CS480/680: Introduction to Machine Learning

Homework 1

Due: 11:59 pm, May 25, 2022, submit on LEARN and/or Crowdmark (TBD).

Name: Yusu (Cosmo) Zhao, Student ID: 20761282

Submit your writeup in pdf and all source code in a zip file (with proper documentation). Write a script for each programming exercise so that the TAs can easily run and verify your results. Make sure your code runs! [Text in square brackets are hints that can be ignored.]

Exercise 1: Perceptron Implementation (5 pts)

Convention: All algebraic operations, when applied to a vector or matrix, are understood to be element-wise (unless otherwise stated).

Algorithm 1: The perceptron algorithm.

Input: $X \in \mathbb{R}^{n\times d}$, $\mathbf{y} \in \{-1,1\}^n$, $\mathbf{w} = \mathbf{0}_d$, $b = 0$, $\mathsf{max\_pass} \in \mathbb{N}$
Output: $\mathbf{w}, b, mistake$

```
for t = 1, 2, ..., max_pass do
    mistake(t) ← 0
    for i = 1, 2, ..., n do
        if y_i(⟨x_i, w⟩ + b) ≤ 0 then
            w ← w + y_i x_i                    // x_i is the i-th row of X
            b ← b + y_i
            mistake(t) ← mistake(t) + 1
```

Implement the perceptron in **??**. Your implementation should take input as $X = [\mathbf{x}_1^\top, \ldots, \mathbf{x}_n^\top]^\top \in \mathbb{R}^{n\times d}$, $\mathbf{y} \in \{-1,1\}^n$, an initialization of the hyperplane parameters $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$, and the maximum number of passes of the training set [suggested $\mathsf{max\_pass} = 500$]. Run your perceptron algorithm on the spambase dataset (use the version on the course website), and plot the number of mistakes (y-axis) w.r.t. the number of passes (x-axis).

Ans:

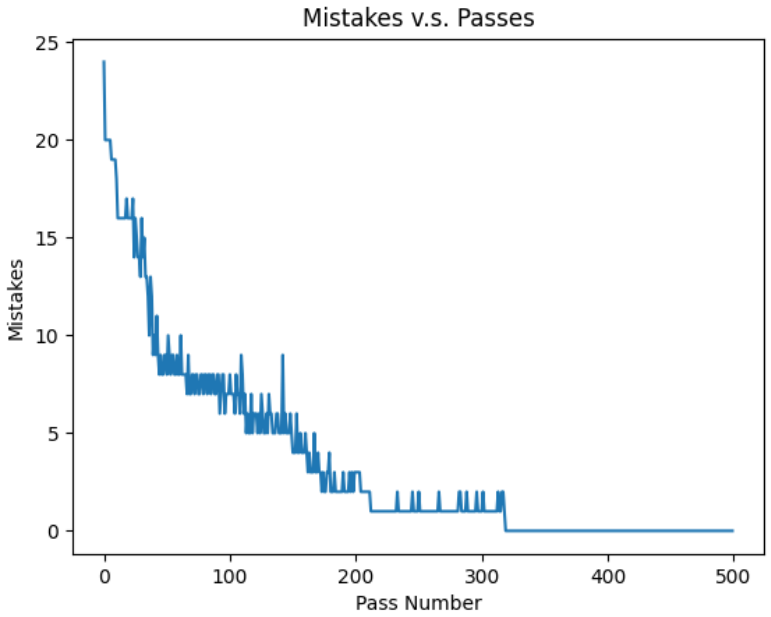

Figure 1: Mistakes v.s. Passes

Exercise 2: Regression Implementation (8 pts)

Recall that ridge regression refers to

$$\min_{\mathbf{w}\in\mathbb{R}^d, b\in\mathbb{R}} \overbrace{\underbrace{\frac{1}{2n}\|X\mathbf{w}+b\mathbf{1}-\mathbf{y}\|_2^2}_{\text{error}}+\lambda\|\mathbf{w}\|_2^2}^{\text{loss}}, \tag{1}$$

where $X \in \mathbb{R}^{n\times d}$ and $\mathbf{y} \in \mathbb{R}^n$ are the given dataset and $\lambda \geq 0$ is the regularization hyperparameter. If $\lambda = 0$, then this is the standard linear regression problem. Observe the distinction between the error (which does not include the regularization term) and the loss (which does).

1. (1 pt) Show that ridge regression can be rewritten as a non-regularized linear regression problem. That is, prove **??** is equivalent to

$$\min_{\mathbf{w}\in\mathbb{R}^d, b\in\mathbb{R}} \frac{1}{2n}\left\|\begin{bmatrix} X & \mathbf{1}_n \\ \sqrt{2\lambda n}I_d & \mathbf{0}_d \end{bmatrix}\begin{bmatrix}\mathbf{w}\\ b\end{bmatrix}-\begin{bmatrix}\mathbf{y}\\ \mathbf{0}_d\end{bmatrix}\right\|_2^2, \tag{2}$$

   where I_d is the d-dimensional identity matrix, and $\mathbf{0}_k$ and $\mathbf{1}_k$ are zero and one column vectors in k dimensions.

