FOML Assignment 4

Ahmik Virani, ES22BTECH11001

November 27, 2024

Question 1: VC-Dimension

For the problem statement, x can be 1 when it lies in the interval: $p < x < q$. Let's treat the other case as x is 0. Thus if we have n points, we can have a total of 2^n possible cases. Now to find VC-Dimension, we need to find the maximum number of points that can be shattered by H.
For the given 1D space, if we take 2 points, then we could either have both inside the boundary or p and q or either 1 inside or none and classify them. But when we have 3 points, then we could have the case that points $= \{x, y, z\}$ but $x <= p, p < y < q, z >= q$, in such a case we could not classify. Thus VC-Dimension is 2.

Question 2: Regularizer

Answer: Let us begin with omiting the w_0 term.

$$y = \sum_{i=1}^{N} w_i x_i$$

Now, due to some noise, say the $x'_i = x_i + \epsilon_i$ Thus the error causes the new predicted output:

$$y' = \sum_{i=1}^{N} w_i x_i + \sum_{i=1}^{N} w_i \epsilon_i \tag{1}$$

$$= y + \sum_{i=1}^{N} w_i \epsilon_i \tag{2}$$

We are interested in $\mathbb{E}\left[(y' - t)^2\right]$

$$\mathbb{E}\left[(y' - t)^2\right] = \mathbb{E}\left[\left(y + \sum_{i=1}^{N} w_i \epsilon_i - t\right)^2\right] \tag{3}$$

$$= \mathbb{E}\left[\left((y - t) + \sum_{i=1}^{N} w_i \epsilon_i\right)^2\right] \tag{4}$$

$$= \mathbb{E}\left[(y - t)^2 + 2(y - t)\left(\sum_{i=1}^{N} w_i \epsilon_i\right) + \left(\sum_{i=1}^{N} w_i \epsilon_i\right)^2\right]. \tag{5}$$

$$= \mathbb{E}\left[(y - t)^2\right] + 0 + \mathbb{E}\left[\sum_{i=1}^{N} w_i^2 \epsilon_i^2\right] \tag{6}$$

Note the above reduces because ϵ_i is independent of ϵ_j as given in the question when i does not equal j and clearly y is the term which is not influenced by error, thus $(y - t)$ is independent of ϵ_i.

Also, $\mathbb{E}\left[\epsilon_i^2\right] = \sigma^2$

Multiplying $\frac{N}{2}$ on both sides and taking $\sum_{i=1}^{N}$, the earlier expression further reduces to:

$$\mathbb{E}\left[\frac{1}{2}\sum_{i=1}^{N}(y' - t)^2\right] = E\left[\frac{1}{2}\sum_{i=1}^{N}(y - t)^2\right] + \frac{N}{2}\sigma^2 \sum_{i=1}^{N} w_i^2$$

Comparing with L2 regularization, $\lambda = \sigma^2 \frac{N}{2}$
Minimizing sum of squares error in noise is same as L2 regularization term.
In summary, the presence of Gaussian noise in the input variables leads to an increased expected error in the predictions. By minimizing the sum-of-squares error averaged over noisy data, we get a penalty on the weights similar to L2 regularization. This relationship indicates the importance of regularization in robust model training, ensuring that the model remains stable and generalizes well to unseen data.

Question 3: Hierarchical Clustering

Part(a): Single Link

Table 1: Step 1

	x_1	x_2	x_3	x_4	x_5	x_6
x_1	0					
x_2	0.12	0				
x_3	0.51	0.25	0			
x_4	0.84	0.16	0.14	0		
x_5	0.28	0.77	0.70	0.45	0	
x_6	0.34	0.61	0.93	0.20	0.67	0

Table 2: Step 2

	(x_1, x_2)	x_3	x_4	x_5	x_6
(x_1, x_2)	0				
x_3	0.25	0			
x_4	0.16	0.14	0		
x_5	0.28	0.70	0.45	0	
x_6	0.34	0.93	0.20	0.67	0

Table 3: Step 3

	(x_1, x_2)	(x_3, x_4)	x_5	x_6
(x_1, x_2)	0			
(x_3, x_4)	0.16	0		
x_5	0.28	0.45	0	
x_6	0.34	0.20	0.67	0

Table 4: Step 4

	$((x_1, x_2), (x_3, x_4))$	x_5	x_6
$((x_1, x_2), (x_3, x_4))$	0		
x_5	0.28	0	
x_6	0.20	0.67	0

