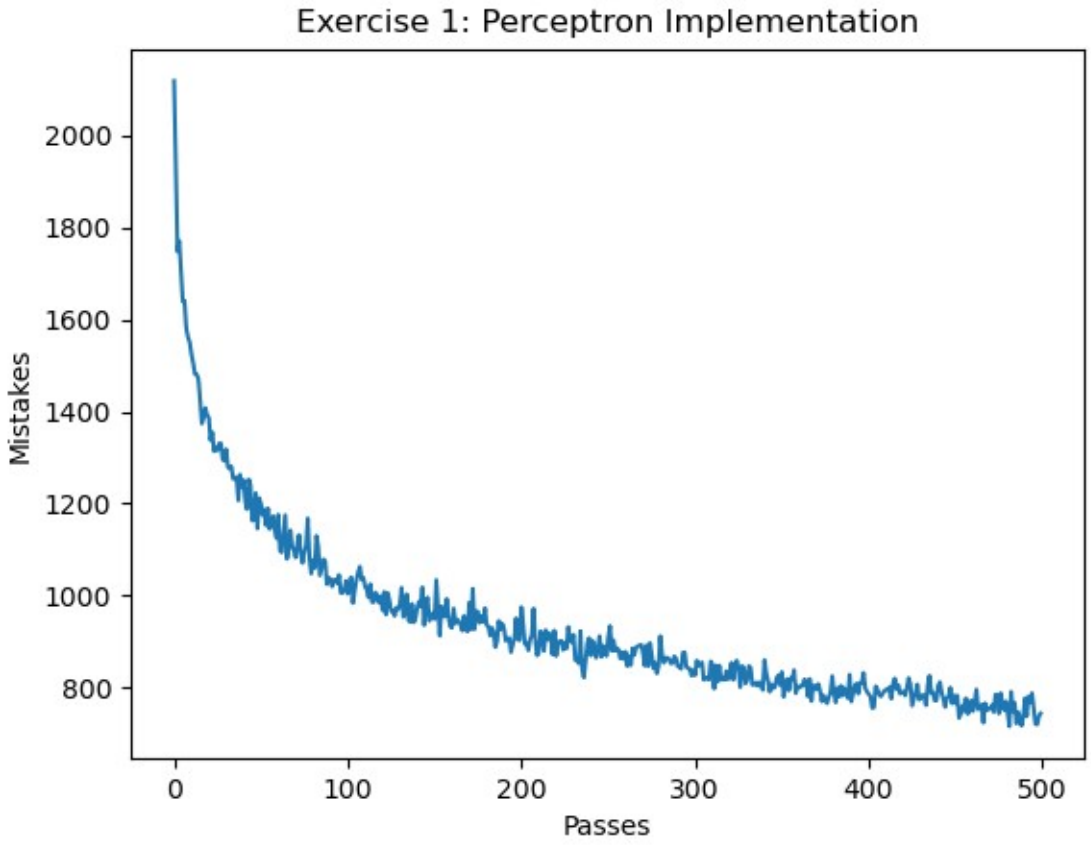
CS480/680: Introduction to Machine Learning Homework 1

Exercise 1: Perceptron Implementation (5 pts)

Implementation provided in zip file labeled “a1q1.py”
To run in terminal execute shell script “./q1.sh”



Andrew Nguyen 20726676

Exercise 2: Regression Implementation (8 pts)

1. Given (1)

$$\frac{1}{2n}\|Xw+b1-y\|_2^2+\lambda\|w\|_2^2$$
$$=\frac{1}{2n}\left(\|Xw+b1-y\|_2^2+2n\lambda\|w\|_2^2\right)$$

Expanding (1) gets

$$\frac{1}{2n}\left(\sum_{i=1}^{n}\left(\sum_{j=1}^{d}X_{ij}w_i+b_i-y_i\right)^2+2n\lambda\sum_{i=1}^{d}w_i\right) \qquad (1.b)$$

since we had a sum of 2 components and can expand each of the components.

Given (2)

$$\frac{1}{2n}\left\|\begin{bmatrix}X & 1_n\\ \sqrt{2\lambda n}I_d & 0_d\end{bmatrix}\begin{bmatrix}w\\ b\end{bmatrix}-\begin{bmatrix}y\\ 0_d\end{bmatrix}\right\|_2^2$$
$$=\frac{1}{2n}\left\|\begin{matrix}Xw+1_nb-y\\ \sqrt{2\lambda n}I_dw\end{matrix}\right\|_2^2$$ we can expand into components to get
$$=\frac{1}{2n}\left(\sum_{i=1}^{n}\left(\sum_{j=1}^{d}X_{ij}w_i+b_i-y_i\right)^2+\sum_{i=1}^{d}\left(\sqrt{2\lambda n}\,w_i\right)^2\right)$$
$$=\frac{1}{2n}\left(\sum_{i=1}^{n}\left(\sum_{j=1}^{d}X_{ij}w_i+b_i-y_i\right)^2+2n\sum_{i=1}^{d}w_i\right) \qquad (2.b)$$

Thus we can see that (1.b) = (2.b) → (1)=(2) as desired

2. Given (1)

$$\underbrace{\frac{1}{2n}\|Xw+b1-y\|_2^2}_{g(w,b)}+\underbrace{\lambda\|w\|_2^2}_{h(w)}$$

Andrew Nguyen 20726676

To take the derivative of (1) we can derive parts g and h separately.
First let's expand g to apply simple derivatives found in Lecture 2c.

$$g(w,b) = \frac{1}{2n}\|Xw + b1 - y\|_2^2$$
$$= \frac{1}{2n}(Xw + b1 - y)^T(Xw + b1 - y)$$
$$= \frac{1}{2n}(X^Tw^T + b^T1^T - y^T)(Xw + b1 - y)$$
$$= \frac{1}{2n}(X^Tw^TXw + b^T1^TXw - y^TXw + X^Tw^Tb1 + b^T1^Tb1 - y^Tb1 - X^Tw^Ty - b^T1^Ty + y^Ty)$$
$$= \frac{1}{2n}(X^Tw^TXw + 2X^Tw^Tb1 - 2X^Tw^Ty + b^T1^Tb1 - 2b^T1^Ty + y^Ty)$$

Using derivative rules and Lecture 2c

$$\frac{\partial}{\partial w}g(w,b) = \frac{1}{2n}(2X^TXw + 2X^Tb1 - 2X^Ty)$$
$$= \frac{1}{n}X^T(Xw + b1 - y)$$

Similarly for h

$$h(w) = \lambda\|w\|_2^2$$
$$= \lambda w^Tw$$
$$= \lambda w^TI_dw$$
$$\frac{\partial}{\partial w}h(w) = \lambda(I_d + I_d^T)w$$
$$= \lambda I_dw + \lambda I_d^Tw$$
$$= 2\lambda w$$

Since $\frac{\partial}{\partial w} = \frac{\partial}{\partial w}g(w,b) + \frac{\partial}{\partial w}h(w)$

$= \frac{1}{n}X^T(Xw + b1 - y) + 2\lambda w$ as desired.

Similarly for $\frac{\partial}{\partial b}$

$$\frac{\partial}{\partial b}g(w,b) = \frac{1}{2n}(2X^Tw^T1 + 1^Tb1 + b^T1^T1 - 21^Ty)$$
$$= \frac{1}{2n}(2I^TXw + 2I^Tb1 - 2I^Ty)$$
$$= \frac{1}{n}1^T(Xw + b1 - y)$$