   Ans: We will prove that the two expressions are equal

$$\begin{aligned}
\frac{1}{2n}\left\|\begin{bmatrix} X & \mathbf{1}_n \\ \sqrt{2\lambda n}I_d & \mathbf{0}_d \end{bmatrix}\begin{bmatrix}\mathbf{w}\\ b\end{bmatrix}-\begin{bmatrix}\mathbf{y}\\ \mathbf{0}_d\end{bmatrix}\right\|_2^2 &= \frac{1}{2n}\left\|\begin{bmatrix} X\mathbf{w}+\mathbf{1}_n b \\ \sqrt{2\lambda n}I_d\mathbf{w}+\mathbf{0}_d b \end{bmatrix}-\begin{bmatrix}\mathbf{y}\\ \mathbf{0}_d\end{bmatrix}\right\|_2^2 \\
&= \frac{1}{2n}\left\|\begin{bmatrix} X\mathbf{w}+b\mathbf{1}_n-\mathbf{y} \\ \sqrt{2\lambda n}\mathbf{w} \end{bmatrix}\right\|_2^2 \\
&= \frac{1}{2n}\left(\sum_{i=1}^n (X_i\mathbf{w}+b-\mathbf{y}_i)^2+\sum_{i=1}^d(\sqrt{2\lambda n}\mathbf{w}_i)^2\right) \\
&= \frac{1}{2n}\left(\sum_{i=1}^n (X_i\mathbf{w}+b-\mathbf{y}_i)^2+2\lambda n\sum_{i=1}^d \mathbf{w}_i^2\right) \\
&= \frac{1}{2n}\sum_{i=1}^n (X_i\mathbf{w}+b-\mathbf{y}_i)^2+\lambda\sum_{i=1}^d \mathbf{w}_i^2 \\
&= \frac{1}{2n}\|X\mathbf{w}+b\mathbf{1}_n-\mathbf{y}\|_2^2+\lambda\|\mathbf{w}\|_2^2
\end{aligned}$$

   From above, we proved that:

$$\frac{1}{2n}\left\|\begin{bmatrix} X & \mathbf{1}_n \\ \sqrt{2\lambda n}I_d & \mathbf{0}_d \end{bmatrix}\begin{bmatrix}\mathbf{w}\\ b\end{bmatrix}-\begin{bmatrix}\mathbf{y}\\ \mathbf{0}_d\end{bmatrix}\right\|_2^2 = \frac{1}{2n}\|X\mathbf{w}+b\mathbf{1}_n-\mathbf{y}\|_2^2+\lambda\|\mathbf{w}\|_2^2$$

   so the minimum of them are equal, and (1) and (2) are equal.

2. (1 pt) Show that the derivatives of **??** are

$$\frac{\partial}{\partial \mathbf{w}} = \frac{1}{n}X^\top(X\mathbf{w}+b\mathbf{1}-\mathbf{y})+2\lambda\mathbf{w} \tag{3}$$

$$\frac{\partial}{\partial b} = \frac{1}{n}\mathbf{1}^\top(X\mathbf{w}+b\mathbf{1}-\mathbf{y}). \tag{4}$$

   Ans: From question 2, we know that:

$$\frac{1}{2n}\|X\mathbf{w}+b\mathbf{1}_n-\mathbf{y}\|_2^2+\lambda\|\mathbf{w}\|_2^2 = \frac{1}{2n}\sum_{i=1}^n (X_i\mathbf{w}+b-\mathbf{y}_i)^2+\lambda\sum_{i=1}^d \mathbf{w}_i^2$$

So we have:

$$\frac{\partial}{\partial \mathbf{w}}(1) = \frac{\partial}{\partial \mathbf{w}}\left(\frac{1}{2n}\sum_{i=1}^{n}(X_i\mathbf{w}+b-\mathbf{y}_i)^2+\lambda\sum_{i=1}^{d}\mathbf{w}_i^2\right)$$
$$= \begin{bmatrix}\frac{\partial}{\partial \mathbf{w}_0} & \frac{\partial}{\partial \mathbf{w}_1} & \cdots\end{bmatrix}^T$$

For each $\mathbf{w}_k$ for k from 1 to d:

$$\begin{aligned}\frac{\partial}{\partial \mathbf{w}_j}(1) &= \frac{\partial}{\partial \mathbf{w}_j}\left(\frac{1}{2n}\sum_{i=1}^{n}(X_i\mathbf{w}+b-\mathbf{y}_i)^2+\lambda\sum_{i=1}^{d}\mathbf{w}_i^2\right)\\ &= \frac{1}{2n}\sum_{i=1}^{n}\left(\frac{\partial}{\partial \mathbf{w}_j}(\sum_{k=1}^{d}X_{ik}\mathbf{w}_k+b-\mathbf{y}_i)^2\right)+\lambda\frac{\partial}{\partial \mathbf{w}_j}\left(\sum_{i=1}^{d}\mathbf{w}_i^2\right)\\ &= \frac{1}{n}\sum_{i=1}^{n}\left(\frac{\partial}{\partial \mathbf{w}_j}\sum_{k=1}^{d}X_{ik}\mathbf{w}_k\right)\left(\sum_{k=1}^{d}X_{ik}\mathbf{w}_k+b-\mathbf{y}_i\right)+2\lambda\mathbf{w}_j\\ &= \frac{1}{n}\sum_{i=1}^{n}X_{ij}\left(\sum_{k=1}^{d}X_{ik}\mathbf{w}_k+b-\mathbf{y}_i\right)+2\lambda\mathbf{w}_j\\ &= \frac{1}{n}\sum_{k=1}^{d}\sum_{i=1}^{n}X_{ij}X_{ik}\mathbf{w}_k+\sum_{i=1}^{n}X_{ij}(b-\mathbf{y}_i)+2\lambda\mathbf{w}_j\\ &= \frac{1}{n}X_j^TX\mathbf{w}+X_j^T(b\mathbf{1}_n-y)+2\lambda\mathbf{w}_j\end{aligned}$$