Finally we get:

Table 5: Step 5

	$(((x_1, x_2), (x_3, x_4))x_6)$	x_5
$(((x_1, x_2), (x_3, x_4))x_6)$	0	
x_5	0.28	0

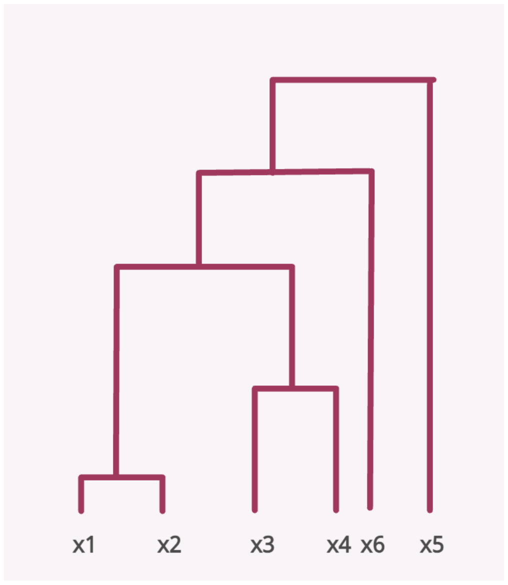


Figure 1: Dendogram for single link

Part B: Complete Link

Table 6: Step 1

	x_1	x_2	x_3	x_4	x_5	x_6
x_1	0					
x_2	0.12	0				
x_3	0.51	0.25	0			
x_4	0.84	0.16	0.14	0		
x_5	0.28	0.77	0.70	0.45	0	
x_6	0.34	0.61	0.93	0.20	0.67	0

Table 7: Step 2

	(x_1, x_2)	x_3	x_4	x_5	x_6
(x_1, x_2)	0				
x_3	0.51	0			
x_4	0.84	0.14	0		
x_5	0.77	0.70	0.45	0	
x_6	0.61	0.93	0.20	0.67	0

Table 8: Step 3

	(x_1, x_2)	(x_3, x_4)	x_5	x_6
(x_1, x_2)	0			
(x_3, x_4)	0.84	0		
x_5	0.77	0.70	0	
x_6	0.61	0.93	0.67	0

Table 9: Step 4

	$((x_1, x_2), x_6)$	(x_3, x_4)	x_5
$((x_1, x_2), x_6)$	0		
(x_3, x_4)	0.93	0	
x_5	0.77	0.70	0

Table 10: Step 5

	$((x_1, x_2), x_6)$	$((x_3, x_4), x_5)$
$((x_1, x_2), x_6)$	0	
$((x_3, x_4), x_5)$	0.93	0

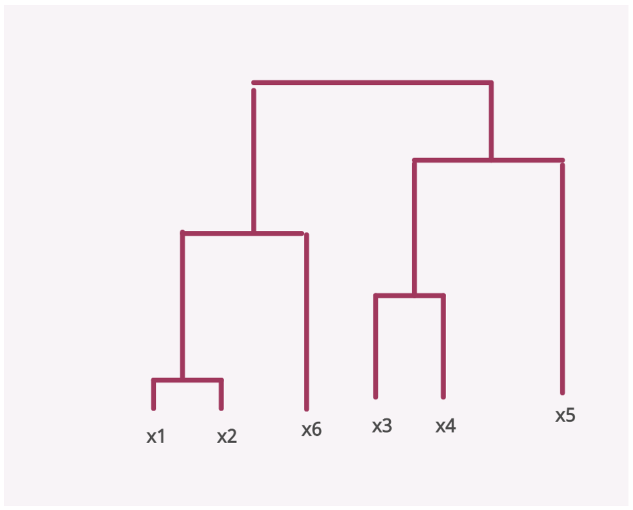


Figure 2: Dendogram for complete link

Part(c):

For this, I tried to get the same graph as part (a) for both of them. Inorder to do so, i replaced the values 0.84 to 0.50 and 0.93 to 0.60 to get the matrix:

Table 11: Step 1

	x_1	x_2	x_3	x_4	x_5	x_6
x_1	0					
x_2	0.12	0				
x_3	0.51	0.25	0			
x_4	0.50	0.16	0.14	0		
x_5	0.28	0.77	0.70	0.45	0	
x_6	0.34	0.61	0.60	0.20	0.67	0

And the dendogram I got was: Which is the same as part(a)

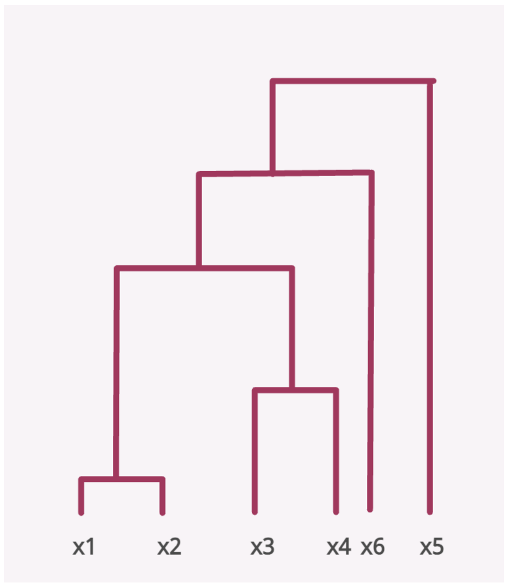


Figure 3: Dendogram for single link

Question 4: Principal Component Analysis

Part(a)

: Given k is uniformly distributed in the range $[1, M]$.
Thus, $E[k] = \frac{M+1}{2}$ and $var(k) = \frac{M^2-1}{12}$.
The covariance matrix as hinted in the question is:

$$C = E[(x - E[x])(x - E[x])^T]$$

In the question, $x = a\delta_k$. Since a is in the k^{th} slot and k is a uniform random variable, thus the expectation of x is the following:

$$\mathbb{E}[x] = \frac{a}{M}1$$

where 1 is the vector full of 1's, and each element will be weighted equally by $\frac{1}{M}$.
Thus, for any one element in $\mathbb{E}[x]$

$$\mathbb{E}[x_i] = \frac{a}{M}$$

The **Covariance** of random variables X_i and X_j, denoted as $\text{Cov}(X_i, X_j)$, is

$$\text{Cov}(X_i, X_j) = \mathbb{E}\big[(X_i - \mathbb{E}[X_i])(X_j - \mathbb{E}[X_j])\big].$$

$\text{Cov}(X_i, X_j)$ is a measure of correlation between X_i and X_j. Another way to look at $\text{Cov}(X_i, X_j)$ is as follows:

$$\begin{aligned}\text{Cov}(X_i, X_j) &= \mathbb{E}\big[(X_i - \mathbb{E}[X_i])(X_j - \mathbb{E}[X_j])\big] \\ &= \mathbb{E}\big[X_iX_j - X_i\mathbb{E}[X_j] - X_j\mathbb{E}[X_i] + \mathbb{E}[X_i]\mathbb{E}[X_j]\big] \\ &= \mathbb{E}[X_iX_j] - \mathbb{E}[X_i]\mathbb{E}[X_j] - \mathbb{E}[X_j]\mathbb{E}[X_i] + \mathbb{E}[\mathbb{E}[X_i]\mathbb{E}[X_j]] \\ &= \mathbb{E}[X_iX_j] - \mathbb{E}[X_i] \cdot \mathbb{E}[X_j]\end{aligned}$$

(Using the Linearity of Expectation)
Credits of the proof above: https://homepage.cs.uiowa.edu/ sriram/5360/fall18/notes/9.10/week4Notes.pd

Now since we got the equation: $C = \mathbb{E}[x_ix_j] - \mathbb{E}[x_i] \cdot \mathbb{E}[x_j]$, we can divide it into two cases, one where $i = j$ and one where $i \neq j$.

$i = j$:

$$\mathbb{E}[x_i.x_i] = \frac{a^2}{M^2}M = \frac{a^2}{M}$$

$$(\mathbb{E}[x_i])^2 = \frac{a^2}{M^2} = \mathbb{E}[x_i].\mathbb{E}[x_j]$$

$$\mathbb{E}[x_i.x_j] = 0$$

This equals zero because at any time only either i^{th} element will have the term a or the j^{th} element will have, so the dot product if $i \neq j$ is zero.