$\frac{\partial}{\partial b}h(w) = 0$

Thus we have

$\frac{\partial}{\partial b} = \frac{1}{n}1^T(Xw + b1 - y)$ as desired

Andrew Nguyen 20726676

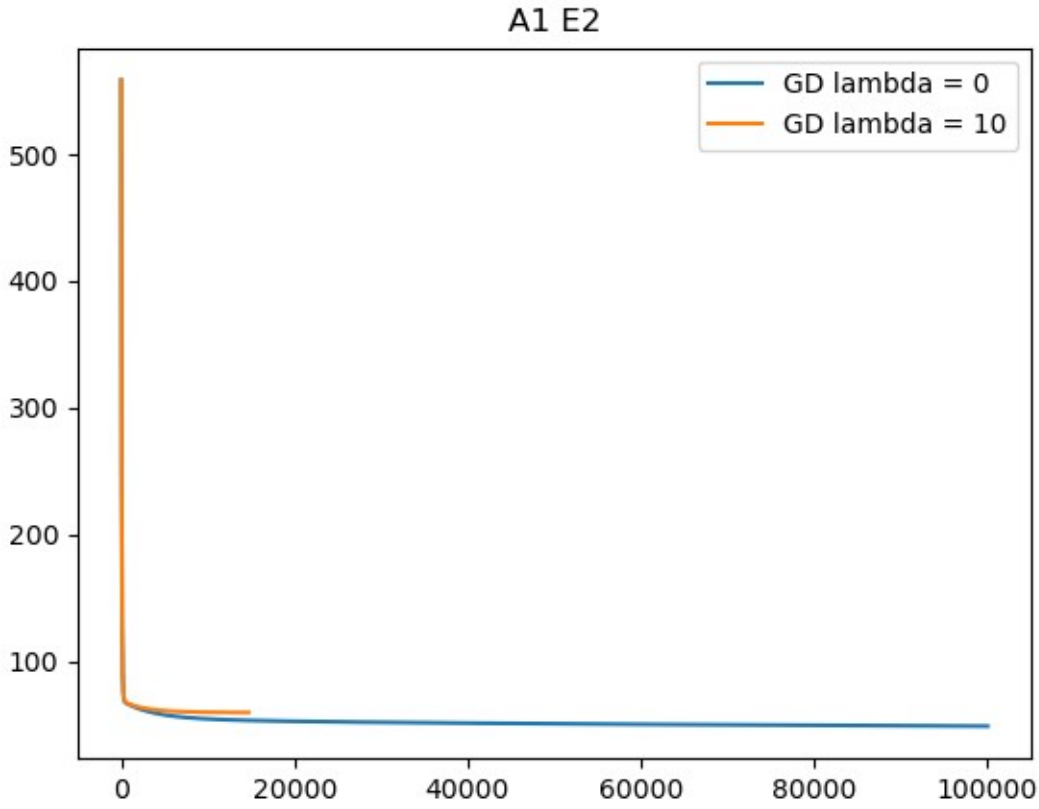
Therefore, we have shown the derivatives of (1) are as desired.

3 and 4. Implementation provided in zip file labeled “a1q2.py”
To run in terminal execute shell script “./q2.sh”

5. Running the implementations oh the Boston housing data set results in the following :
(RR – ridge regression; GD – gradient descent)

Training error for RR with lambda = 0 is 7.416138458765631
Training loss for RR with lambda = 0 is 7.416138458765631
Testing error for RR with lambda = 0 is 185.11147864906565
Training error for RR with lambda = 10 is 7.418708804723804
Training loss for RR with lambda = 10 is 7.418708804723804
Testing error for RR with lambda = 10 is 201.16839695495364
RR took: 0.013806343078613281 seconds
Training error for GD with lambda = 0 is 48.64112292362268
Training loss for GD with lambda = 0 is 48.64112292362268
Testing error for GD with lambda = 0 is 52.30678290155527
Training error for GD with lambda = 10 is 59.36674354271292
Training loss for GD with lambda = 10 is 59.36674354271292
Testing error for GD with lambda = 10 is 60.39069608136287
GD took: 1.8323917388916016 seconds