Therefore:

$$\frac{\partial}{\partial \mathbf{w}} = \frac{1}{n}X^\top(X\mathbf{w}+b\mathbf{1}-\mathbf{y})+2\lambda\mathbf{w}$$

Similarly:

$$\begin{aligned}\frac{\partial}{\partial b} &= \frac{\partial}{\partial b}\left(\frac{1}{2n}\|X\mathbf{w}+b\mathbf{1}_n-\mathbf{y}\|_2^2+\lambda\|\mathbf{w}\|_2^2\right)\\ &. = 2\frac{1}{2n}\mathbf{1}_n^T(X\mathbf{w}+\mathbf{1}_nb-\mathbf{y})\\ &= \frac{1}{n}\mathbf{1}^\top(X\mathbf{w}+b\mathbf{1}-\mathbf{y})\end{aligned}$$

Because we can simple substitute $\mathbf{1}$ to X above as these two equations have the same form, and do not depend on the matrix dimensions.

3. (2 pts) Implement ridge regression using the closed form solution for linear regression as derived in lecture. You may find the function `numpy.linalg.solve` useful.

Ans: From question 1 we know that:

$$\frac{1}{2n}\left\|\begin{bmatrix}X & \mathbf{1}_n\\ \sqrt{2\lambda n}I_d & \mathbf{0}_d\end{bmatrix}\begin{bmatrix}\mathbf{w}\\ b\end{bmatrix}-\begin{bmatrix}\mathbf{y}\\ \mathbf{0}_d\end{bmatrix}\right\|_2^2 = \frac{1}{2n}\|X\mathbf{w}+b\mathbf{1}_n-\mathbf{y}\|_2^2+\lambda\|\mathbf{w}\|_2^2$$

So their gradients are also equal. By inspenction, we found that the left side has the form of $\|A'w'-z'\|_2^2$ with:

$$A' = \begin{bmatrix}X & \mathbf{1}_n\\ \sqrt{2\lambda n}I_d & \mathbf{0}_d\end{bmatrix}, w' = \begin{bmatrix}\mathbf{w}\\ b\end{bmatrix}, z' = \begin{bmatrix}\mathbf{y}\\ \mathbf{0}_d\end{bmatrix}$$

By the formula we introduce in lecture, we can find that the gradient is:

$$\frac{1}{2n}(2A'^TA'w'-2A'^Tz')$$

Setting the above gradient to zero, we get the equation:

$$A'^TA'w' = A'^Tz'$$

which is equivalent to:

$$\begin{bmatrix} X & \mathbf{1}_n \\ \sqrt{2\lambda n} I_d & \mathbf{0}_d \end{bmatrix}^T \begin{bmatrix} X & \mathbf{1}_n \\ \sqrt{2\lambda n} I_d & \mathbf{0}_d \end{bmatrix} \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix} = \begin{bmatrix} X & \mathbf{1}_n \\ \sqrt{2\lambda n} I_d & \mathbf{0}_d \end{bmatrix}^T \begin{bmatrix} \mathbf{y} \\ \mathbf{0}_d \end{bmatrix}$$

4. (2 pts) Implement the gradient descent algorithm for solving ridge regression. The following incomplete pseudo-code may of help. Your training loss should monotonically decrease during iteration; if not try to tune your step size η, e.g. make it smaller.

Algorithm 2: Gradient descent for ridge regression.

Input: $X \in \mathbb{R}^{n \times d}$, $\mathbf{y} \in \mathbb{R}^n$, $\mathbf{w}_0 = \mathbf{0}_d$, $b_0 = 0$, max_pass $\in \mathbb{N}$, $\eta > 0$, tol > 0
Output: $\mathbf{w}, b$

```
for t = 1, 2, ..., max_pass do
    w_t ←
    b_t ←
    if ||w_t − w_{t−1}|| ≤ tol then                    // can use other stopping criteria
        break
w ← w_t,  b ← b_t
```

Ans: Notice that the multiplications below are mathmatical. They are not element-wise.

