Introduction to Machine Learning (CSCI-UA.473): Homework 2

Instructor: Lerrel Pinto

September 20, 2022

1 Submission Instructions

You must typeset the answers using IATEX and compile them into a single PDF file. Name the pdf file as ⟨ Your-NetID ⟩ _hw2.pdf and the notebook containing the coding portion as ⟨Your-NetID ⟩ _hw2.ipynb. The PDF file should contain solutions to both the theory portion and the coding portion. Submit the files through the following Google Form - https://forms.gle/Rf63VnEaMoLcWr7p7 The due date is October 4, 2022, 11:59 PM. You may discuss the questions with each other but each student must provide their own answer to each question.

2 Questions

3 Question 1: Empirical vs. Expected Cost (10 points)

We approximate the true cost function with the empirical cost function defined by:

$$\mathbb{E}_x[E(g(x), f(x))] = \frac{1}{N} \sum_{i=1}^{N} E\left(g\left(x^i\right), y^i\right),$$

where N is the number of training samples, f is the unknown function, g is the learnable function, E is the cost function, y^i is the label associated with the input x^i. In Eq. 1] the left-hand side of the equation represents the expected value of the cost between $g(x)$ and $f(x)$ for every x in the dataset, and the right-hand side approximates this expectation by computing a mean over the errors assigning equal weight to each sample. In the above equation is it okay to give an equal weight to the cost associated with each training example? Given that we established that not every data x is equally likely, is taking the sum of all per-example costs and dividing by N reasonable? Should we weigh each per-example cost differently, depending on how likely each x is? Justify your answer.

It is ok to give an equal weight to the cost associated with each training example if every x in the dataset are equally likely because the output will not have any bias based on all equally likely dataset.

Simply taking the sum of all per-example costs and dividing by N is not reasonable if every x in the dataset is not equally likely. It will cause the bias (which is, the output may tends to the value of x that have the large probability). The x with the more frequency need to be weigh less and vice versa.

Generally speaking, we should weigh each per-example cost differently depending on how likely each x is. More precisely, we need to get the expectation of all the $p_i of x_i$, and then get the probability,q_i of $\frac{p_x i}{E[P_x]}$ if the probability $q_i > 1$, this means that the x weigh more than the mean of the p_i. Therefore, we use the cost times $\frac{1}{q_i}$ to balance the x_i with different p_i, and make them similar to the x_i that euqally likely.

4 Question 2: Simple Linear Regression Model (10 points)

Consider the following model: $Y_i = 5 + 0.5X_i + \epsilon_i, \epsilon_i \overset{iid}{\sim} N(0, 1)$

1. What is $\mathbb{E}[Y \mid X = 0], \mathbb{E}[Y \mid X = -2]$ and $\mathrm{Var}[Y \mid X]$?

   Ans.

   when $X = 1$, $Y = 5 + \epsilon_i \Rightarrow \mathbb{E}[Y \mid X = 0] = \mathbb{E}[5 + \epsilon_i] = 5 + \mathbb{E}[\epsilon_i] = 5$
   (By Linearity of expectation)

   when $X = -2$, $Y = 4 + \epsilon_i \Rightarrow \mathbb{E}[Y \mid X = -2] = \mathbb{E}[4 + \epsilon_i] = 4 + \mathbb{E}[\epsilon_i] = 4$
   (By Linearity of expectation)

   $\mathrm{Var}[Y \mid X] = \mathrm{Var}[5 + 0.5X_i \mid X] = \mathrm{Var}[5 + 0.5X] = \mathrm{Var}[\epsilon_i]$
   Since $\epsilon_i \overset{iid}{\sim} N(0, 1)$, $\mathrm{Var}[\epsilon_i] = 1$

2. What is the probability of $Y > 5$, given $X = 2$?

   $P(Y > 5 \mid X = 2) = P((6 + \epsilon_i) > 5) = P(\epsilon_i > -1)$
   Since ϵ_i is a independent identically distributed random variable, there exists:

$$Z_n = \sqrt{n}\frac{\overline{X_n} - \mu}{\sigma} = \frac{\overline{X_n} - E[\overline{X_n}]}{\sqrt{Var(\overline{X_n})}}$$

Therefore, $Z_n = \epsilon_i$ because the mean of ϵ is 0 and variance is 1 $\implies P(\epsilon > -1) = P(Z_n > -1) = 1 - 0.159 = 0.841$

3. If X has a mean of zero and variance of 10 , what are $\mathbb{E}[Y]$ and $\operatorname{Var}[Y]$?

   The mean of X is the expectation of X, which is 0, and the mean of ϵ is also 0.

   Therefore, $E[Y] = 5 + 0.5 \cdot 0 + 0 = 5$
   $Var[Y] = 0.5^2 Var[X] + Var[\epsilon] = 3.5$

4. What is $\operatorname{Cov}(X, Y)$?

   $Cov(X, Y) = E[XY] - E[X]E[Y]$
   $= E[5X + 0.5X^2 + \epsilon X] - E[X]E[5 + 0.5X + \epsilon]$
   $= 5E[X] + 0.5E[X^2] + E[\epsilon]E[X] - 5E[X] + 0.5E[X]^2 - E[\epsilon]$
   $= 0.5(E[X^2] + E[X]^2)$
   $= 0.5Var(X)$

5 Question3: Least Squares Regression (10 points)

Consider the linear regression model:

$$y = \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_k x_k + \epsilon, \epsilon$$

where y is a dependent variable, x_i corresponds to independent variables and θ_i corresponds to the parameters to be estimated. While approximating a best-fit regression line, though the line is a pretty good fit for the dataset as a whole, there may be an error between the predicted value $\hat{y}$ and true value y for every data point $\mathbf{x} = [x_1, x_2, \ldots, x_k]$ in the dataset. This error is captured by $\epsilon \sim N\left(0, \sigma^2\right)$, where for each data point with features x_i, the label $\hat{y}$ is drawn from a Gaussian with mean $\theta^{\mathbf{T}}\mathbf{x}$ and variance σ^2. Given a set of N observations, provide the closed form solution for an ordinary least squares estimate $\hat{\theta}$ for the model parameters θ.

For the ordinary least squares method, the assumption is that $\operatorname{Var}\left(\epsilon_i \mid X_i\right) = \sigma^2$, where σ is a constant value. However, when $\operatorname{Var}\left(\epsilon_i \mid X_i\right) = f\left(X_i\right) \neq \sigma^2$, the error term for each observation X_i has a weight W_i corresponding to it. This is called Weighted Least Squares Regression. In this scenario, provide a closed form weighted least squares estimate $\hat{\theta}$ for the model parameters θ.

1. Ordinary least squares method:

minimize the loss $\|X^\intercal\theta - y\|^2$

$=< (X^\intercal\theta - Y), (X^\intercal\theta - Y) >$

$= (X^\intercal\theta - Y)^\intercal(X^\intercal\theta - Y)$

$= (X^\intercal\theta - Y^\intercal)(X^\intercal\theta - Y)$

$L = \theta^\intercal XX^\intercal\theta - \theta^\intercal XY - Y^\intercal X^\intercal\theta + Y^\intercal Y$

$\frac{\partial L}{\partial \theta} = 0$

When the derivative equals 0, θ reaches the minimum

$\Longrightarrow \frac{\partial}{\partial \theta}(\theta^\intercal XX^\intercal\theta - \theta^\intercal XY - Y^\intercal X^\intercal\theta + Y^\intercal Y) = 0$

$\Longrightarrow 2XX^\intercal\theta - 2XY = 0$

$\Longrightarrow XX^\intercal\theta = XY$

$\Longrightarrow \theta = (XX^\intercal)^{-1}XY$

2. Weighted least squares method: minimize the loss $\|W^{1/2}(X^T\theta - Y)\|^2$

$=< (W^{1/2}(X^T\theta - Y)), (W1/2(X^T\theta - Y)) >$

$= (X^T\theta - Y)^T W(X^T\theta - Y)$

$= (X^T\theta - Y^T)W(X^T\theta - Y)$

$L = \theta^T XWX^T\theta - \theta^T XWY - Y^T WX^T\theta + Y^T WY$

When the derivative equals 0, θ reaches the minimum

$\frac{\partial L}{\partial \theta}(\theta^T XWX^T\theta - \theta^T XWY - Y^T WX^T\theta + Y^T WY) = 0$

$\Longrightarrow 2XWX^T\theta - 2XWY = 0$

$\Longrightarrow XWX^T\theta = XWY$

$\Longrightarrow \theta = (XWX^T)^{-1}XWY$

6 Question 4: Linear vs Logistic Regression (5 points)

Explain. with equations, the difference between linear and logistic regression.

The equation of linear regression:$y = X^\intercal w$
For linear regression, the input data is $X \in \mathbb{R}^{d\times n}, Y \in \mathbb{R}^n$, where ($\overrightarrow{x} \in \mathbb{R}^d, y \in \mathbb{R}^1$)corresponds to a data point.

The equation of logistic regression:$p(y) = \frac{e^{w^\intercal x}}{1+e^{w^\intercal x}}$

From the equation above, we can find that the linear regression is a linear approach modeling that handles the relationship of a variable and another one, but the logic regression does not handle any correlation between variables; the input of the logistic regression is an event and the output is probability. Linear Regression is used by regression problems, while Logistic regression is used by classification problems.

Homework 2: Linear Regression

The is the coding potion of Homework 2. The homework is aimed at testing the ability to deal with a real-world dataset and use linear regression on it.

In [125]:
```
import numpy as np
import pandas as pd

# Plotting libraries
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
```

Load Dataset

Loading the California Housing dataset using sklearn.

In [170]:
```python
# Load dataset
from sklearn.datasets import fetch_california_housing
housing = fetch_california_housing()
#this print is used for check
print(housing.DESCR)
```

```
.. _california_housing_dataset:

California Housing dataset
--------------------------

**Data Set Characteristics:**

    :Number of Instances: 20640

    :Number of Attributes: 8 numeric, predictive attributes and the targe
t

    :Attribute Information:
        - MedInc        median income in block
        - HouseAge      median house age in block
        - AveRooms      average number of rooms
        - AveBedrms     average number of bedrooms
        - Population    block population
        - AveOccup      average house occupancy
        - Latitude      house block latitude
        - Longitude     house block longitude

    :Missing Attribute Values: None

This dataset was obtained from the StatLib repository.
http://lib.stat.cmu.edu/datasets/ (http://lib.stat.cmu.edu/datasets/)

The target variable is the median house value for California districts.

This dataset was derived from the 1990 U.S. census, using one row per cen
sus
block group. A block group is the smallest geographical unit for which th
e U.S.
Census Bureau publishes sample data (a block group typically has a popula
tion
of 600 to 3,000 people).

It can be downloaded/loaded using the
:func:`sklearn.datasets.fetch_california_housing` function.

.. topic:: References

    - Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions,
      Statistics and Probability Letters, 33 (1997) 291-297
```

Part 1 : Analyse the dataset

In [127]:

```
# Put the dataset along with the target variable in a pandas dataframe
data = pd.DataFrame(housing.data, columns=housing.feature_names)
# Add target to data
data['target'] = housing['target']
data.head()
```

Out[127]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	target
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

Part 1a : Check for missing values in the dataset

The dataset might have missing values represented by a `NaN` . Check if the dataset has such missing values.

In [128]:

```
# Check for missing values
def is_null(dataframe):
    """
    This function takes as input a pandas dataframe and outputs whether the
    dataframe has missing values. Missing values can be detected by checkin
    for the presence of None or NaN. inf or -inf must also be treated as a

    Input:
        dataframe: Pandas dataframe
    Output:
        Return True is there are missing value in the dataframe. If not, re
    """
    check_nan = dataframe.isnull().values.any()
    check_inf = np.isinf(dataframe).values.sum()
    if((check_nan) or (check_inf!=0)):
        return True
    else:
        return False


#     raise NotImplementedError()
```

In [129]:

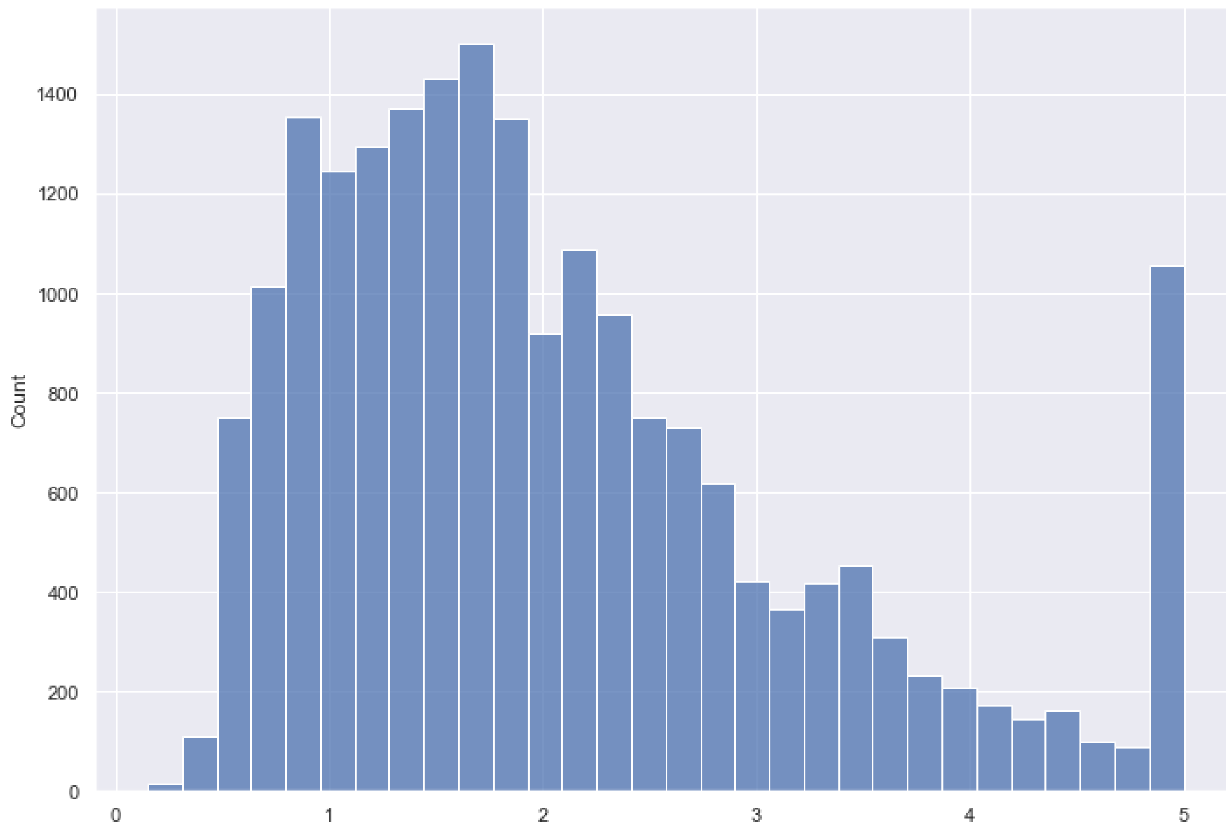
```
# === DO NOT MOVE/DELETE ===
# This cell is used as a placeholder for autograder script injection.

# This dataset has no null values; you can run this cell as a sanity check.
print(f"The data has{'' if is_null(data) else ' no'} missing values.")
assert not is_null(data)
```

```
The data has no missing values.
```

Part 1b: Studying the distribution of the target variable

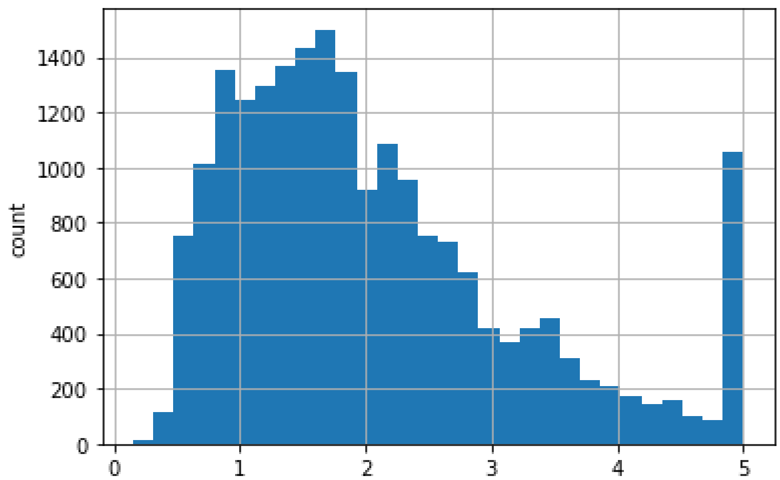
Plot the histogram of the target variable over a fixed number of bins (say, 30).

Example histogram output:



Hint: Use the histogram plotting function available in Seaborn in Matplotlib.

In [130]:
```
# Plot histogram of target variable

# YOUR CODE HERE

plt.hist(data['target'] , bins = 30)
plt.ylabel('count')
plt.grid()
plt.show()


#raise NotImplementedError()
```



Part 1c: Plotting the correlation matrix

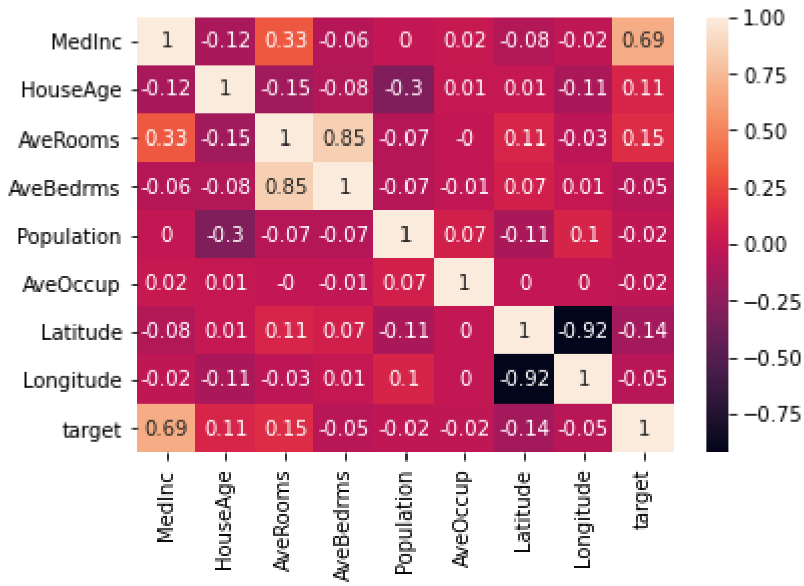
Given the dataset stored in the `data` variable, plot the correlation matrix for the dataset. The dataset has 9 variables (8 features and one target variable) and thus, the correlation matrix must have a size of `9x9` .

Hint: You may use the correlation matrix computation of a dataset provided by the `pandas` library.

Link: [What is a correlation matrix? (https://www.displayr.com/what-is-a-correlation-matrix/)](https://www.displayr.com/what-is-a-correlation-matrix/)

In [131]:
```
# Correlation matrix
def get_correlation_matrix(dataframe):
    """
    Given a pandas dataframe, obtain the correlation matrix
    computing the correlation between the entities in the dataset.

    Input:
        dataframe: Pandas dataframe
    Output:
        Return the correlation matrix as a pandas dataframe, rounded off to
    """
    # YOUR CODE HERE
    corrM = dataframe.corr().round(2);

    return corrM

    #raise NotImplementedError()
# Plot the correlation matrix
correlation_matrix = get_correlation_matrix(data)
# annot = True to print the values inside the square
sns.heatmap(data=correlation_matrix, annot=True)
```

Out[131]: <AxesSubplot:>



In [132]:
```
# === DO NOT MOVE/DELETE ===
# This cell is used as a placeholder for autograder script injection.

# You can check your output against the expected correlation matrix below:
ground_truth = np.array([
    [1.0, -0.12, 0.33, -0.06, 0.0, 0.02, -0.08, -0.02, 0.69],
    [-0.12, 1.0, -0.15, -0.08, -0.3, 0.01, 0.01, -0.11, 0.11],
    [0.33, -0.15, 1.0, 0.85, -0.07, 0.0, 0.11, -0.03, 0.15],
    [-0.06, -0.08, 0.85, 1.0, -0.07, -0.01, 0.07, 0.01, -0.05],
    [0.0, -0.3, -0.07, -0.07, 1.0, 0.07, -0.11, 0.1, -0.02],
    [0.02, 0.01, 0.0, -0.01, 0.07, 1.0, 0.0, 0.0, -0.02],
    [-0.08, 0.01, 0.11, 0.07, -0.11, 0.0, 1.0, -0.92, -0.14],
    [-0.02, -0.11, -0.03, 0.01, 0.1, 0.0, -0.92, 1.0, -0.05],
    [0.69, 0.11, 0.15, -0.05, -0.02, -0.02, -0.14, -0.05, 1.0],
])
assert np.allclose(ground_truth, get_correlation_matrix(data).to_numpy(), r
```

Part 1d: Extracting relevant variables

Based on the correlation matrix obtained in the previous part, identify the top-4 most relevant features from the dataset for predicting the target variable.

In [151]:
```
new_df = correlation_matrix.sort_values(by="target", ascending = False)
ind_list = list(new_df.index)[1:5];
print(ind_list)
```

```
['MedInc', 'AveRooms', 'HouseAge', 'Population']
```

Part 2: Data Manipulation

This section is focused on arranging the dataset in a format suitable for training the linear regression model.

Part 2a: Normalize the dataset

Find the mean and standard deviation corresponding to each feature and target variable in the dataset. Use the values of the mean and standard deviation to normalize the dataset.

In [136]:
```
features = np.concatenate([data[name].to_numpy()[:, None] for name in housi
target = housing['target']

# Normalize data
def normalize(features, target):
        # YOUR CODE HERE
    features_mean=np.mean(features, axis = 0)
    features_std=np.std(features, axis = 0)
    target_mean=np.mean(target)
    target_std=np.std(target)
    new_features=np.zeros((len(features[:]),len(features[:][0])))
    new_target=np.zeros(np.size(target))

    for i in range(len(features[:])):
        for j in range(len(features[:][0])):
            new_features[i][j]=(features[i][j]-features_mean[j])/features_s

    for i in range(np.size(target)):
        new_target[i]=(target[i]-target_mean)/target_std


    return new_features, new_target


        #raise NotImplementedError()

features_normalized, target_normalized = normalize(features, target)
```

In [137]:
```
# === DO NOT MOVE/DELETE ===
# This cell is used as a placeholder for autograder script injection.
assert all(np.abs(features_normalized.mean(axis=0)) < 1e-2), "Mean should b
assert all(np.abs(features_normalized.std(axis=0) - 1) < 1e-2), "Standard d
assert np.abs(target_normalized.mean(axis=0)) < 1e-2, "Mean should be close
assert np.abs(target_normalized.std(axis=0) - 1) < 1e-2, "Standard deviatio
```

Part 2b: Train-Test Split

Use the train-test split function from `sklearn` and execute a 80-20 train-test split of the dataset.

In [115]:
```
# YOUR CODE HERE
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(features_normalized,tar
#raise NotImplementedError()
```

In [116]:
```
# === DO NOT MOVE/DELETE ===
# This cell is used as a placeholder for autograder script injection.

# Sanity checking:
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(16512, 8)
(4128, 8)
(16512,)
(4128,)
```

Part 3: Linear Regression

In this part, a linear regression model is used to fit the dataset loaded and normalized above.

Part 3a: Code for Linear Regression

Implement a closed-form solution for ordinary least squares linear regression in `MyLinearRegression` , and print out the RMSE and R^2 between the ground truth and the model prediction.

In [117]:
```
class MyLinearRegression:
    def __init__(self):
        self.theta = None

    def fit(self, X, Y):
        # Given X and Y, compute theta using the closed-form solution for l
        # YOUR CODE HERE
        XTX = np.matmul(np.transpose(X),X);
        XTX1 = np.linalg.inv(XTX);
        Xmul = np.matmul(XTX1,np.transpose(X));
        self.theta = np.matmul(Xmul,np.transpose(Y));
        #raise NotImplementedError()

    def predict(self, X):
        # Predict Y for a given X
        # YOUR CODE HERE
        return np.matmul(X,self.theta);
        #raise NotImplementedError()
```

In [118]:
```
# Train the model on (X_train, Y_train) using Linear Regression
my_model = MyLinearRegression()
my_model.fit(X_train, Y_train)
```

In [119]:
```
from sklearn.metrics import mean_squared_error, r2_score

# Compute train RMSE using (X_train, Y_train)
y_train_predict = my_model.predict(X_train)
train_rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict)))
train_r2 = r2_score(Y_train, y_train_predict)
print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(train_rmse))
print('R2 score is {}'.format(train_r2))
print("\n")

# Compute test RMSE using (X_test, Y_test)
y_test_predict = my_model.predict(X_test)
test_rmse = (np.sqrt(mean_squared_error(Y_test, y_test_predict)))
test_r2 = r2_score(Y_test, y_test_predict)
print("The model performance for testing set")
print("--------------------------------------")
print('RMSE is {}'.format(test_rmse))
print('R2 score is {}'.format(test_r2))
```

```
The model performance for training set
--------------------------------------
RMSE is 0.6269848126925497
R2 score is 0.608893585337847


The model performance for testing set
--------------------------------------
RMSE is 0.6302717773501296
R2 score is 0.5943507042437144
```

Part 3b: Compare with LinearRegression from sklearn.linear_model

Use LinearRegression from the `sklearn` package to fit the dataset and compare the results obtained with your own implementaion of Linear Regression.

The linear regressor should be named `model` for the cells below to run properly.

In [120]:
```
# YOUR CODE HERE
from sklearn.linear_model import LinearRegression
model = LinearRegression().fit(X_train, Y_train);
#raise NotImplementedError()
```

In [121]:
```
# model evaluation for training set
y_train_predict = model.predict(X_train)
sklearn_train_rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict))
sklearn_train_r2 = r2_score(Y_train, y_train_predict)

print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(sklearn_train_rmse))
print('R2 score is {}'.format(sklearn_train_r2))
print("\n")

# model evaluation for testing set
y_test_predict = model.predict(X_test)
sklearn_test_rmse = (np.sqrt(mean_squared_error(Y_test, y_test_predict)))
sklearn_test_r2 = r2_score(Y_test, y_test_predict)

print("The model performance for testing set")
print("--------------------------------------")
print('RMSE is {}'.format(sklearn_test_rmse))
print('R2 score is {}'.format(sklearn_test_r2))
```

```
The model performance for training set
--------------------------------------
RMSE is 0.626982226453801
R2 score is 0.6088968118672872


The model performance for testing set
--------------------------------------
RMSE is 0.6302930934638351
R2 score is 0.5943232652466204
```

Part 3c: Analysis Linear Regression Performance

In this section, provide the observed difference in performance along with an explanation of the following:

- Difference between training between unnormalized and normalized data.
- Difference between training on all features versus training on the top-5 most relevant features in the dataset.
- Difference between (1) training on all features (unnormalized), (2) training on top-4 unnormalized features, and (3) training on top-4 normalized features.

Write your answer below.

YOUR ANSWER HERE

In [169]:
```
print("1. The difference between training between unnormalized and normalize
from sklearn.model_selection import train_test_split
unnormX_train, unnormX_test, unnormY_train, unnormY_test = train_test_split(

my_newmodel = MyLinearRegression()
my_newmodel.fit(unnormX_train, unnormY_train)

unnormy_train_predict = my_newmodel.predict(unnormX_train)
newsklearn_train_rmse = (np.sqrt(mean_squared_error(unnormY_train, unnormy_t
newsklearn_train_r2 = r2_score(unnormY_train,unnormy_train_predict)

print("The output of the unnormalized model and the normalized model:")
print("The unnormalized model:")
print("--------------------------------------")
print('RMSE is {}'.format(newsklearn_train_rmse))
print('R2 score is {}'.format(newsklearn_train_r2))
print("\n")


print("The normalized model:")
print("--------------------------------------")
print('RMSE is {}'.format(sklearn_train_rmse))
print('R2 score is {}'.format(sklearn_train_r2))
print("\n")
print("From the above outputs, we notice that the RMSE in the unnormalized m
print("As for R2 score, it is smaller in the unnormalized data, which means
print("Explanation:")
print("The columns of unnormalized data may have the different unit between
print("\n")
print("2. Difference between training on all features versus training on the
np.set_printoptions(suppress=True)
new_df = correlation_matrix.sort_values(by="target", ascending = False)
my_list = list(new_df.index)[0:6]
new_data = data[my_list]
my_list.remove("target")
new_features = np.concatenate([new_data[name].to_numpy()[:, None] for name i
new_target = new_data['target']
new_normfeatures,new_normtarget = normalize(new_features,new_target)
T5X_train, T5X_test, T5Y_train, T5Y_test = train_test_split(new_normfeatures
my_newmodel = MyLinearRegression()
my_newmodel.fit(T5X_train, T5Y_train)
T5y_train_predict = my_newmodel.predict(T5X_train)
T5newsklearn_train_rmse = (np.sqrt(mean_squared_error(T5Y_train, T5y_train_p
T5newsklearn_train_r2 = r2_score(T5Y_train,T5y_train_predict)
print("\n")
print("The normalized model:")
print("--------------------------------------")
print('RMSE is {}'.format(sklearn_train_rmse))
print('R2 score is {}'.format(sklearn_train_r2))
print("\n")

print("Top 5 relevant features model:")
print("--------------------------------------")
print('RMSE is {}'.format(T5newsklearn_train_rmse))
print('R2 score is {}'.format(T5newsklearn_train_r2))
print("\n")
```

```
print("From the above outputs, we notice that the RMSE in the top 5 relevant
print("As for R2 score, it is smaller in the top 5 relevant features model,
print("Explanation:")
print("The top 5 relevant features model have less features compare with the
print("\n")

print("3. Difference between (1) training on all features (unnormalized), (2
my_list = list(new_df.index)[0:5]
my_list.remove("target")
new_features = np.concatenate([new_data[name].to_numpy()[:, None] for name i
new_target = new_data['target']
T4X_train, T4X_test, T4Y_train, T4Y_test = train_test_split(new_features,new
my_newmodel = MyLinearRegression()
my_newmodel.fit(T4X_train, T4Y_train)
T4y_train_predict = my_newmodel.predict(T4X_train)
T4newsklearn_train_rmse = (np.sqrt(mean_squared_error(T4Y_train, T4y_train_p
T4newsklearn_train_r2 = r2_score(T4Y_train,T4y_train_predict)


print("The unnormalized model:")
print("--------------------------------------")
print('RMSE is {}'.format(newsklearn_train_rmse))
print('R2 score is {}'.format(newsklearn_train_r2))
print("\n")


print("The top 4 relevant features model:")
print("--------------------------------------")
print('RMSE is {}'.format(T4newsklearn_train_rmse))
print('R2 score is {}'.format(T4newsklearn_train_r2))
print("\n")

new_normfeatures,new_normtarget = normalize(new_features,new_target)
T4X_normtrain, T4X_normtest, T4Y_normtrain, T4Y_normtest = train_test_split(
my_newmodel = MyLinearRegression()
my_newmodel.fit(T4X_normtrain, T4Y_normtrain)
T4y_normtrain_predict = my_newmodel.predict(T4X_normtrain)
T4newsklearn_train_rmse = (np.sqrt(mean_squared_error(T4Y_normtrain, T4y_nor
T4newsklearn_train_r2 = r2_score(T4Y_normtrain,T4y_normtrain_predict)

print("The normalized top 4 relevant features model: ")
print("--------------------------------------")
print('RMSE is {}'.format(T4newsklearn_train_rmse))
print('R2 score is {}'.format(T4newsklearn_train_r2))
print("\n")
print("From above, we notice that the RMSE of the unnormalized top4 relevant
print("The R2 of the normalized and unnormalized of top 4 relevant features
print("Explanation:")
print("Compare with (1) training on all features (unnormalized), the  (2) tr
print("because (2) is unnormalized, it may have many different units, which
print("Compare with the (2)unnormalized top 4 features model, the (3) normal
print("The R2 of (2) and (3) are similar to each other, because both of them
```

```
1. The difference between training between unnormalized and normalized da
ta.
The output of the unnormalized model and the normalized model:
The unnormalized model:
--------------------------------------
RMSE is 0.7763476606405088
R2 score is 0.5496648800413839


The normalized model:
--------------------------------------
RMSE is 0.626982226453801
R2 score is 0.6088968118672872


From the above outputs, we notice that the RMSE in the unnormalized model
is larger than the one in normalized model,which mean that the unnormaliz
ed model has greater error.The lower the RMSE, the better a given model i
s able to “fit” a dataset.
As for R2 score, it is smaller in the unnormalized data, which means that
there are less proportion of the variance for a dependent variable in unn
ormalized data.
Explanation:
The columns of unnormalized data may have the different unit between each
other, which may has a bad influence on the output. On the other hand, wh
en we normalize the data, we scale the value between 0 and 1.Thus, normal
ization generates new values under the same scale while maintains the dis
tribution of the data.


2. Difference between training on all features versus training on the top
-5 most relevant features in the dataset.


The normalized model:
--------------------------------------
RMSE is 0.626982226453801
R2 score is 0.6088968118672872


Top 5 relevant features model:
--------------------------------------
RMSE is 0.6947743825638367
R2 score is 0.5197487647160381


From the above outputs, we notice that the RMSE in the top 5 relevant fea
tures model is larger than the one in normalized model,which mean that th
e top 5 relevant features model has greater error.The lower the RMSE, the
better a given model is able to “fit” a dataset.
As for R2 score, it is smaller in the top 5 relevant features model, whic
```

```
h means that there are less proportion of the variance for a dependent va
riable in unnormalized data.
Explanation:
The top 5 relevant features model have less features compare with the nor
malized model, which will have a big influence on the accuracy


3. Difference between (1) training on all features (unnormalized), (2) tr
aining on top-4 unnormalized features, and (3) training on top-4 normaliz
ed features
The unnormalized model:
--------------------------------------
RMSE is 0.7763476606405088
R2 score is 0.5496648800413839


The top 4 relevant features model:
--------------------------------------
RMSE is 0.8029960095151611
R2 score is 0.5182185281856457


The normalized top 4 relevant features model:
--------------------------------------
RMSE is 0.695809148206016
R2 score is 0.5183171703569155


From above, we notice that the RMSE of the unnormalized top4 relevant fea
tures model and the unnormalized model are greater than the normalized mo
del with 4 relevant features
The R2 of the normalized and unnormalized of top 4 relevant features mode
l are both less than the unnormalized model of all features.
Explanation:
Compare with (1) training on all features (unnormalized), the  (2) traini
ng on top-4 unnormalized features have greature RMSE and smaller R2,
because (2) is unnormalized, it may have many different units, which can
lead a bad influence on the RMSE, and the the features of (2) are way mor
e less than (1), so there are less variability explained by (2)
Compare with the (2)unnormalized top 4 features model, the (3) normalized
top 4 features model's RMSE are way more less than (2), which means that
(3) are way more accurate than (2) because of normalization
The R2 of (2) and (3) are similar to each other, because both of them hav
e only few of the features compare with (1),which means that both of the
variabilities explained by them are less than (1).
```

In [ ]: