Machine Learning - Assignment 1

Ankur Sharma [2016225]

Q1 Linear Regression

Functions implemented by me:

- *def feature_normalisation(X):* Normalises X. Used for feature scaling.
- *def gradient_descent(X, y, alpha=0.01, time=1000, lam=None, reg='ridge'):* Given X and y, and other parameters like learning rate, no of epochs, regularization penalty, it return rmse history and weights history array, as calculated by gradient descent algorithm, to plot the graph.

Learning Rate: 0.01
Time (no. of iterations) = 1000
Hyperparameter for L2 Regularization: 0.1
Hyperparameter for L1 Regularization: 0.01

(i). R(w) = 0 :

Train RMSE for Fold: 1 : 0.5150127907702405
Validation RMSE for Fold: 1 : 0.5138504038081531

Train RMSE for Fold: 2 : 0.5219753795369334
Validation RMSE for Fold: 2 : 0.4795523930269948

Train RMSE for Fold: 3 : 0.5305167893013567
Validation RMSE for Fold: 3 : **0.4432167362929887** [Fold with best Validation RMSE]

Train RMSE for Fold: 4 : 0.48830829761068484
Validation RMSE for Fold: 4 : 0.6019412168341801

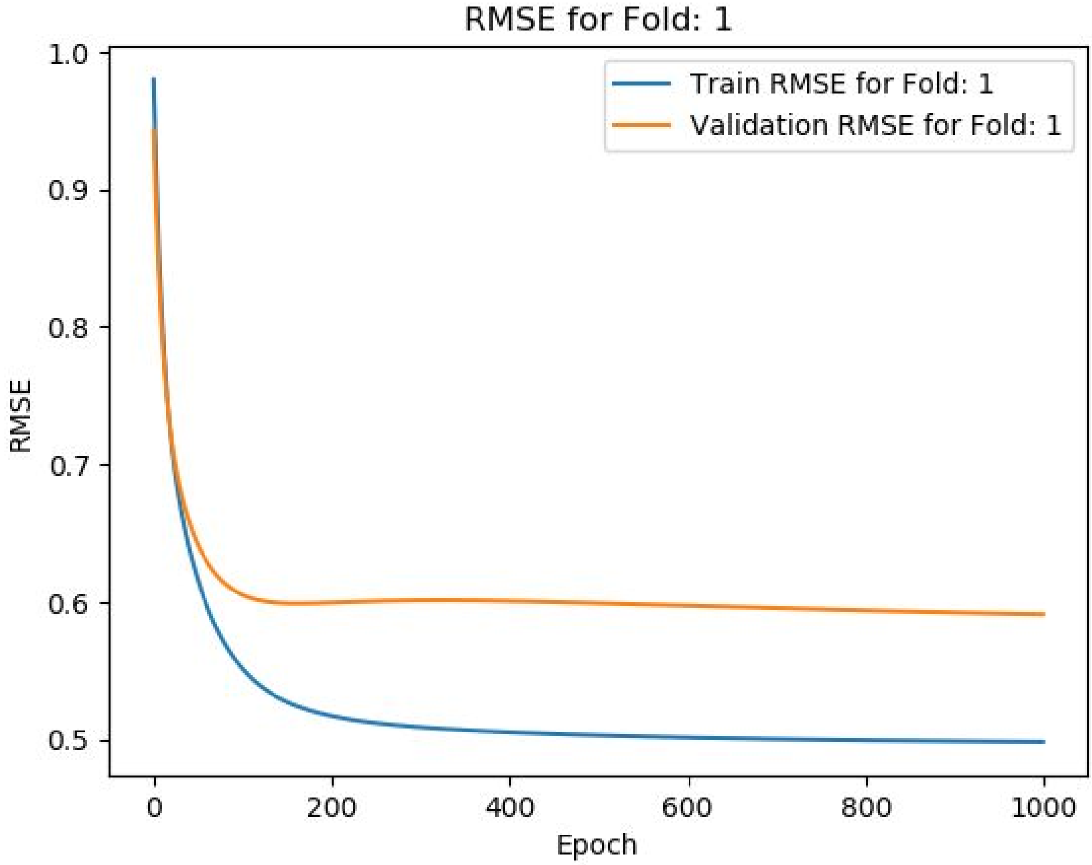
Train RMSE for Fold: 5 : 0.4943770849351848
Validation RMSE for Fold: 5 : 0.5988337821271202

