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

Alvin Lee

September 20, 2022

Solutions

Question 1

It is not reasonable to assign equal weights to the cost associated with each training example. By definition we can define the expected true cost function as:

$$\mathbb{E}_x[E(g(x), f(x))] = \sum_{i=1}^{N} E(g(x), f(x))p_X(x) \tag{1}$$

The equation given in the problem assumes that $p_X(x)$ which is the probability that the event x occurs is $\frac{1}{N}$. Since we established that not every data x is equally likely we need to weigh each per-example cost differently. We should weigh it based on the probability $p_X(x)$ (the probability that the event x occurs)

Question 2

1. We can find the expected value of Y given a number by the following:

$$\mathbb{E}[Y|X = 0] = \mathbb{E}(5 + 0.5(0) + \epsilon_i) = \mathbb{E}(5) + \mathbb{E}(\epsilon_i) = 5 + 0 = 5$$

$$\mathbb{E}[Y|X = -2] = \mathbb{E}(5 + 0.5(-2) + \epsilon_i) = \mathbb{E}(5) + \mathbb{E}(-1) + \mathbb{E}(\epsilon_i) = 5 - 1 = 4$$

To find varience, we can use the following expression:

$$Var(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2 \tag{2}$$

$\forall x \in \mathbb{R}$,

$$Var(Y|X = x) = \mathbb{E}((5 + 0.5x + \epsilon_i)^2) - \mathbb{E}(5 + 0.5x + \epsilon_i)^2$$

Expanding this out, we get:

$$= \mathbb{E}(25 + 0.25x^2 + \epsilon_i^2 + 5x + 10\epsilon_i + x\epsilon_i) - (\mathbb{E}(5) + \mathbb{E}(0.5x))^2$$

We know that $\mathbb{E}(\epsilon_i^2) = Var(\epsilon_i) + \mathbb{E}(\epsilon_i)^2 = Var(\epsilon_i) = 1$. Since x is constant,

$$= 25 + 0.25x^2 + 5x + 1 - (5 + 0.5x)^2 = 1$$

Since $\forall x \in \mathbb{R},\ Var(Y|X = x) = 1 \implies Var(Y|X) = 1$

2. The probability of $Y > 5$ given $X = 2$ is:

$$P(5 + 1 + \epsilon_i > 5) = P(\epsilon_i > -1)$$

Given that $\epsilon_i \overset{idd}{\sim} N(0, 1)$ we can find

$$P(\epsilon_i > -1) = \int_{-1}^{\infty} \frac{e^{-\frac{x^2}{2}}}{\sqrt{2\pi}}$$

We can use the z-score table to estimate this integral:

$$P(\epsilon_i > -1) \approx 0.8413$$

3. Given $\mathbb{E}[X] = 0$ and $Var(X) = 10$

$$\mathbb{E}[Y] = \mathbb{E}[5 + X_i + \epsilon_i] = \mathbb{E}[5] + \mathbb{E}[X] + \mathbb{E}[\epsilon_i] = 5 + 0 + 0 = 5$$

$$Var[Y] = \mathbb{E}[Y^2] - \mathbb{E}[Y]^2$$
$$= \mathbb{E}[25 + 0.25X_i^2 + \epsilon_i^2 + 5X_i + 10\epsilon_i + X_i\epsilon_i] - 25$$
$$= 25 + 0.25\mathbb{E}[X_i^2] + \mathbb{E}[\epsilon_i^2] - 25$$

We know that $\mathbb{E}[\alpha^2] = Var(\alpha) + \mathbb{E}[\alpha]^2$. Since $\mathbb{E}[X_i] = 0$ and $\mathbb{E}[\epsilon_i] = 0$,

$$= 0.25(10) + 1 = 3.5$$

4. $Cov(X, Y) = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y] = \mathbb{E}[XY] - 0 = \mathbb{E}[X(5 + 0.5X + \epsilon_i)] = \mathbb{E}[5X + 0.5X^2 + \epsilon_i X] = 0 + 0.5(10) + 0 = 5$

Question 3

For each observation, when we derive when $\hat{\theta}$ is at the minima, the derivative of the loss function is:

$$\hat{\theta}_i = (xx^T)^{-1}x(y - \epsilon_i) \tag{3}$$

Given a set of N observations, we can give equal weights to each observation, finding that the approximate $\hat{\theta}$ is:

$$\mathbb{E}(\hat{\theta}) = \frac{1}{N}\sum_{i=0}^{N}(xx^T)^{-1}x(y - \epsilon_i) \tag{4}$$

However, given a weight w_i for each x_i we can find the weighted least squares estimate $\hat{\theta}$ as:

$$\mathbb{E}(\hat{\theta}) = \sum_{i=0}^{N} w_i(xx^t)^{-1}x(y - \epsilon_i) \tag{5}$$

Question 4

A linear regression is used to predict an n x 1 matrix given a d x n matrix. On the other hand, a logistic regression is used for classification. For example, it will take an input of data and the output is a classification placed into a bin, either 0 or 1. An equation for linear regression is $Y = X^T w$, where Y is a vector of n x 1 On the other hand, the equation for a logistic regression is a probability function, $p(y) = \frac{e^{w^T x}}{1+e^{w^T x}}$.