Andrew Nguyen 20726676



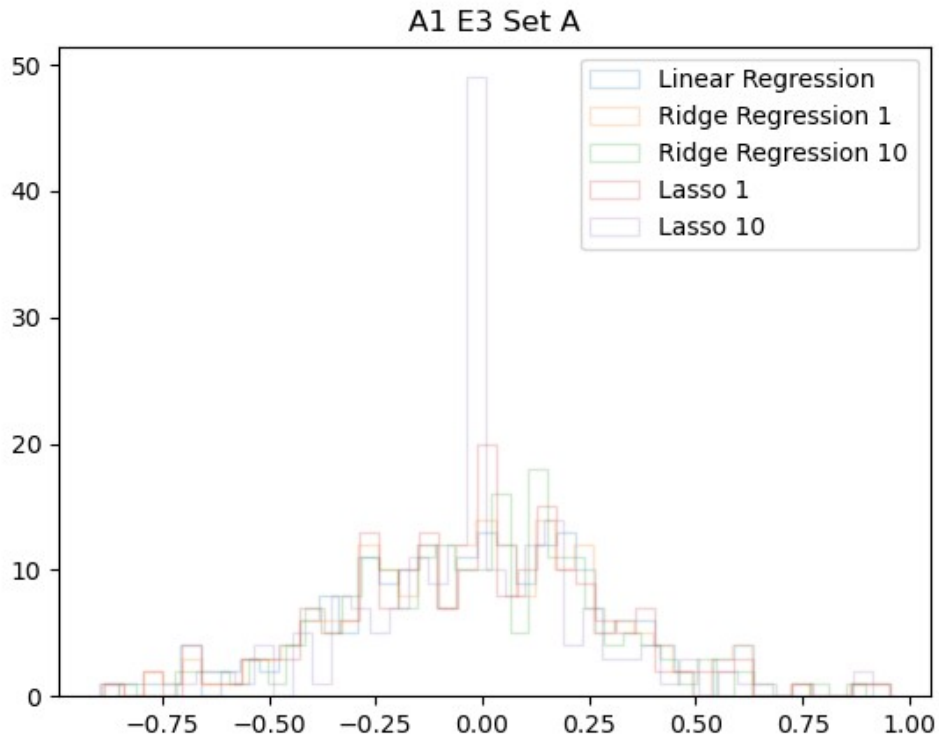
By comparing the running time of the two approaches the ridge regression approach is faster.
I believe the gradient descent implementation to be better as it provides a vastly superior testing error even though it's training error is greater than that of ridge regression. This is also a benefit as it shows that the model output of the gradient descent implementation is not over-fit to the training data set as the ridge regression output is. Since the output of the gradient descent implementation is presumed to be not over-fit, it is more flexible and that is evident in it's results in the testing data set.