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \left(\tfrac{1}{n} X^\top (X\mathbf{w}_{t-1} + b_{t-1}\mathbf{1}_n - \mathbf{y}) + 2\lambda \mathbf{w}_{t-1}\right) * \eta$$

$$b_t = b_{t-1} - \left(\tfrac{1}{n} \mathbf{1}_n^\top (X\mathbf{w}_{t-1} + b_{t-1}\mathbf{1}_n - \mathbf{y})\right) * \eta$$

5. (3 pts) Test your two implementations on the Boston housing dataset (to predict the median house price, i.e., y). Use the train and test splits provided on course website. Try $\lambda \in \{0, 10\}$ and report your training error, training loss and test error for each. You may have trouble getting gradient descent to work if you don't standardize your data (you may not use sklearn to standardize your data, do it yourself). Note that you would have to standardize both your training and test features. (Try gradient descent both without and with standardizing the data and see how it differs.) For the gradient descent algorithm, plot the training loss over iterations. Compare the running time of the two approaches, which is faster? Overall, which approach do you think is better? Explain why.

Ans:
The graph is attached below. And the training error and test error are listed as follows:

methods	train error	training loss	test error
lambda0, closed form	4.847149319454688	4.847149319454688	185.11147877100467
lambda10, closed form	38.728489031407676	38.815542966133066	57.10971556808989
lambda0, gradient descent	36.327071579707585	See Graph Below	47.71665235364033
lambda10, gradient descent	44.607081689993855	See Graph Below	39.79430484819916

The running time of closed form is 0ms, while the running time for the gradient descent algorithm is about 0.0625 ms, or 62500000 ns, both are measured in CPU time. So the closed form is definitely faster. Overall in this case, I think I would prefer the closed form solution. Firstly, it runs much faster. Secondly, the result generated by the two methods should be the same, since theoretically they will give the same solution vector (both are looking for the minimum in convex functions). The acutal test error are also close However, the closed form only works here because this is linear regression and we are able to derive it. If we cannot even derive the form, then gradient descent will be the only option.

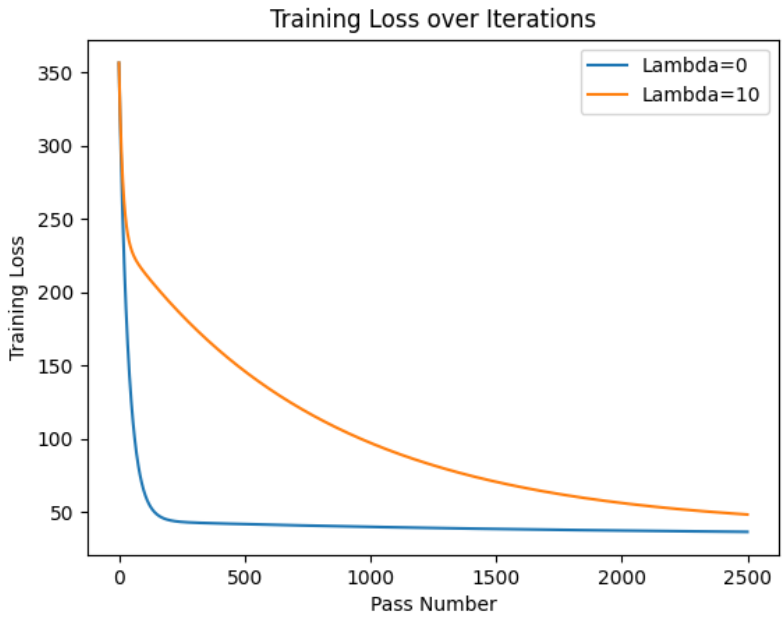

Figure 2: Training Loss over Iterations

Exercise 3: Playing with Regression (3 pts)

You may use the Python package scikit-learn for this exercise (and only for this exercise).

Train (unregularized) linear regression, ridge regression, and lasso on the mystery datasets A, B, and C on the course website (using X_train and Y_train for each dataset). For ridge regression and lasso, use k-fold cross validation to determine appropriate choices of the hyperparameters (you may NOT use functions like RidgeCV and LassoCV, you must implement it yourself). Report the average mean squared error on the test set for each method, as well as the selected regularization parameters. Which approach performs best in each case? For each dataset (A, B, C), do the following: Create a histogram where the x-axis is divided into bins corresponding to the values of coordinates of the parameter vector, and the y-axis is the number of coordinates of the parameter vector which fall into each bin. Plot the three parameter vectors (one for unregularized, one for ridge, one for lasso) on the same histogram, so they're all visible at once (change the opacity setting for the bars if necessary). [There should be three total histograms, each with three parameter vectors overlayed.]