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 [63]:
```
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 [64]:
```
# Load dataset
from sklearn.datasets import fetch_california_housing
housing = fetch_california_housing()
```

Part 1 : Analyse the dataset

In [65]:
```
# 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[65]:

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

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 [66]:
```
# Check for missing values
def is_null(dataframe):
    """
    This function takes as input a pandas dataframe and outputs whethe
    dataframe has missing values. Missing values can be detected by ch
    for the presence of None or NaN. inf or -inf must also be treated

    Input:
        dataframe: Pandas dataframe
    Output:
        Return True is there are missing value in the dataframe. If no
    """
    return np.nan in dataframe.values or np.inf in dataframe.values or
    raise NotImplementedError()
```

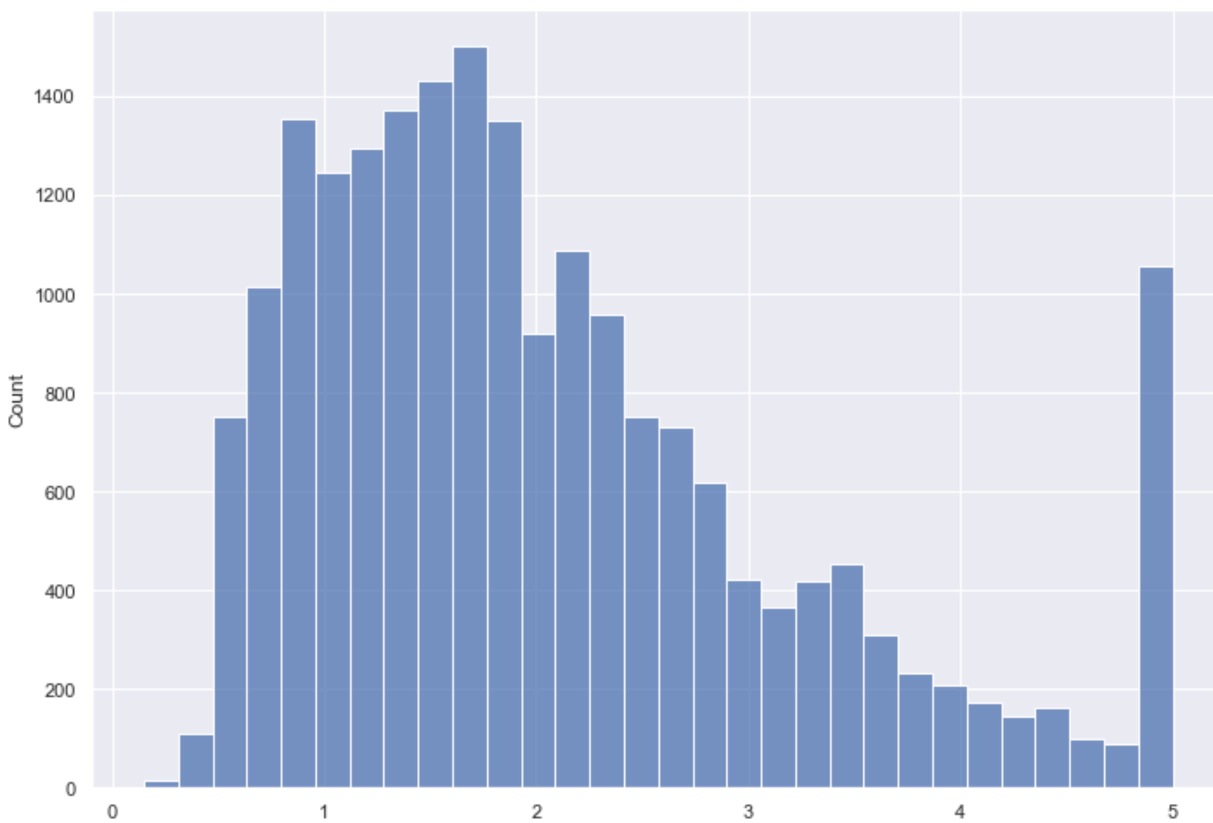
In [67]:
```
# === DO NOT MOVE/DELETE ===
# This cell is used as a placeholder for autograder script injection.

# This dataset has no null values; you can run this cell as a sanity c
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).

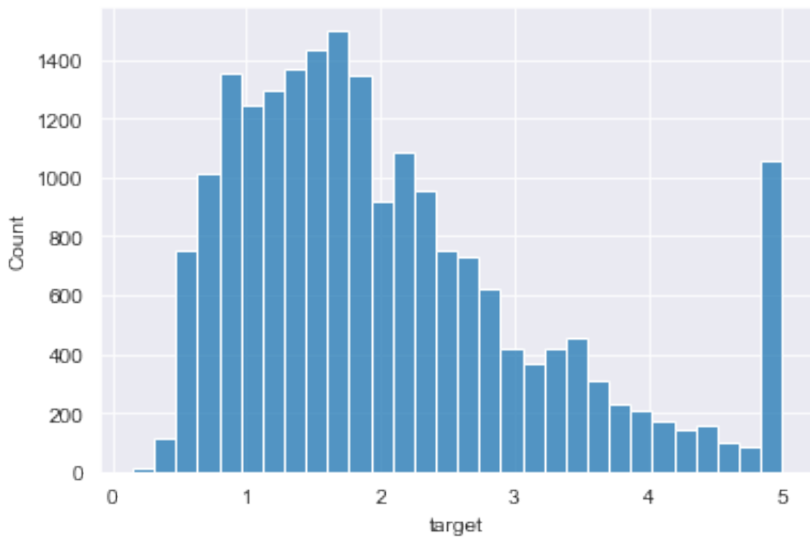
Example histogram output:



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

In [68]:
```
# Plot histogram of target variable

# YOUR CODE HERE
sns.histplot(data["target"], bins=30)
```

Out[68]: <AxesSubplot:xlabel='target', ylabel='Count'>



Part 1c: Plotting the correlation matrix

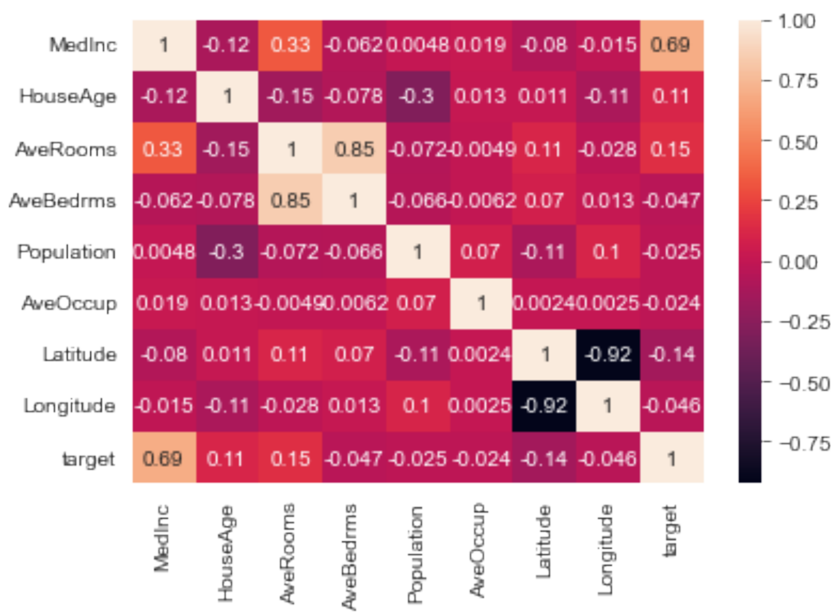
Given the dataset stored in the `data` variable, plot the correlation matrix for the dataset. The dataset has 9 variables (8 features and one target variable) and thus, the correlation matrix must have a size of `9x9` .

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 [69]:
```
# 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 c
    """
    # YOUR CODE HERE
    corrM = data.corr()
    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[69]: <AxesSubplot:>



In [70]:
```
# === 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 be
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
```

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.

The top-4 most relevant features are MedInc, AveRooms, HouseAge, AveOccup.

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 [71]:
```
features = np.concatenate([data[name].to_numpy()[:, None] for name in
target = housing['target']

# Normalize data
def normalize(features, target):

    for i in range(len(features[0])):
        features[:,i] = (features[:,i] - features[:,i].mean())/feature

    target = (target - target.mean())/target.std()

    return features,target

    raise NotImplementedError()

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

In [72]:
```

# === 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 sho
assert all(np.abs(features_normalized.std(axis=0) - 1) < 1e-2), "Stanc
assert np.abs(target_normalized.mean(axis=0)) < 1e-2, "Mean should be
assert np.abs(target_normalized.std(axis=0) - 1) < 1e-2, "Standard dev
```

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 [73]:
```
# YOUR CODE HERE
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(features_normalize
```

In [74]:
```
# === 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 [75]:
```
class MyLinearRegression:
    def __init__(self):
        self.theta = None
        self.W = None

    def fit(self, X, Y):
        # Given X and Y, compute theta using the closed-form solution
        # YOUR CODE HERE

        column = np.ones((len(X),1))
        X = np.append(X,column, axis=1)

        Xt = np.transpose(X)

        W = np.matmul(np.linalg.inv(np.matmul(Xt,X)),np.matmul(Xt,Y))

        self.theta = W[-1]
        self.W = W[:-1]
        return W
        raise NotImplementedError()

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

In [76]:
```
# Train the model on (X_train, Y_train) using Linear Regression
my_model = MyLinearRegression()
my_model.fit(X_train, Y_train)
```

Out[76]:
```
array([ 0.70590566,  0.10470833, -0.20177254,  0.23504992, -0.0015603
9,
       -0.03131917, -0.77621997, -0.75156345, -0.0024601 ])
```

In [77]:
```
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 0.6266691251111148
R2 score is 0.6028927820797623


The model performance for testing set
--------------------------------------
RMSE is 0.6315196824041804
R2 score is 0.6177097919203134
```

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 [78]:
```
# YOUR CODE HERE
from sklearn.linear_model import LinearRegression
model = LinearRegression().fit(X_train,Y_train)
```

In [79]:
```
# model evaluation for training set
y_train_predict = model.predict(X_train)
sklearn_train_rmse = (np.sqrt(mean_squared_error(Y_train, y_train_prec
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 0.6266691251111148
R2 score is 0.6028927820797623


The model performance for testing set
--------------------------------------
RMSE is 0.6315196824041804
R2 score is 0.6177097919203134
```

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.

In [84]:
```
features = np.concatenate([data[name].to_numpy()[:, None] for name in
target = housing['target']


X_train, X_test, Y_train, Y_test = train_test_split(features, target,

model = LinearRegression().fit(X_train,Y_train)
y_train_predict = model.predict(X_train)
sklearn_train_rmse = (np.sqrt(mean_squared_error(Y_train, y_train_prec
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 0.7186846170820259
R2 score is 0.607328701848081


The model performance for testing set
--------------------------------------
RMSE is 0.7478294402580221
R2 score is 0.5993031256212871
```

In the unnormalized data, the RMSE is higher. This implies that the predicted value is further from the actual values. The R2 scores are similar implying that both models account for a similar varience.

In [85]:
```
.concatenate([data[name].to_numpy()[:, None] for name in housing['feat
ing['target']


st, Y_train, Y_test = train_test_split(features, target, train_size = (

rRegression().fit(X_train,Y_train)
ct = model.predict(X_train)
_rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict)))
_r2 = r2_score(Y_train, y_train_predict)

del performance for training set")
--------------------------------")
s {}'.format(sklearn_train_rmse))
re is {}'.format(sklearn_train_r2))


ation for testing set
t = model.predict(X_test)
rmse = (np.sqrt(mean_squared_error(Y_test, y_test_predict)))
r2 = r2_score(Y_test, y_test_predict)

del performance for testing set")
--------------------------------")
s {}'.format(sklearn_test_rmse))
re is {}'.format(sklearn_test_r2))
```

```
The model performance for training set
--------------------------------------
RMSE is 0.7805632904621855
R2 score is 0.5427663656407771


The model performance for testing set
--------------------------------------
RMSE is 0.7883246581914368
R2 score is 0.5317007729082652
```

Type *Markdown* and LaTeX: α^2

When we only train the top-5 relevant features, the R2 score decreases and the RMSE increases. This is likely due to the fact that the model is only training on data that has less varience. This means it's overfitting for that data and not accounting for the varience in the other data.

In [86]:
```
features = np.concatenate([data[name].to_numpy()[:, None] for name in
target = housing['target']


X_train, X_test, Y_train, Y_test = train_test_split(features, target,

model = LinearRegression().fit(X_train,Y_train)
y_train_predict = model.predict(X_train)
sklearn_train_rmse = (np.sqrt(mean_squared_error(Y_train, y_train_prec
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 0.7860838893506152
R2 score is 0.5386280444285896


The model performance for testing set
--------------------------------------
RMSE is 0.7801106237951789
R2 score is 0.5320221870769073
```

In [87]:
```
features = np.concatenate([data[name].to_numpy()[:, None] for name in
target = housing['target']
features_normalized, target_normalized = normalize(features, target)
X_train, X_test, Y_train, Y_test = train_test_split(features_normalize

model = LinearRegression().fit(X_train,Y_train)
y_train_predict = model.predict(X_train)
sklearn_train_rmse = (np.sqrt(mean_squared_error(Y_train, y_train_prec
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 0.6775321873625405
R2 score is 0.5380683421740401


The model performance for testing set
--------------------------------------
RMSE is 0.6902498758555163
R2 score is 0.5351292962754159
```

When taking the top-4 normalized features, the R2 score is similar to the R2 score of the unnormalized features, however, the RMSE is much lower which means that the error of the predicted values are less. However, it seems that taking the normalized features of all features is the best way to train the data.