Thus we get

$$C_{ij} = \delta_{i,j} * \frac{a^2}{M} - \frac{a^2}{M^2}$$

Thus from the question:

$$\lambda = -\frac{a^2}{M^2}, \mu = \frac{a^2}{M}$$

Part(b)

From above, we can write $C = -\frac{a^2}{M^2} 1_{MxM} + \frac{a^2}{M} I_{MxM}$, where 1_{MxM} is an MxM matrix of all ones, and I_{MxM} is an identity matrix of dimensions MxM.
To find the eigenvalues and eigenvectors, we need to find $C.v = \lambda.v$, where v is the eigenvector and λ is the eigenvalue.

Let us try with $v = (1, ..., 1)^T$. This basically means that $J.v = M$ and $I.v = v$. Thus combining this:

$$C.v = -\frac{a^2}{M^2} Mv + \frac{a^2}{M} v$$

This just boils down to:

$$C.v = 0$$

Thus, $v = (1, ..., 1)^T$ is an eigenvector and the corresponding eigenvalue equals 0.

Next to find the other eigenvector, it must be orthogonal to $v = (1, ..., 1)^T$. Let w be one of the eigenvectors that are orthogonal, this means that $w \cdot v = 0$.
Now, $J \cdot w = 0$, this is because J is a matrix full of ones, and because $v \cdot w = 0$, and v also is a vector full of ones, $J \cdot w$ should also be a zero matrix just of a higher dimension equal to $Mx1$. Next $I \cdot w = w$. This means:

$$C \cdot w = \frac{a^2}{M} w + 0 = \frac{a^2}{M} w$$

This basically means that all orthogonal vectors have the same eigenvalue equal to $\frac{a^2}{M}$.

Part C

No PCA is not a good method to select features here. In our case the features are not correlated because except for the addition in the diagonal part of the covariance matrix, the rest of the entire matrix is same. Also we can see that most of the eigenvectors are also same. PCA will not be able to select any meaningful features becuase all features have the same variance. Thus PCA is not useful here.

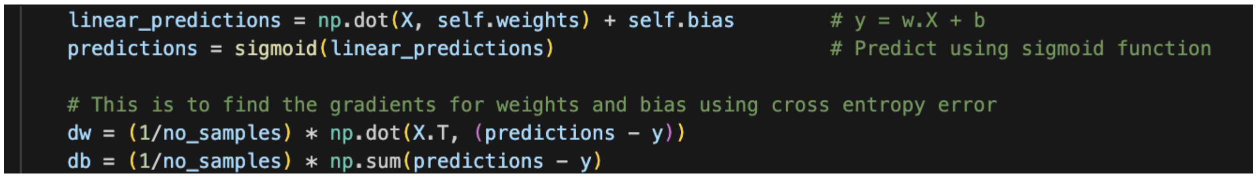
Question 5: Logistic Regression

Part (a): The implementation

Logistic regression is very similar to linear regression. But instead of specific value, we work with probabilities. We put this inside a sigmoid function and get probabilities from 0 and 1.

$$h = \frac{1}{1 + \exp(-wx - b)}$$

Here instead of using the mean square error, we use cross entropy. And also we use gradient descent, for that we calculate gradient of error function in terms of weights and bias.



```
    linear_predictions = np.dot(X, self.weights) + self.bias        # y = w.X + b
    predictions = sigmoid(linear_predictions)                       # Predict using sigmoid function

    # This is to find the gradients for weights and bias using cross entropy error
    dw = (1/no_samples) * np.dot(X.T, (predictions - y))
    db = (1/no_samples) * np.sum(predictions - y)
```

Figure 4: Image showing python code

Part(b)

(i): The logistic model is:

$$\frac{1}{1 + e^{1-1.5x_1-0.5x_2}}$$

The cross entropy error function is:

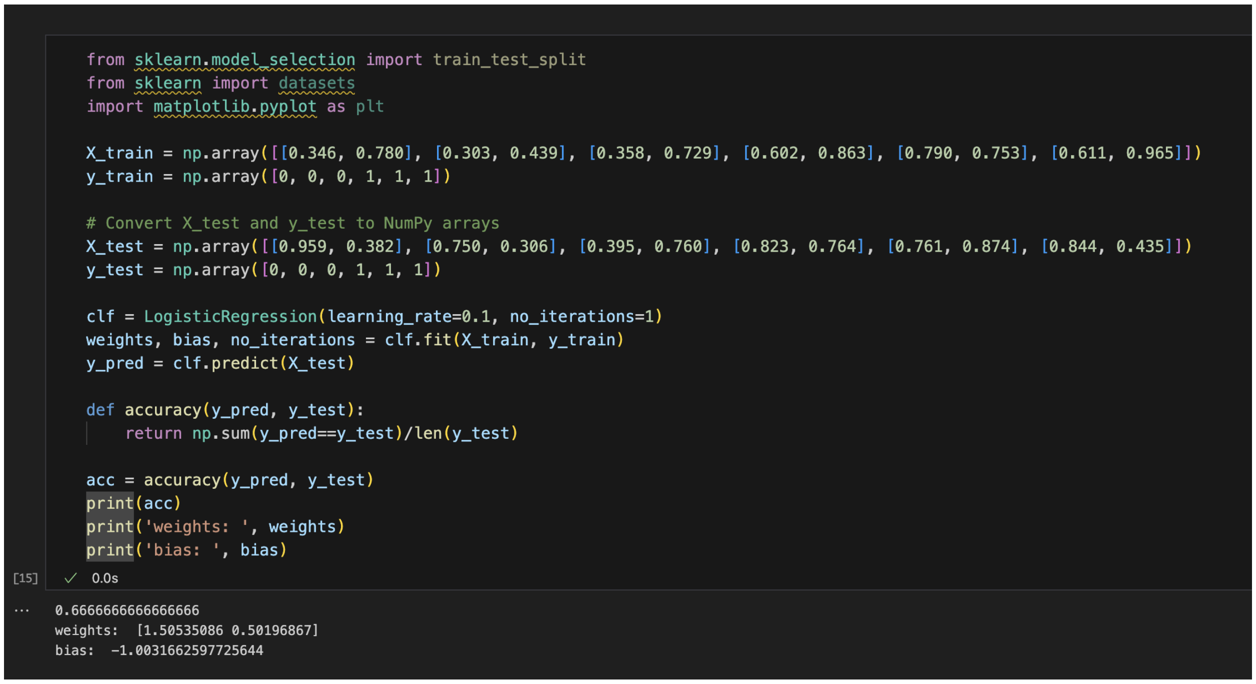
$$\frac{1}{N}\sum_{i=1}^{N}[y\log(h) + (1-y)\log(1-h)]$$

Here, $y = truevalue$, $h = predictedvaluegiveninpart(a)$.

$$h = \frac{1}{1 + \exp(-wx - b)}$$

part(ii): Here the value I got after 1 iteration were:

$$\theta_0 = -1.0531662597725642, \theta_1 = 1.50535086, \theta_2 = 0.50196867$$



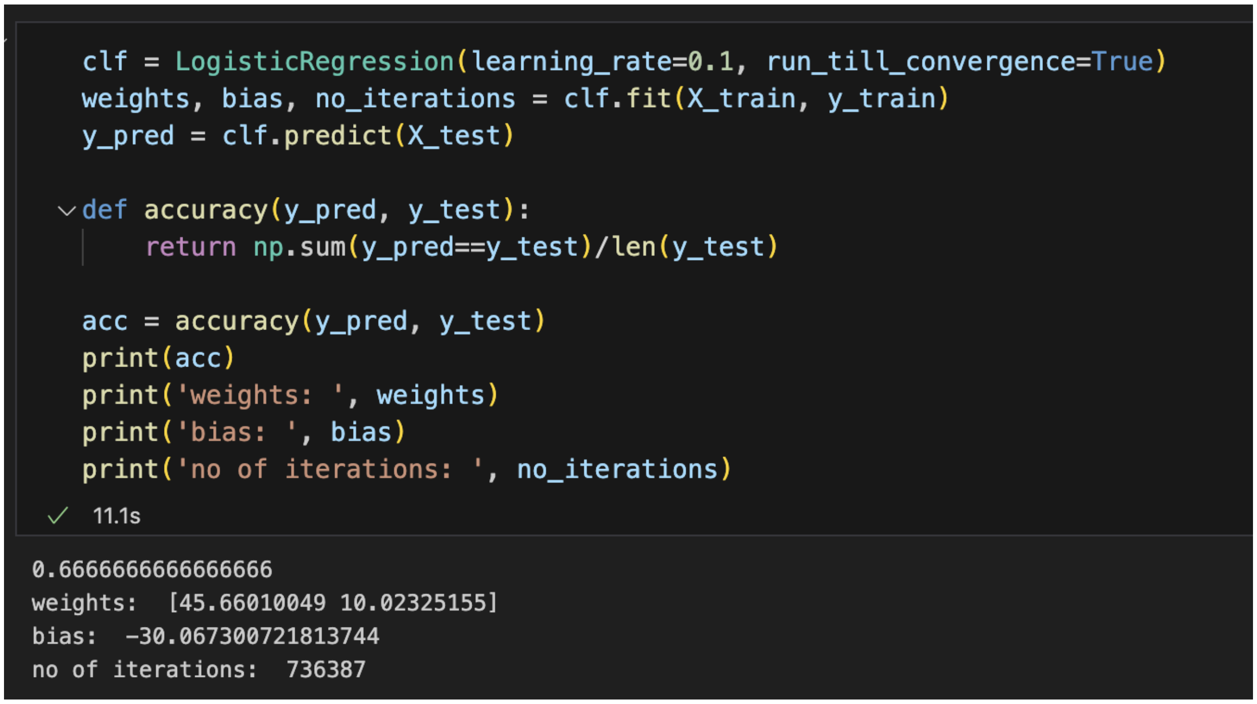
```
from sklearn.model_selection import train_test_split
from sklearn import datasets
import matplotlib.pyplot as plt

X_train = np.array([[0.346, 0.780], [0.303, 0.439], [0.358, 0.729], [0.602, 0.863], [0.790, 0.753], [0.611, 0.965]])
y_train = np.array([0, 0, 0, 1, 1, 1])

# Convert X_test and y_test to NumPy arrays
X_test = np.array([[0.959, 0.382], [0.750, 0.306], [0.395, 0.760], [0.823, 0.764], [0.761, 0.874], [0.844, 0.435]])
y_test = np.array([0, 0, 0, 1, 1, 1])

clf = LogisticRegression(learning_rate=0.1, no_iterations=1)
weights, bias, no_iterations = clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

def accuracy(y_pred, y_test):
    return np.sum(y_pred==y_test)/len(y_test)

acc = accuracy(y_pred, y_test)
print(acc)
print('weights: ', weights)
print('bias: ', bias)
```

```
[15]  ✓ 0.0s
```

```
0.6666666666666666
weights:  [1.50535086 0.50196867]
bias:  -1.0031662597725644
```

Figure 5: Image showing output for part 2

part(iii):



```
clf = LogisticRegression(learning_rate=0.1, run_till_convergence=True)
weights, bias, no_iterations = clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

def accuracy(y_pred, y_test):
    return np.sum(y_pred==y_test)/len(y_test)

acc = accuracy(y_pred, y_test)
print(acc)
print('weights: ', weights)
print('bias: ', bias)
print('no of iterations: ', no_iterations)
```

```
✓ 11.1s
```

```
0.6666666666666666
weights:  [45.66010049 10.02325155]
bias:  -30.067300721813744
no of iterations:  736387
```

Figure 6: Image showing output for part 3

We can see that it took 736387 steps to converge. And the results I got were:

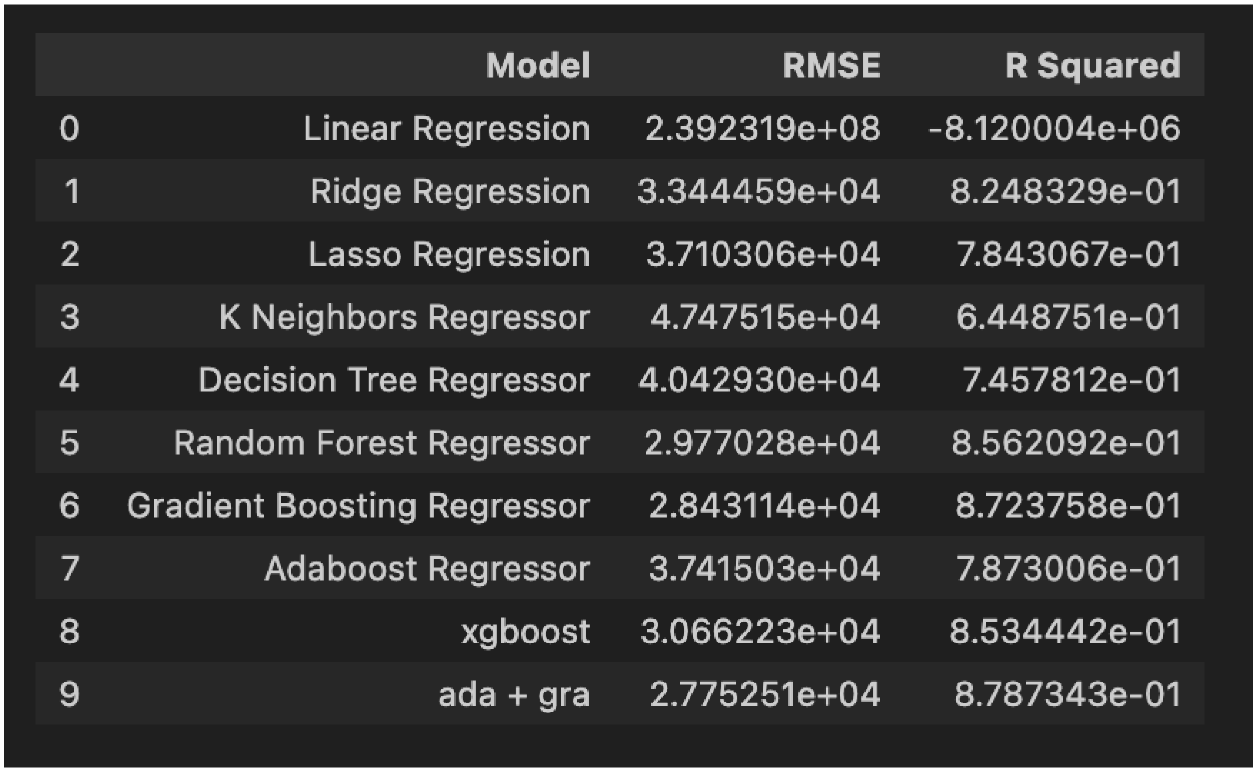
- Precision: 0.6
- Recall: 1.0
- Accuracy: 0.6666666666666666

Credits: https://www.youtube.com/watch?v=YYEJ_GUguHw, Note that I have followed this youtube tutorial for reference but whereever I felt necessary I have made changes as per the question.

Question 6: Kaggle (House Prices)

Before testing the models, I began with the preprocessing steps. I had identified that there were several categorical variables, some of which were ordinal and other could do with one hot encoding. Then I also ensured that any of the columns with more than 50% null values I remove. And the remaining null values I assigned the median.
The multiple models I have tried are:

- Linear Regression
- Ridge Regression
- Lasso Regression
- K Neighbors Regressor
- Decision Tree Regressor
- Random Forest Regressor
- Gradient Boosting Regressor
- Adaboost Regressor
- Adaboost Regression with base estimator as Gradient Boosting Regressor
- XGBoost Regressor



	Model	RMSE	R Squared
0	Linear Regression	2.392319e+08	-8.120004e+06
1	Ridge Regression	3.344459e+04	8.248329e-01
2	Lasso Regression	3.710306e+04	7.843067e-01
3	K Neighbors Regressor	4.747515e+04	6.448751e-01
4	Decision Tree Regressor	4.042930e+04	7.457812e-01
5	Random Forest Regressor	2.977028e+04	8.562092e-01
6	Gradient Boosting Regressor	2.843114e+04	8.723758e-01
7	Adaboost Regressor	3.741503e+04	7.873006e-01
8	xgboost	3.066223e+04	8.534442e-01
9	ada + gra	2.775251e+04	8.787343e-01

Figure 7: Image RMSE scores on training

Above is a photo of the scores I got during training and based on that I have made my models. Note that we can see mostly ensemble methods as the models which are giving the best rmse values. The reason why ensemble models work well is because they combine multiple learners instead of just one. Now, I saw that gradient boosting is giving the best, however from my hackathon experience I learnt that if we use adaboost along with another algorithm as base learner it gives much high accuracy, that is why I ended up using adaboost with gradient boosting and it ended up giving much better result. To do hyperparameter tuning I used RandomizedSearchCV.

My best score is 0.13134 given by model Adaboost with base learner of gradient boosting.
My second best score is 0.13275 given by xgboost.
Note that I ran with random forest as well but it was not giving desired results.
For this question I have taken reference from the official kaggle site on https://www.kaggle.com/code/swatisi
of-8-regression-models-to-predict-strength to learn more about regression models.