Ans:

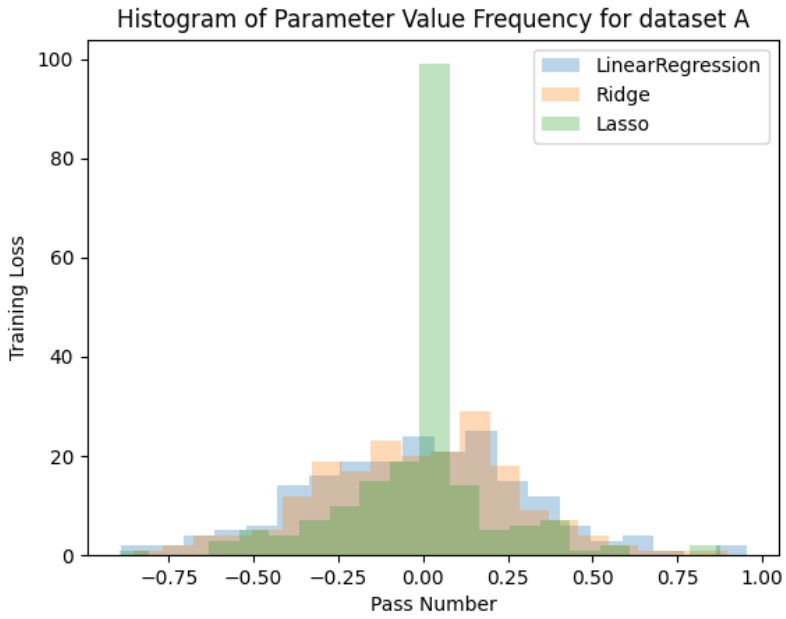

Figure 3: Histogram of Parameter Value Frequency for dataset A

For dataset A:
Model: LinearRegression gives MSE = 3.247400173556322
Model: Ridge gives MSE = 2.77803413209245 with lambda = 10
Model: Lasso gives MSE = 6.1844343662602865 with lambda = 0.1
It seems like the best model is the ridge regression in this case since it gives the lowest MSE.

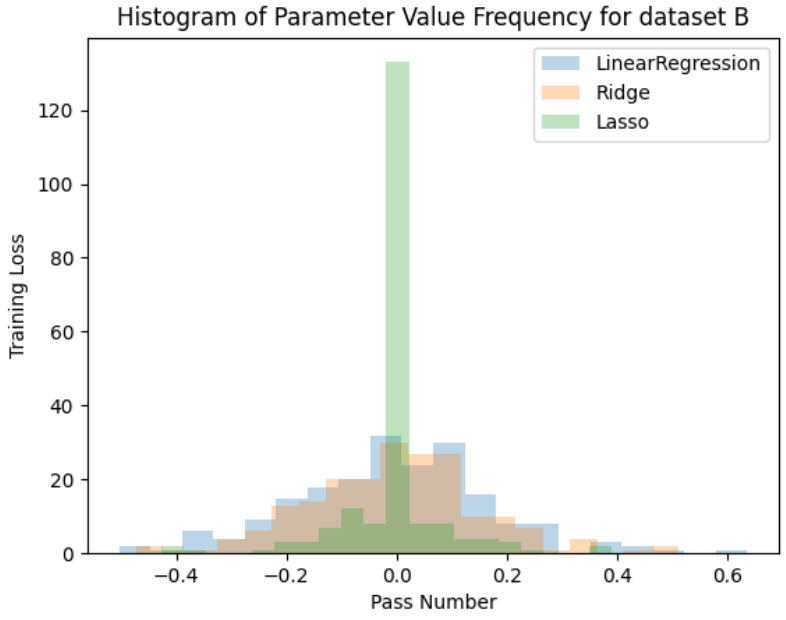

Figure 4: Histogram of Parameter Value Frequency for dataset B

For dataset B:

Model: LinearRegression gives mse = 2.7426823746517033
Model: Ridge gives mse = 2.059713219854696 with lambda = 10
Model: Lasso gives mse = 2.610022700635093 with lambda = 0.1
It seems like the best model is the ridge regression in this case since it gives the lowest MSE. But the difference among them are not significant

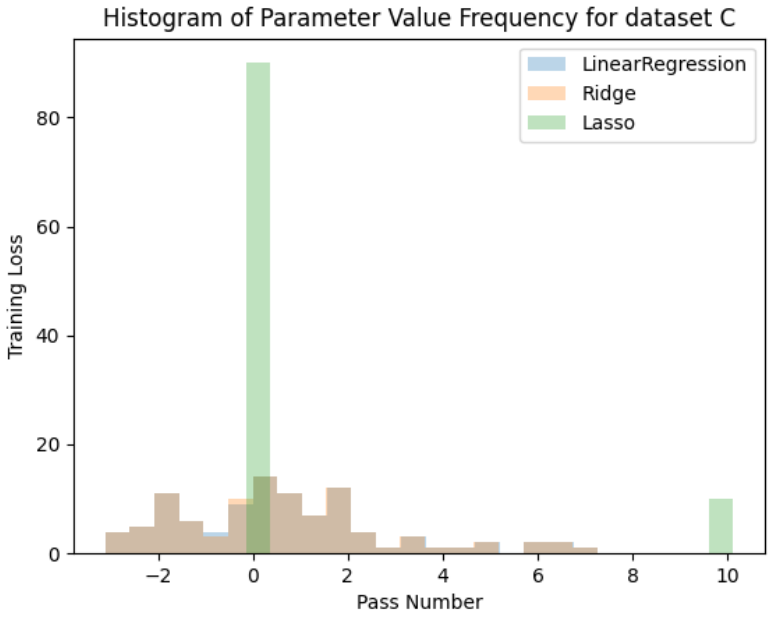

Figure 5: Histogram of Parameter Value Frequency for dataset C

For dataset C:
Model: LinearRegression gives mse = 506.1084477369717
Model: Ridge gives mse = 506.24141706882205 with lambda = 0.1
Model: Lasso gives mse = 1.2485953471555624 with lambda = 0.1
It seems like the best model is the lasso regression in this case since it gives the lowest MSE.

Exercise 4: Nearest Neighbour Regression (7 pts)

1. (3 pts) Implement k-nearest neighbour regression, for a dataset of n X, y pairs where $X \in \mathbb{R}^d$ and $y \in \mathbb{R}$. This is similar to k-nearest neighbour classification, but instead of aggregating labels by taking the majority, we average the y values of the k nearest neighbours. Use ℓ_2 distance as the distance metric, that is, the distance between points X_1 and X_2 is $\|X_1 - X_2\|_2$. Ensure that your implementation is $O(nd)$ time for all values of k, and argue that this is the case.

   Ans:

Algorithm 3: The perceptron algorithm.

Input: $X_{train} \in \mathbb{R}^{n \times d}$, $\mathbf{y}_{train} \in \mathbb{R}^n$, $X \in \mathbb{R}^d$, $k \in \mathbb{N}$
Output: $\mathbf{y}_{new}$

```
for t = 1, 2, ..., n do
    dist(t) ← dist(X, X_train[i])
kth = quickSelect(dist, 0, len(dists) − 1, k)
sum ← 0
for i = 1, 2, ..., n do
    if dist(i) ≤ kth then
        sum ← sum + y_train[i]
y_new ← sum/k
```

The code is in file q4.py.
To calculate the distance, which is the L2-norm, it takes $O(d)$, since we need to iterate over the vector and do $O(1)$ operation in each iteration. Then, we do it for each $X_{train}[i]$. So the L2-norm calculation takes $O(nd)$.
Then, we do a quick selection to find the k-th smallest element in the distance. Quick select algorithm will take $O(n)$ in average.
Finally, we loop through the X again, and find out all elements that is less than or equal to the k-th element and add the corresponding y to the sum. This only takes $O(n)$ time. The retuning value of y_{new} only takes $O(1)$.
Overall, this algorithm will take $O(nd) + O(n) + O(n) + O(1) = O(nd)$ in average.

2. (2 pts) For the training sets of the $d = 1$ mystery datasets D and E on the course website, compute a) the (unregularized) least squares linear regression solution and b) the k-nearest neighbour regression solution with each integer k from 1 to 9. For each dataset, on one plot with the x and y axes corresponding to the x and y values, display the least squares solution, the 1-nearest neighbour solution, and the 9-nearest neighbour solution. Be sure to plot these solutions for all points between the minimum and maximum x values, not just for the points in the dataset. On another plot, with k on the x axis and test mean-squared error on the y axis, display the error of the k-NN solution for each value of k. On the same plot, include the test error of the least squares solution as a horizontal line. When does each approach perform better, and why?

   Ans:

Figure 6: Solutions for dataset D

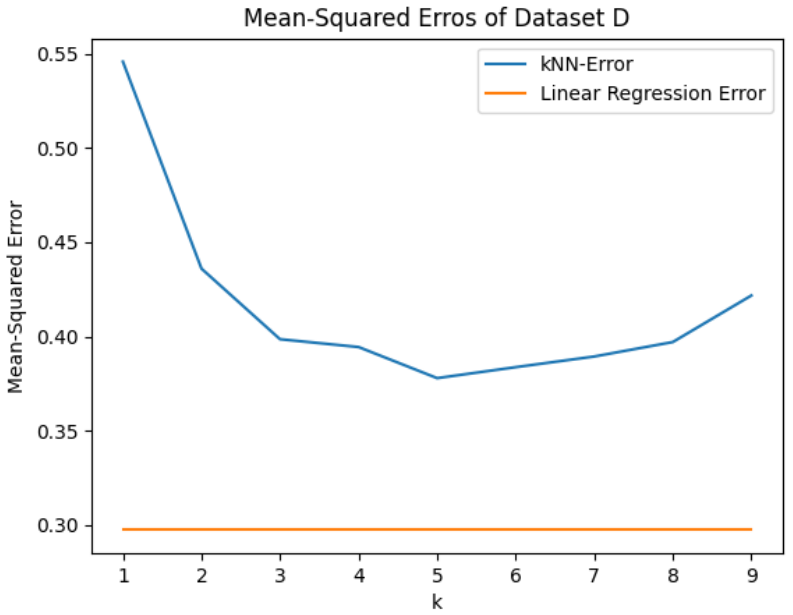

Figure 7: Errors for dataset D

It looks like the linear regression is better for dataset D since it has a smaller mean squared error. If we look at how the datapoints, we can see that the datapoints are spread linearly, which is perfect for linear regression to model it.