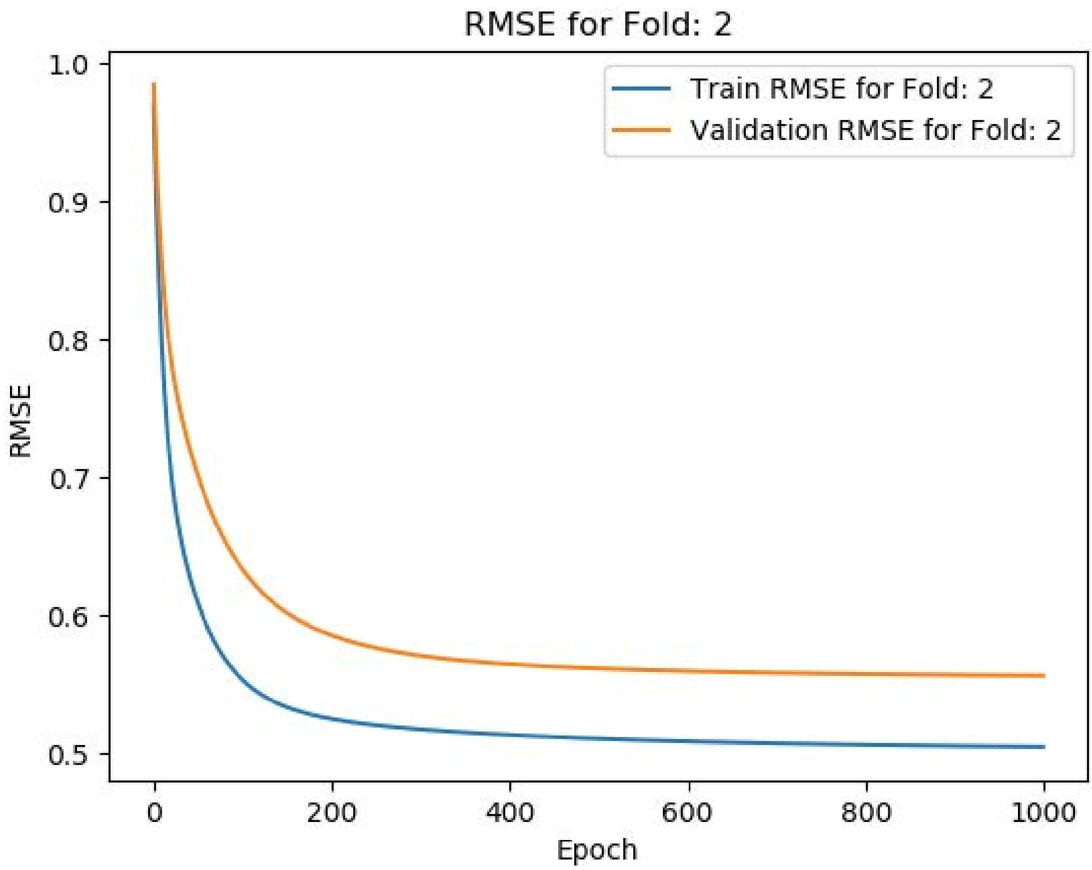
Train Mean RMSE over all Folds 0.5100380684308801
Validation Mean RMSE over all Folds 0.5274789064178874

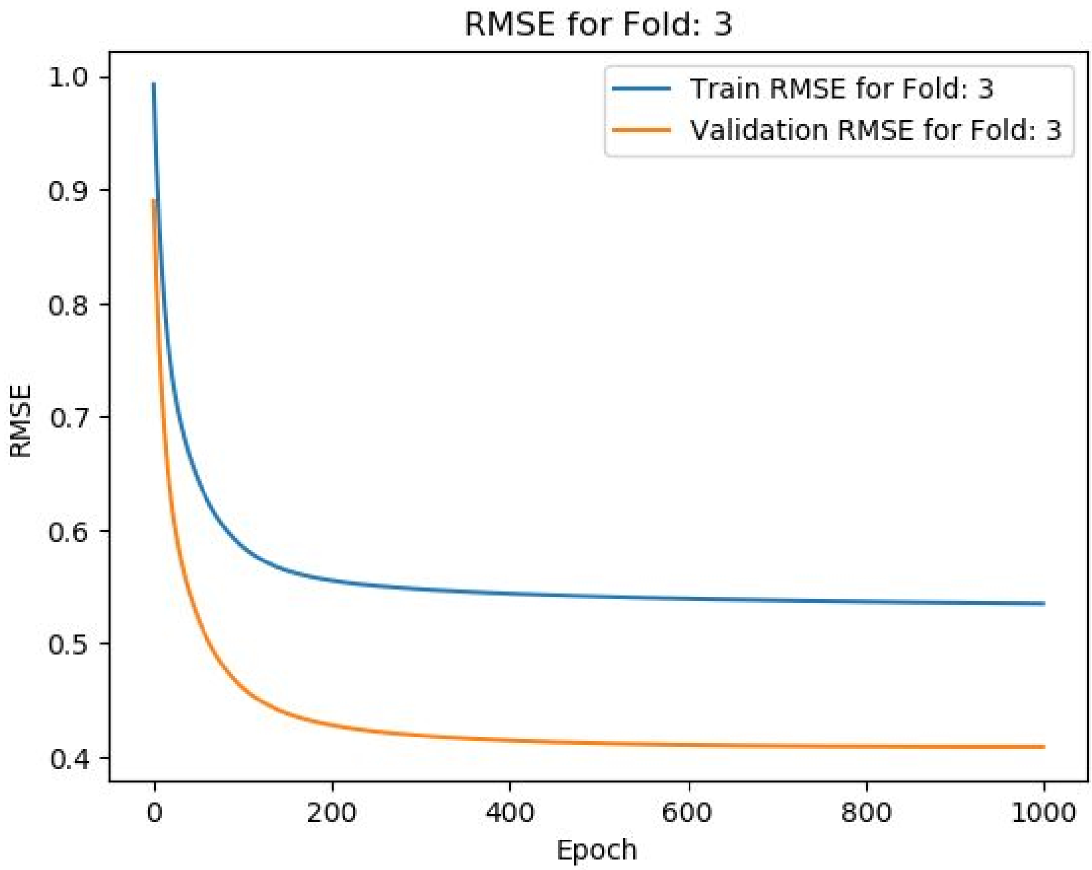
Train STD RMSE over all Folds 0.016149765392958374
Validation STD RMSE over all Folds 0.06359080757996508

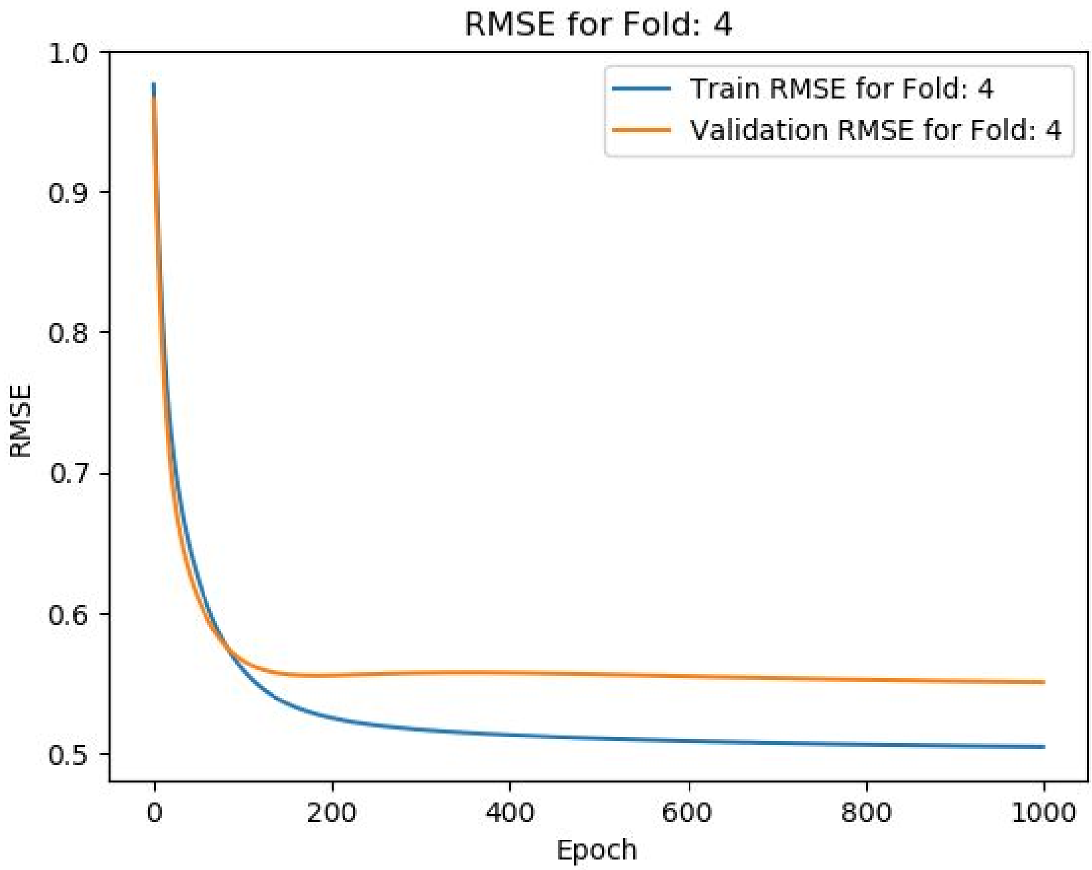
Hence, Train Mean and Standard Deviation : $(0.5100380684308801 \pm 0.016149765392958374)$
Validation Mean and Standard Deviation: $(0.5274789064178874 \pm 0.06359080757996508)$

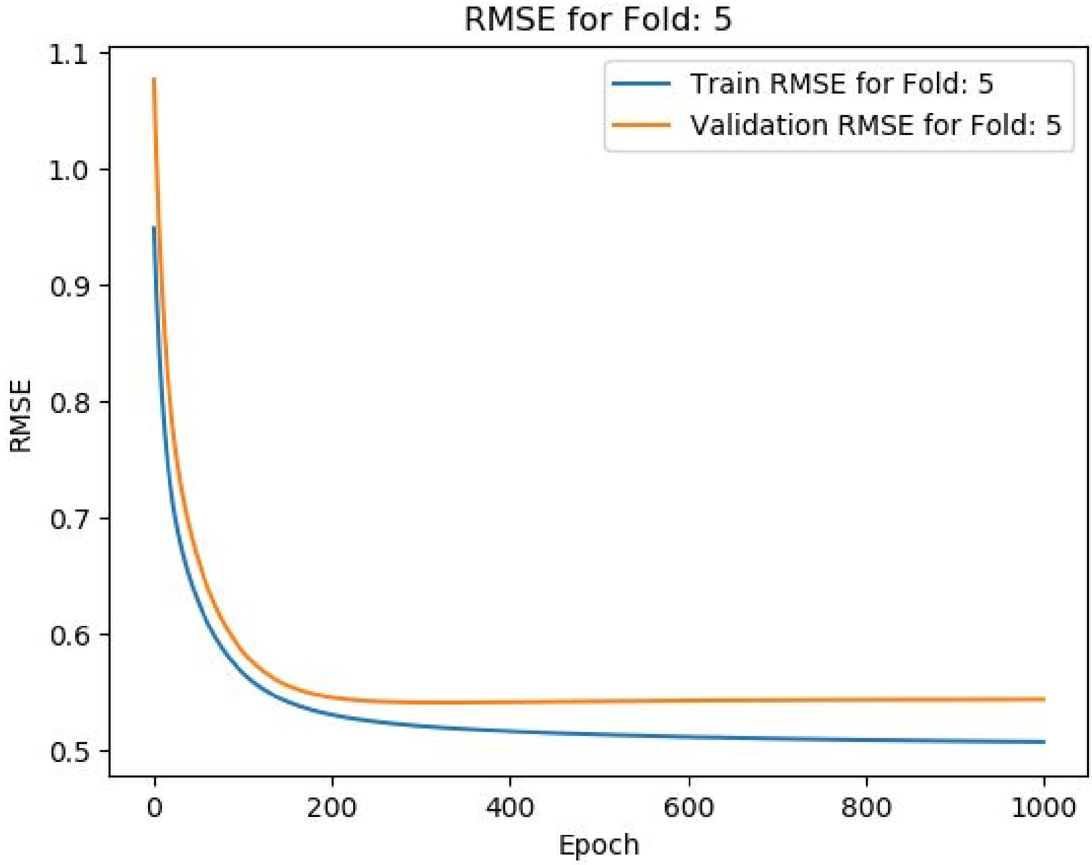
RMSE Plots:

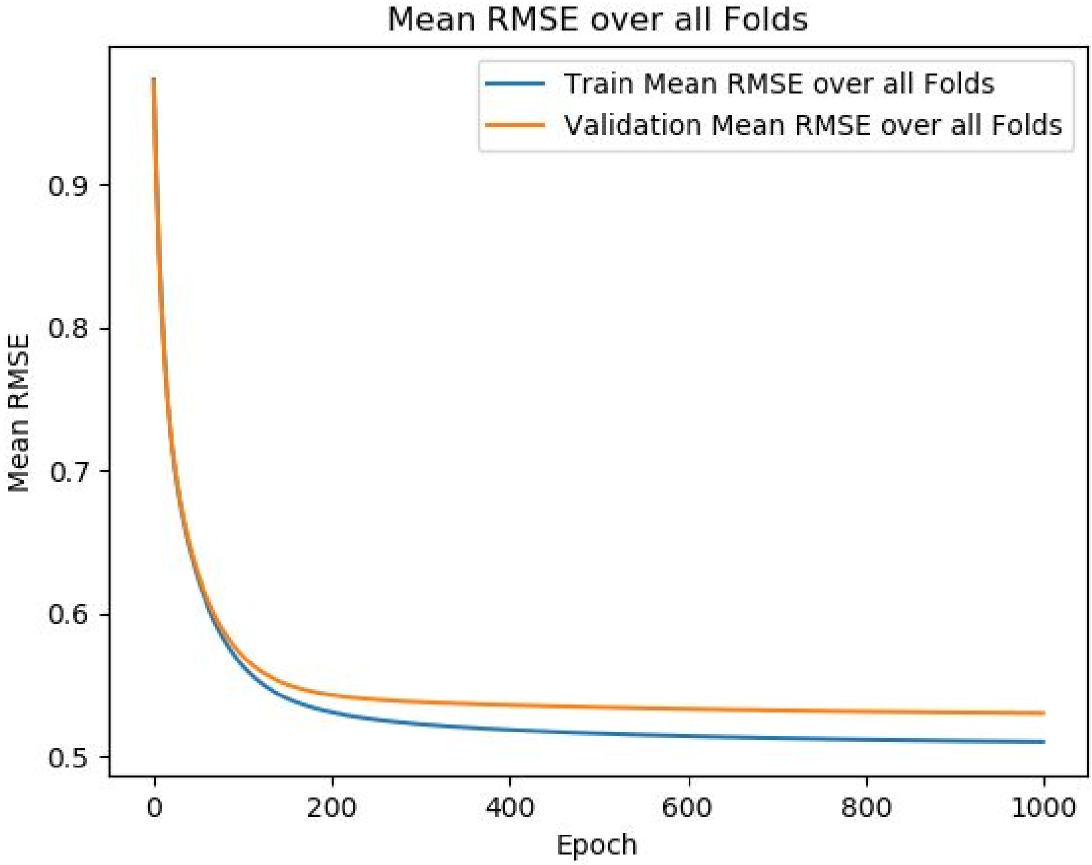

RMSE for Fold: 1
Train RMSE for Fold: 1
Validation RMSE for Fold: 1
RMSE
Epoch
1.0
0.9
0.8
0.7
0.6
0.5
0
200
400
600
800
1000

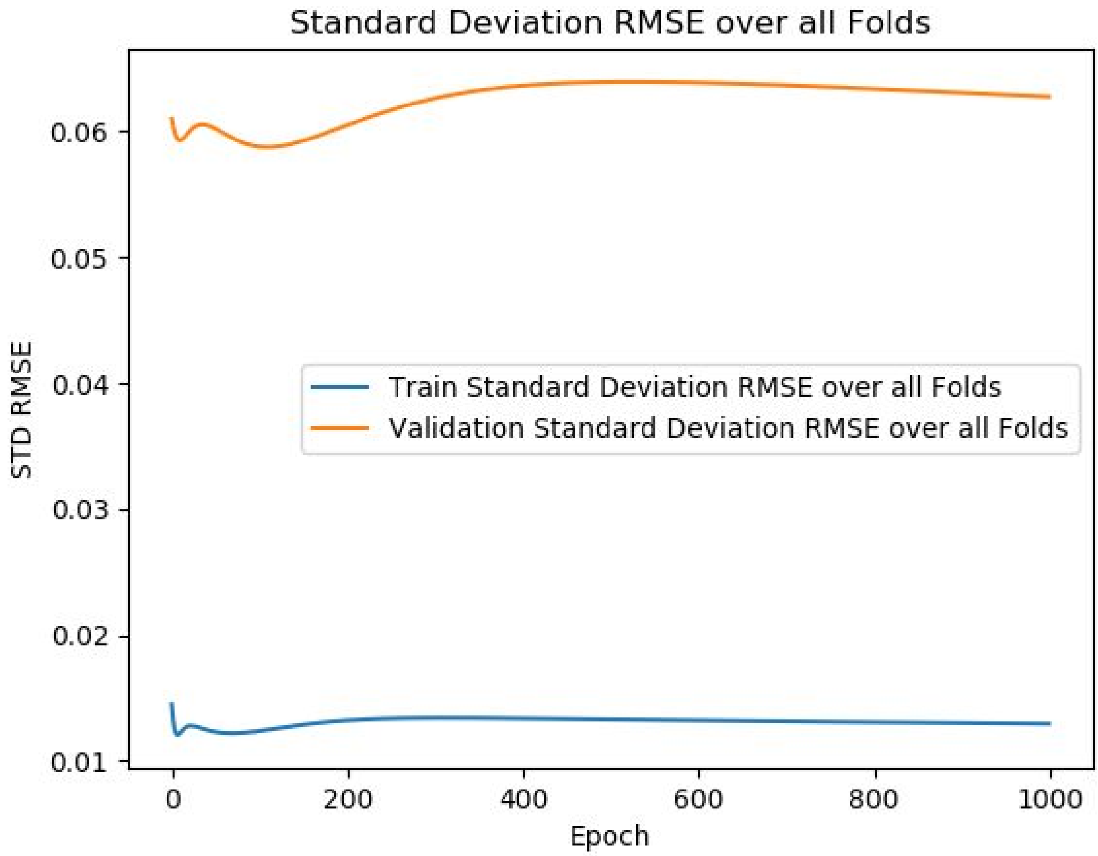

RMSE for Fold: 2
Train RMSE for Fold: 2
Validation RMSE for Fold: 2
RMSE
1.0
0.9
0.8
0.7
0.6
0.5
0
200
400
600
800
1000
Epoch


RMSE for Fold: 3
Train RMSE for Fold: 3
Validation RMSE for Fold: 3
RMSE
Epoch
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0
200
400
600
800
1000


RMSE for Fold: 4
Train RMSE for Fold: 4
Validation RMSE for Fold: 4
RMSE
Epoch
1.0
0.9
0.8
0.7
0.6
0.5
0
200
400
600
800
1000


RMSE for Fold: 5
Train RMSE for Fold: 5
Validation RMSE for Fold: 5
RMSE
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0
200
400
600
800
1000
Epoch


Mean RMSE over all Folds
Train Mean RMSE over all Folds
Validation Mean RMSE over all Folds
Mean RMSE
0.9
0.8
0.7
0.6
0.5
0
200
400
600
800
1000
Epoch

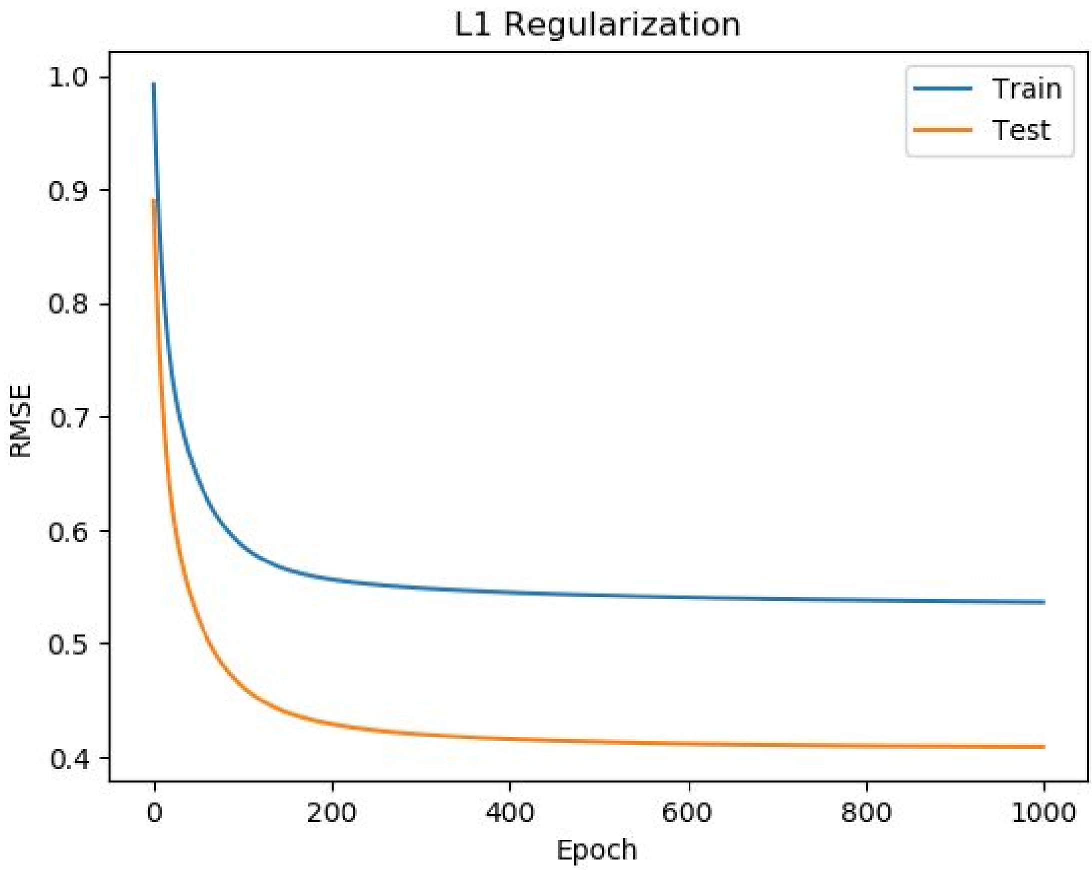


(ii). Regularization:

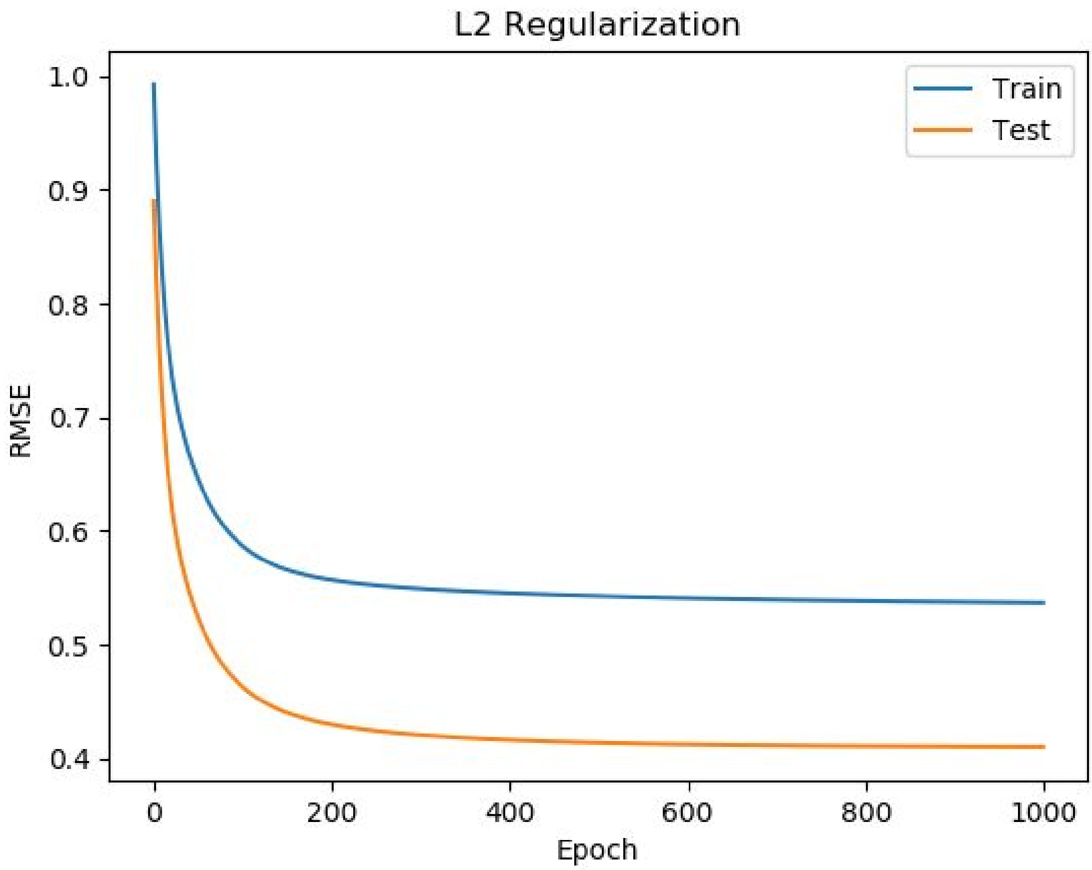
a. **L2 regularization:**
RMSE on Train Set: 0.531960882767265
RMSE on Test Set: 0.4384584080257893

b. **L1 regularization:**
RMSE on Train Set: 0.5315079369788621
RMSE on Test Set: 0.4430745908375048

RMSE Plots:



L1 Regularization
Train
Test
RMSE
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0
200
400
600
800
1000
Epoch



(iii) Comparison of models:

a. Linear Regression Least RMSE on Validation Set(Fold 3): **0.4432167362929887**
b. L2 Regression RMSE on Test Set: **0.4384584080257893**
c. L1 Regression RMSE on Test Set: **0.4430745908375048**

Clearly, **RMSE(Least-Squared) > RMSE(L1) > RMSE(L2).** Hence, L2 is the best model.
Conclusion: All the three models have trains RMSE greater than their Validation RMSE which implies all the algorithms generalises data well and hence, does not overfit the data. Difference in RMSE values of train and test data is very low and also, the RMSE values are not that high which implies that none of the models underfit the data. Hence, all the three models are good fit.

Q2 Logistic Regression

Functions implemented by me:

- *def logistic_regression(penalty='l2'):* Given regularization penalty, it runs scikit learn Logistic regression method on training data. I am also saving my model locally once it is trained using pickle so that next time I want to some computation on my trained classifier, I don't have to re-train it.

(i) L2 Regularization:
Train Accuracy for 0th digit: 0.9842

Test Accuracy for 0th digit: 0.9834

Train Accuracy for 1th digit: 0.9878
Test Accuracy for 1th digit: 0.9854

Train Accuracy for 2th digit: 0.9514
Test Accuracy for 2th digit: 0.9487

Train Accuracy for 3th digit: 0.9472
Test Accuracy for 3th digit: 0.9467

Train Accuracy for 4th digit: 0.9722
Test Accuracy for 4th digit: 0.971

Train Accuracy for 5th digit: 0.9528
Test Accuracy for 5th digit: 0.9457

Train Accuracy for 6th digit: 0.9792
Test Accuracy for 6th digit: 0.9791

Train Accuracy for 7th digit: 0.9724
Test Accuracy for 7th digit: 0.9666

Train Accuracy for 8th digit: 0.9204
Test Accuracy for 8th digit: 0.917

Train Accuracy for 9th digit: 0.9308
Test Accuracy for 9th digit: 0.9207

(ii) L1 Regularization:
Train Accuracy for 0th digit: 0.9848
Test Accuracy for 0th digit: 0.9866

Train Accuracy for 1th digit: 0.9888
Test Accuracy for 1th digit: 0.9896

Train Accuracy for 2th digit: 0.964
Test Accuracy for 2th digit: 0.9647

Train Accuracy for 3th digit: 0.9462
Test Accuracy for 3th digit: 0.9457

Train Accuracy for 4th digit: 0.9726
Test Accuracy for 4th digit: 0.9718

Train Accuracy for 5th digit: 0.9556
Test Accuracy for 5th digit: 0.9539

Train Accuracy for 6th digit: 0.9788
Test Accuracy for 6th digit: 0.9733

Train Accuracy for 7th digit: 0.9716
Test Accuracy for 7th digit: 0.9706

Train Accuracy for 8th digit: 0.9232
Test Accuracy for 8th digit: 0.9172

Train Accuracy for 9th digit: 0.9328
Test Accuracy for 9th digit: 0.9302

(iii) Comparison:
Train and test Accuracy for each digit on both the models is greater than 90% which implies none of the models overfit and underfit . L1 regularization has slightly better accuracy than L2. Hence, both the models are good fit.

Assignment 1 : Theory questions

Q3 In logistic regression,
we have,
log (odds) = $\omega_0 + \omega_1 x_1 + \omega_2 x_2 + \dots + \omega_d x_d$
$= \omega^T x$ where d is no. of features

$\Rightarrow$ log (odds) is the linear combination of x_i's and parameters ω_i's.

let $h_\omega = \omega^T x$.
h_ω is our hypothesis
we pass our hypothesis to sigmoid function which models the probability.

Since hypothesis $\omega^T x$ is linear $\therefore$ we have linear decision boundary with equation $\omega^T x = 0$ (eqn of hyper plane)

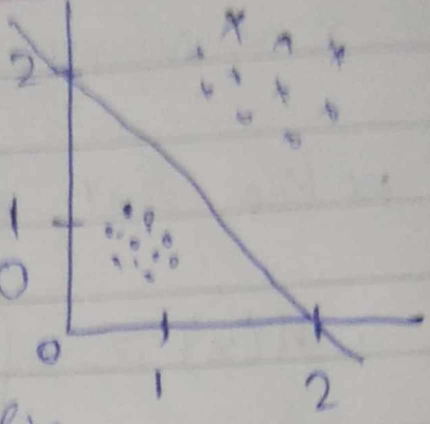
Conclusion:

1. $\omega^T x$ is a linear combination of x_i's $\Rightarrow$ linear decision boundary
2. Sigmoid function is just to spit out probability [0, 1] (given our hypothesis) and has nothing to do with seperating the data linearly or non-linearly.

$$\left[\phi(\omega^T x) = p = \frac{1}{1 + e^{-\omega^T x}} \right]$$

$\Rightarrow$ $-\infty < \omega^T x < \infty$, $0 \le \phi(\omega^T x) \le 1$
$\Rightarrow$ Sigmoid function squishes $\omega^T x$ to spit out probability.

For example :



here, we have
hypothesis : $-x+2-y=0$

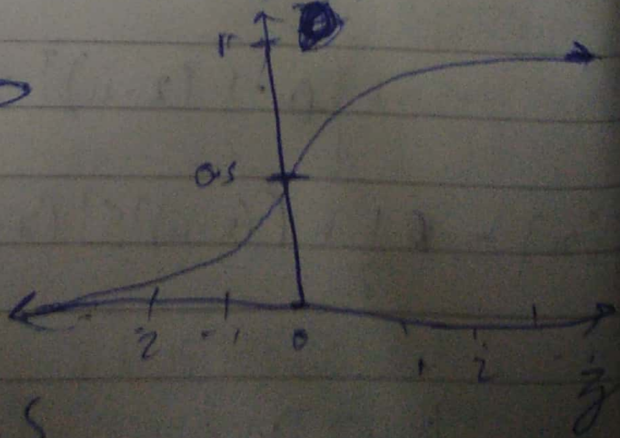
Which is the eq^n of a line.

Let (x^*, y^*) is a point above this line
then $y^* > -x^*+2$
and (x^o, y^o) is a point below this line
then $y^o < -x^o+2$

or, $y^* + x^* - 2 > 0$
$y^o + x^o - 2 < 0$

we pass this to sigmoid function and it returns the probability of ~~point above the line or below it.~~
point $\in \{o\}$ and $\{x\}$

~~we~~



if $z > 0$
$P(z = x) \geq 0.5$
else
$P(z = x) \leq 0.5$

Scanned by CamScanner

Q5 $x \sim N(\mu, \Sigma)$

we know that entropy of x is given by

$$H[x] = -\int_{-\infty}^{\infty} f_x(x) \left(\ln f_x(x)\right) . dx$$

where $f_x(x)$ is pdf of x

we know that

$$f_x(x) = \frac{\exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)}{\sqrt{(2\pi)^D |\Sigma|}}$$

where D is dimensionality of x.

Now,

$$H[x] = -E[\ln f_x(x)].$$

$$\left[\text{as } E[g(y)] = \int_{-\infty}^{\infty} P_y . g(y) . dy\right]$$

$$\therefore H(x) = -E\left[\ln\left(\frac{\exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)}{\sqrt{(2\pi)^D |\Sigma|}}\right)\right]$$

$$= -E\left[-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu) - \frac{1}{2}\left(D \ln 2\pi + \ln|\Sigma|\right)\right]$$

$$H[x] = \frac{1}{2} E\left[(x-\mu)^T \Sigma^{-1} (x-\mu)\right] + \frac{1}{2}\left(D \ln 2\pi + \ln|\Sigma|\right)$$

We know that,

~~Trace function, $Tr(X^T A X) = X^T A X$~~

Scanned by CamScanner

Trace function, $Tr(x^T y x) = x^T y x = Tr(y x x^T)$

$\therefore E[x^T y x] = E[Tr(y x x^T)] = Tr(E[y x x^T])$ — (?)

using (?), we get:

$$H[x] = \frac{1}{2} Tr[E[\Sigma^{-1} (x-\mu)(x-\mu)^T]] + \frac{1}{2}(D \ln 2\pi + \ln |\Sigma|)$$

a Σ^{-1} is a constant

$$H[x] = \frac{1}{2} Tr(\Sigma^{-1} E[(x-\mu)(x-\mu)^T]) + \frac{1}{2}(D \ln 2\pi + \ln|\Sigma|)$$

Now, $E[(x-\mu)(x-\mu)^T] = \Sigma$

$$H[x] = \frac{1}{2} Tr(\Sigma^{-1}\Sigma) + \frac{1}{2}(D \ln 2\pi + \ln |\Sigma|)$$

$$= \frac{1}{2} Tr(I) + \frac{1}{2}(D \ln 2\pi + \ln |\Sigma|)$$

↳ identity matrix of D dimension

$$H[x] = \frac{1}{2}\left(\sum_{i=1}^{D} 1\right) + \frac{1}{2}(D \ln 2\pi + \ln |\Sigma|)$$

$$H[x] = \frac{D}{2} + \frac{D}{2} \ln 2\pi + \frac{1}{2} \ln |\Sigma|$$

$$H[x] = \frac{\ln |\Sigma|}{2} + \frac{D}{2}(1 + \ln 2\pi)$$

Scanned by CamScanner

4. Explain the logit transformation & derive the expression for logistic regression.

Logistic Regression is used to classify binary classes. The idea is to use our good old 'gradient descent' for linear regression but in logistic regression $Y \in \{0, 1\}$, not $Y \in R$.

$$Pr(y|x; p) = \begin{cases} p & y=1 \\ 1-p & y=0 \end{cases}$$

$$= p^{y} (1-p)^{(1-y)}$$

$$\text{odds} = p/1-p$$

odds map 'p' from [0,1] to [0,∞)
odds is monotonically increasing function so is log (odds)
and log (odds) maps odds from [0,∞] to (-∞, ∞).

∴ Range of log(odds) is (-∞, ∞)
we can use linear regression to predict this. Once we are able to predict log(odds), we can calculate p from it b/c odds is a invertible function.

$$\therefore \log(\text{odds}) = w^T x$$

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right) = w^T x$$

$\therefore \log\left(\frac{p}{1-p}\right) = w^T x$

$\frac{p}{1-p} = e^{w^T x} \Rightarrow p = e^{w^T x} - p e^{w^T x}$

$\Rightarrow p\left(1 + e^{w^T x}\right) = e^{w^T x}$

$p = \frac{e^{w^T x}}{1 + e^{w^T x}} = \frac{1}{1 + e^{-w^T x}}$

This gives us the $Pr(Y=1) = p = \frac{1}{1+e^{w^T x}}$

and $1-p = \frac{e^{-w^T x}}{1+e^{-w^T x}} = \frac{1}{1+e^{w^T x}}$

→ log likelihood:

$$l(w) = \sum_{i=1}^{N} \log\left[\left(p(x_i;w)\right)^{y_i} \cdot \left(1-p(x_i;w)\right)^{1-y_i}\right]$$

$$= \sum_{i=1}^{N} y_i \log p(x_i;w) + (1-y_i) \log(1-p(x_i;w))$$

$$= \sum_{i=1}^{N} y_i \left(\log p(x_i;w) - \log(1-p(x_i;w))\right) + \log\left(1-p(x_i;w)\right)$$

$$= \sum_{i=1}^{N} y_i \left(\underbrace{\log\left(\frac{p(x_i;w)}{1-p(x_i;w)}\right)}_{w^T x}\right) + \log\underbrace{\left(1-p(x_i;w)\right)}_{\frac{1}{1+e^{w^T x}}}$$

$$l(w) = \sum_{i=1}^{N} y_i w^T x - \log\left(1+e^{w^T x}\right)$$

Since this is a concave fn we will use gradient ascent.

$$\frac{\partial}{\partial \omega_j} l(\omega) = \sum_{i=1}^{N} \left(y_i \frac{\partial \omega^T x_i}{\partial \omega_j} - \frac{e^{\omega^T x}}{1+e^{\omega^T x}} \frac{\partial \omega^T x}{\partial \omega_j} \right)$$

$$= \sum_{i=1}^{N} \left(y_i x_{ij} - x_{ij} \, p(x_i ; \omega) \right)$$

$$= \sum_{i=1}^{N} x_{ij} \left(y_i - p(x_i ; \omega) \right)$$

∴ Gradient Ascent Rule:

$$\omega_j \longrightarrow \omega_j + \alpha \sum_{i=1}^{N} x_{ij} \left(y_i - p(x_i ; \omega) \right)$$

where x_{ij} mean i^{th} datapoint of j^{th} feature.

Scanned by CamScanner