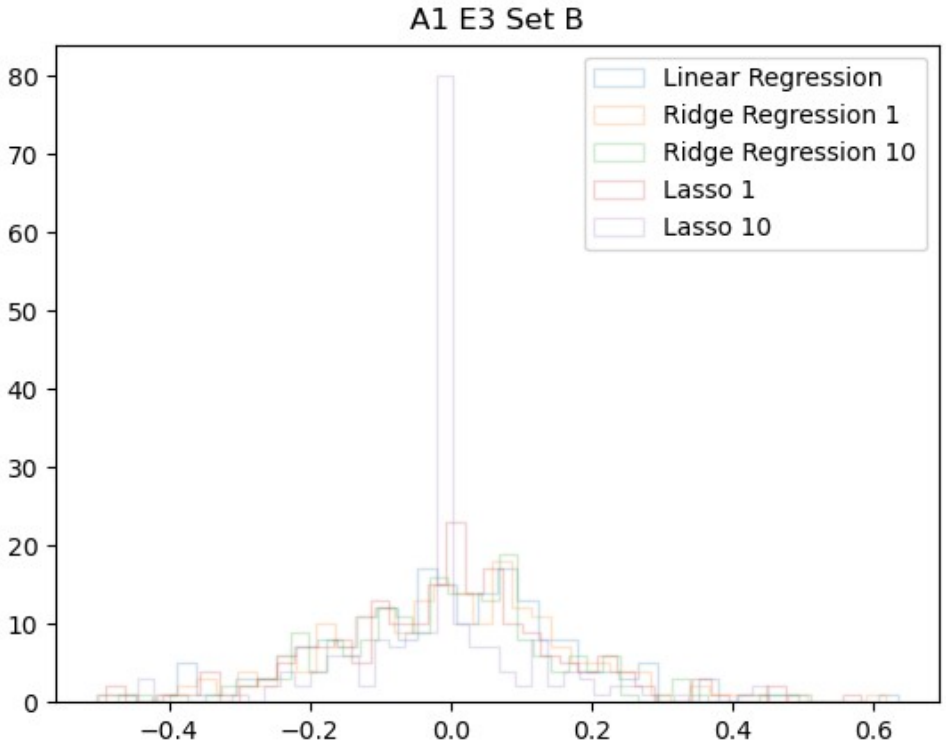
Andrew Nguyen 20726676

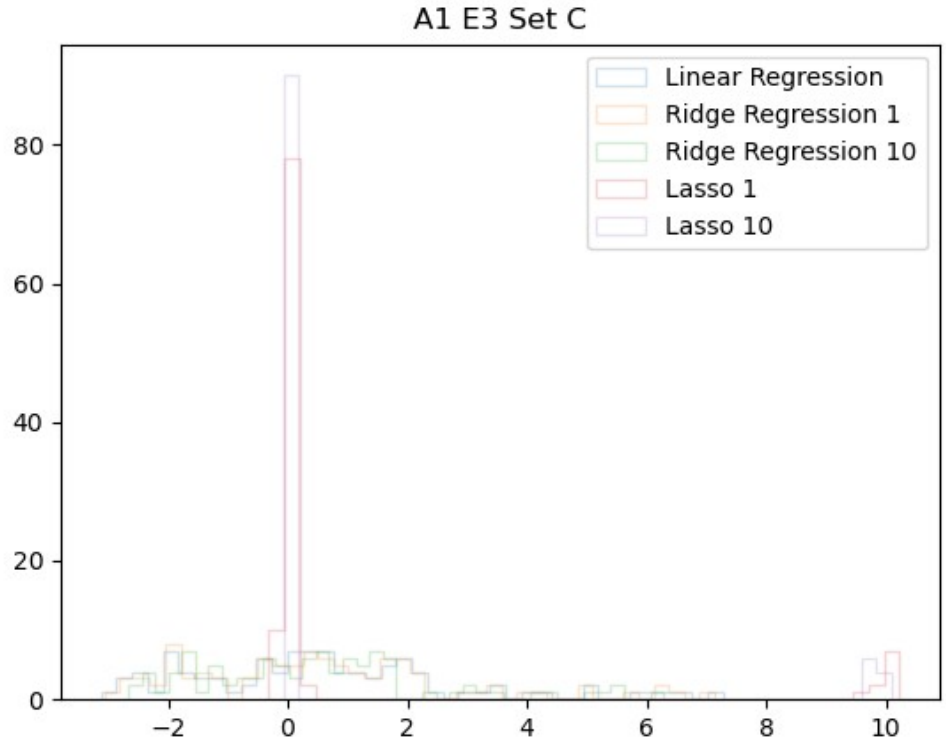
Exercise 3: Playing with Regression (3 pts)

Implementation provided in zip file labeled “a1q3.py”
To run in terminal execute shell script “./q3.sh”

Running the implementation results in:
Average Mean Squared Error for Set A using Linear Regression is: 3.2474001735563336
Average Mean Squared Error for Set A using Ridge Regression 1 is: 3.1393937714571583
Average Mean Squared Error for Set A using Ridge Regression 10 is: 2.778034132092459
Average Mean Squared Error for Set A using Lasso 1 is: 3.020398627748416
Average Mean Squared Error for Set A using Lasso 10 is: 3.602632561104983
Best performer of set A is: Ridge Regression 10
Average Mean Squared Error for Set B using Linear Regression is: 2.7426823746517077
Average Mean Squared Error for Set B using Ridge Regression 1 is: 2.616205111992319
Average Mean Squared Error for Set B using Ridge Regression 10 is: 2.0597132198546957
Average Mean Squared Error for Set B using Lasso 1 is: 2.265165239893118
Average Mean Squared Error for Set B using Lasso 10 is: 1.8096725417753268
Best performer of set B is: Lasso 10
Average Mean Squared Error for Set C using Linear Regression is: 507.7516187628963
Average Mean Squared Error for Set C using Ridge Regression 1 is: 505.2773084049703
Average Mean Squared Error for Set C using Ridge Regression 10 is: 515.8917389668197
Average Mean Squared Error for Set C using Lasso 1 is: 1.8774717034372645
Average Mean Squared Error for Set C using Lasso 10 is: 1.3525617864578996
Best performer of set C is: Lasso 10




A1 E3 Set B
Linear Regression
Ridge Regression 1
Ridge Regression 10
Lasso 1
Lasso 10
80
70
60
50
40
30
20
10
0
−0.4
−0.2
0.0
0.2
0.4
0.6

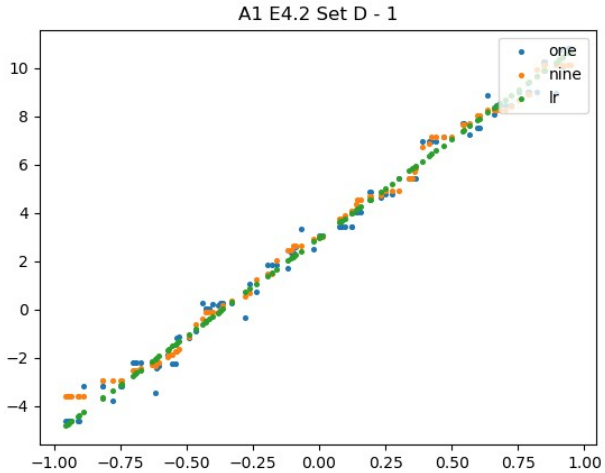

A1 E3 Set C
Linear Regression
Ridge Regression 1
Ridge Regression 10
Lasso 1
Lasso 10
80
60
40
20
0
−2
0
2
4
6
8
10

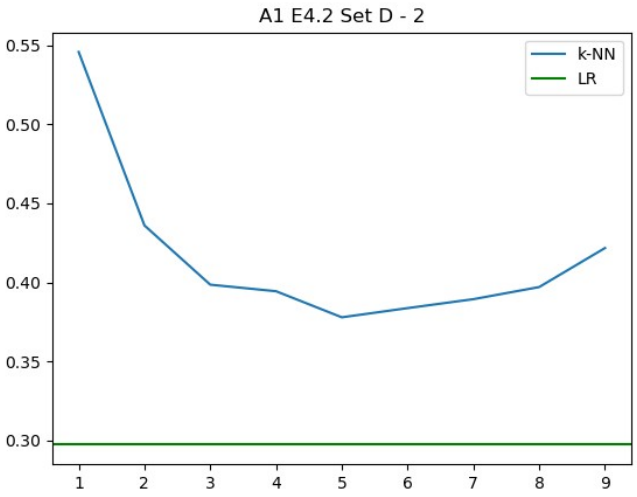
Exercise 4: Nearest Neighbor Regression (7 pts)
Implementation provided in zip file labeled “a1q4.py”
To run in terminal execute shell script “./q4.sh”
1. The implementation for kNN is as follows:

```
def knn(X, y, x, k):
        n = len(y)
        d = []
        for i in range(n):
                d.append(numpy.linalg.norm(x - X[i]))
        diff = d[:]
        kth = kthSmallest(diff, 0, n - 1, k)
        aggrigate = 0
        counter = 0
        for j in range(n):
                If (d[j] <= kth):
                        aggrigate = aggrigate + y[j]
        return aggrigate / k
```

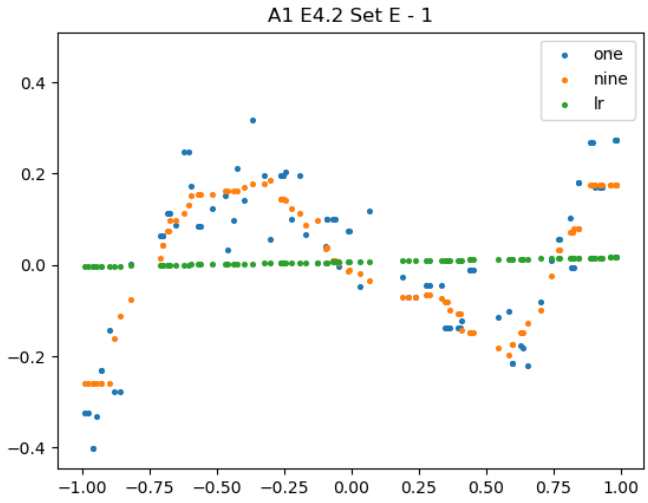
The implementation is in O(nd).
Line 1 and 2 perform in O(1).
The for loop runs n times to find the distance between points which takes d time.
Thus the for loop runs in O(nd).
Copying the array in line 5 takes n time so that we can run quickselect to find the kth smallest distance in line 6.
Running quickselect lets us find the kth smallest distance in an expected complexity of O(n).
Thus lines 5 and 6 each run in O(n) time.
Line 7 and 8 run in O(1).
The final for loop runs in O(n) as it runs n times and it’s inner block runs in O(1).
We can now see that the dominant factor is the first for loop that runs in O(nd) time and thus we can see that the implementation runs in O(nd).

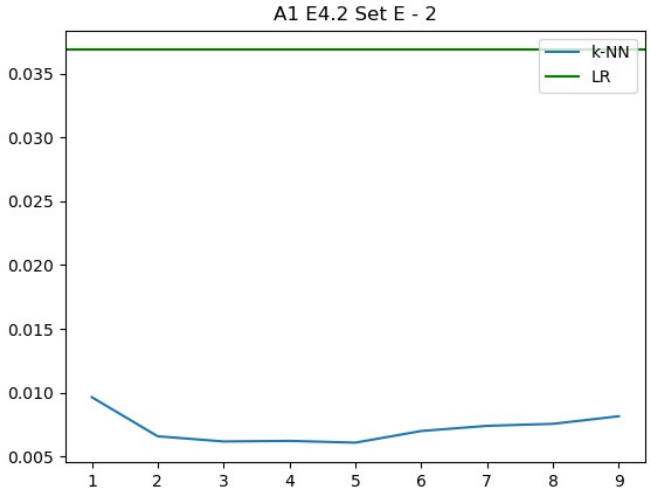
2.Running the implementation on set D results in the following graphs:





Running the implementation on set E results in the following graphs:

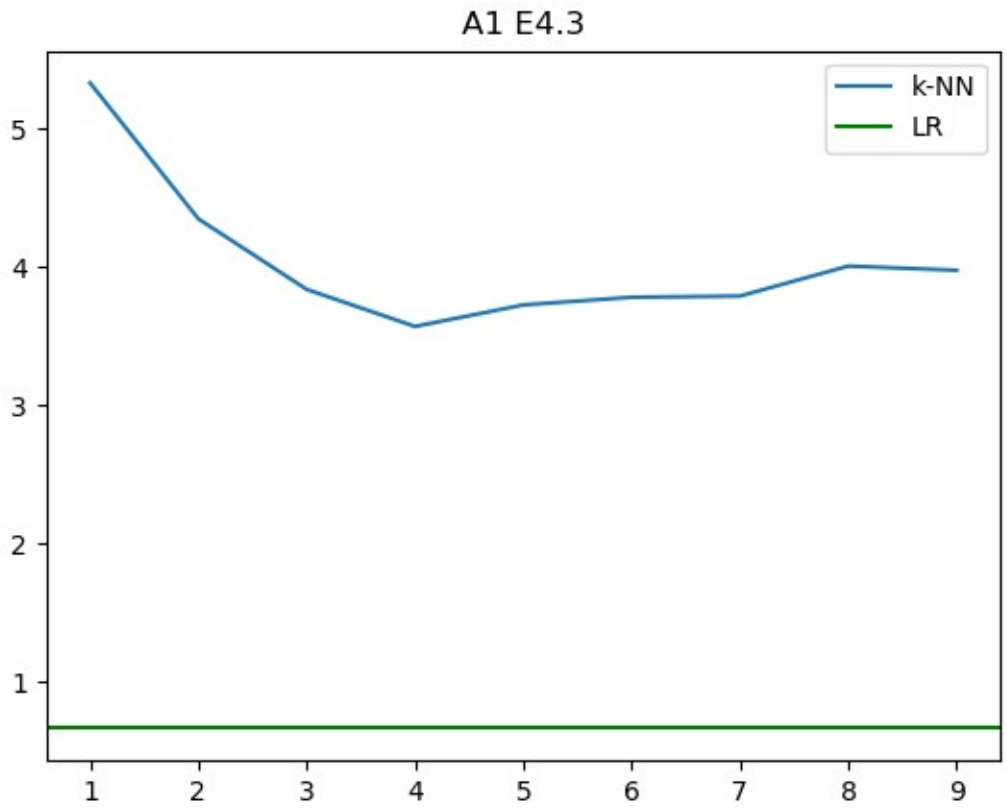




Linear regression performs better on set D as it is likely due to set D being linear in nature. This gives an advantage to the linear regression model since the kNN algorithm does not output a linear model. This causes the test error of the kNN to be greater than that of the linear regression. The kNN solution is not as good a fit for linear test cases as linear regression is.

The kNN implementation performs better on the set E as the set is non-linear. It seems to follow a cubic function which causes the linear regression model to have large testing error as many points will be far away from the linear regression result. Since kNN does not output a linear model, it is able to better fit the set E as it is capable of following along the points in set E causing the points to not be too far away from the predicted result by kNN. Since kNN is able to predict a non linear model, it is better suited to tasks similar to set E.

3. Running the implementation on set F results in the following:



The linear regression model performs better than the kNN model for set F. This is likely due to the kNN being based on euclidean distance. As the dimension increases, the distance between points increases in the higher dimensional space. This causes the kNN to search a larger amount of space and it's accuracy decreases as a result since it uses euclidean distance. Since linear regression finds a vector of best fit, it is generally not affected greatly by higher dimensions other than that the vector will have

another entry. The error of the predicted model from that of the test model should not be much greater than it would be for a lower dimensional model with similar spread.