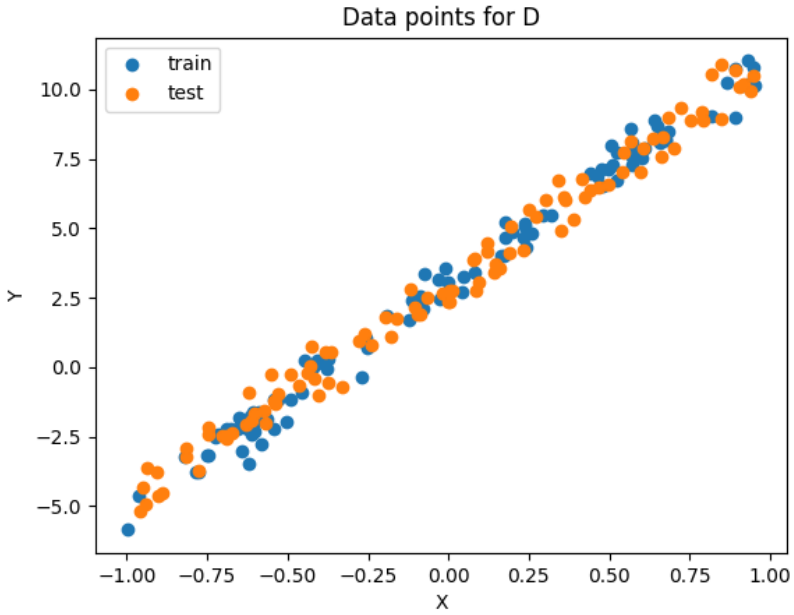

Figure 8: Datapoints for dataset D

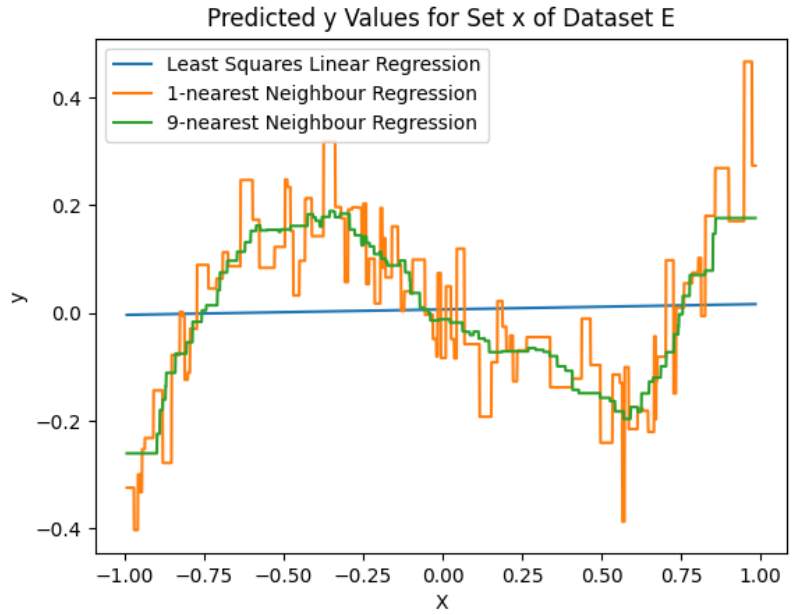

Figure 9: Solutions for dataset E

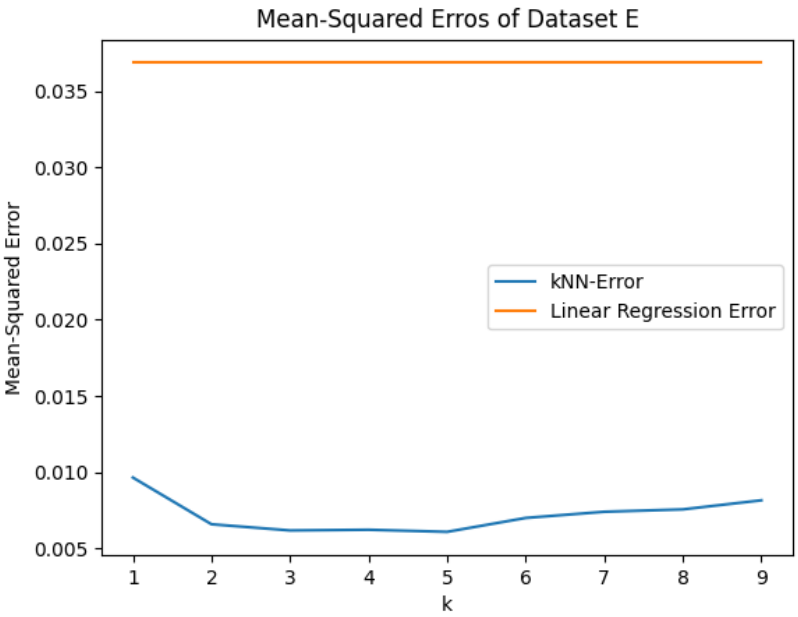

Figure 10: Errors for dataset E

It looks like the k-nn regression is better for dataset E since it has a smaller mean squared error. More specifically, the 5-NN is the best. In this case, our datapoints are pretty much like a sine wave, and linear regression cannot predict it very well in this case. We can see from the solution diagram that the linear regression gives about a straigh line. The result is probably the best ”linearly” fit of the data, but cannot be good for predicting the value, especially that this is not a binarly classifier. Instead, the nearest neighbour algorithm can give result of the ”sine wave” like curve because it depends on the datapoints close to the point. In other words, in this dataset, the data points closer to the actual point are more important.

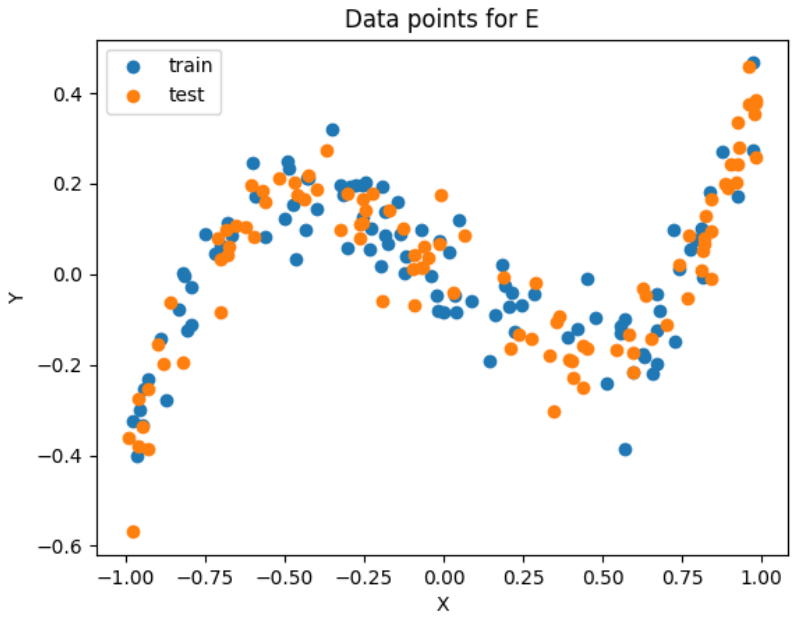

Figure 11: Datapoints for dataset E

3. (2 pts) For the training set of the $d = 20$ mystery dataset F on the course website (Dataset on course website updated May 15), compute a) the (unregularized) least squares linear regression solution and b) the k-nearest neighbour regression solution with each integer k from 1 to 9. Plot, with k on the x axis and test mean-squared error on the y axis, display the error of the k-NN solution for each value of k. On the same plot, include the test error of the least squares solution as a horizontal line. Which approach performs better, and why? Hint: consider inspecting distances between the test points and their k nearest neighbours in the training set.

   Ans:

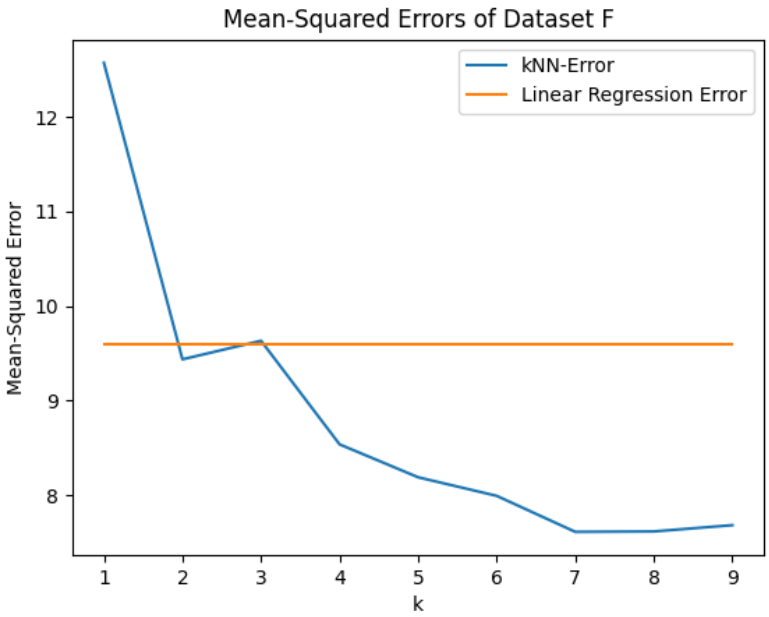

Figure 12: Errors for dataset F

It looks like the k-nn regression is better for dataset F since it has a smaller mean squared error. More specifically, the 7-NN is the best.
A sample data of the distance between the test data point and the train data point is as follows:
(2.739081206438655, 2.7819777208056644, 2.980837647067063, 3.0328653312547393, 3.053496114967776, 3.085910664335926, 3.1002830933648524, 3.106419540372066, 3.17015836472704, 3.176246530582605, 3.241642822244411, 3.262594462096352, 3.3044890637952613, 3.3055354256469833, 3.309942414538081, 3.3630879735638666, 3.3767138978681652)
This is in sorted order. In fact, it looks like our test data point are almost about the same distance from its neighbours. In this example, the test data point are around 3 unit far from its closest 50 neighbours. This means our test data points are about in the "center" of the train data set. In that case, the linear regression model is probably not gonna help. Instead, if we use kNN with more neighbours and we use the average as the aggregate function, then the result might be more correct. If we look at the mean square graph, we can see that when we are using fewer neighbours, the error is huge, and that is likely because the model is "biased". (The test data is in the "center" of other data points, but we only used one to model it). As we use more and more neighbours, we are getting closer and closer to the real value.