculate_precision(y_test, W, X_test)
    print('test precision =', prec_test)

    plt.plot(loss_SGD, c = cl[i], ls = '-', marker = 'o', label = 'stochastic␣
 ↪gradient descent (batch_size = ' + str(bs[i]) + ')')

plt.grid()
plt.legend()
plt.xlabel('epoch')
plt.ylabel('loss')
```

```
---------------------------------------
batch_size = 10
Dataset Size :  1347
```

```
reduce learning rate to 0.005
reduce learning rate to 0.0025
reduce learning rate to 0.00125
reduce learning rate to 0.000625
reduce learning rate to 0.0003125
reduce learning rate to 0.00015625
reduce learning rate to 7.8125e-05
reduce learning rate to 3.90625e-05
Final Loss Value :  6775.26035476595
training precision = 13.21455085374907
test precision = 15.777777777777777
--------------------------------------
batch_size = 50
Dataset Size :  1347
reduce learning rate to 0.005
reduce learning rate to 0.0025
reduce learning rate to 0.00125
reduce learning rate to 0.000625
reduce learning rate to 0.0003125
reduce learning rate to 0.00015625
reduce learning rate to 7.8125e-05
reduce learning rate to 3.90625e-05
Final Loss Value :  1290.2466192487811
training precision = 27.171492204899778
test precision = 23.333333333333332
--------------------------------------
batch_size = 100
Dataset Size :  1347
Final Loss Value :  91.59868972107643
training precision = 75.27839643652561
test precision = 74.22222222222223
```

[527]: Text(0, 0.5, 'loss')

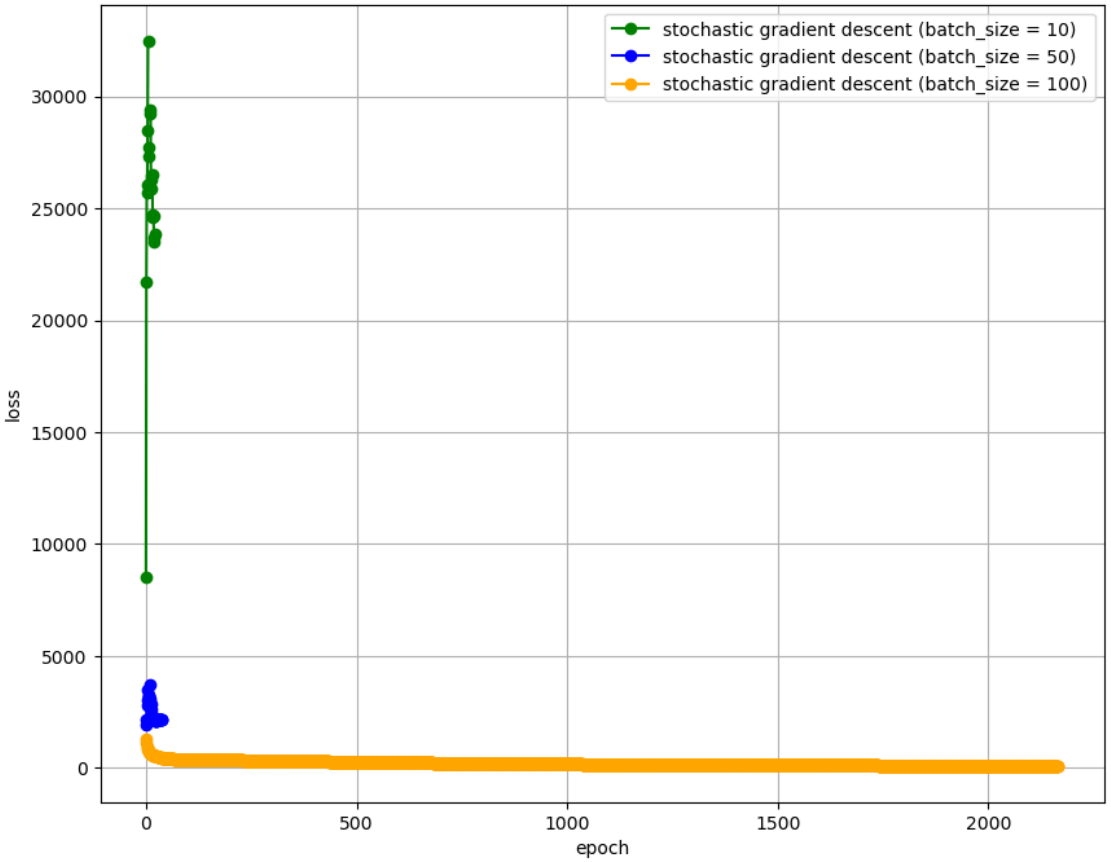


[546]:
```
# evaluation of different batch size
bs = [10, 50, 100]
cl = ['green', 'blue', 'orange']
# Experimented Learning Rates
lr = [3.0e-1, 3.0e-1, 1.0e-2]
fig, ax = plt.subplots(figsize=(10, 8))

for i in range(len(bs)):

    print('--------------------------------------')
    print('batch_size =', bs[i])
    W, b, loss_SGD = LogisticRegression_SGD(X_train, y_train, bs[i], lr[i], eta␣
 ↪= 2.0e-1, eps = 1.0e-4, max_epoch = 2500)

    print("Final Loss Value : ", loss_SGD[-1])
    prec_train = calculate_precision(y_train, W, X_train)
    print('training precision =', prec_train)

    prec_test = calculate_precision(y_test, W,  X_test)
```

```
    print('test precision =', prec_test)

    plt.plot(loss_SGD, c = cl[i], ls = '-', marker = 'o', label = 'stochastic␣
 ↪gradient descent (batch_size = ' + str(bs[i]) + ')')

plt.grid()
plt.legend()
plt.xlabel('epoch')
plt.ylabel('loss')
```

```
----------------------------------------
batch_size = 10
Dataset Size :  1347
reduce learning rate to 0.15
reduce learning rate to 0.075
Final Loss Value :  23823.882027049494
training precision = 32.44246473645137
test precision = 35.55555555555556
----------------------------------------
batch_size = 50
Dataset Size :  1347
/var/folders/c9/3q_3q36n4k9_t4gnwrhyd7s80000gn/T/ipykernel_72269/3995326649.py:3
8: RuntimeWarning: divide by zero encountered in log
  Zlog = -np.log(Z)
reduce learning rate to 0.15
reduce learning rate to 0.075
reduce learning rate to 0.0375
reduce learning rate to 0.01875
reduce learning rate to 0.009375
reduce learning rate to 0.0046875
reduce learning rate to 0.00234375
reduce learning rate to 0.001171875
reduce learning rate to 0.0005859375
reduce learning rate to 0.00029296875
reduce learning rate to 0.000146484375
Final Loss Value :  2185.15948244421
training precision = 70.2301410541945
test precision = 70.0
----------------------------------------
batch_size = 100
Dataset Size :  1347
Final Loss Value :  91.29047788240926
training precision = 73.57089829250185
test precision = 72.0
```

[546]: Text(0, 0.5, 'loss')


stochastic gradient descent (batch_size = 10)
stochastic gradient descent (batch_size = 50)
stochastic gradient descent (batch_size = 100)
loss
epoch
30000
25000
20000
15000
10000
5000
0
0
500
1000
1500
2000

[ ]: