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

Instructor: Lerrel Pinto

September 20, 2022

Submission Instructions

You must typeset the answers using LaTeX 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.

Questions

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

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

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

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.

Answer:

Since every data x is not equally like, it is not okay to give an equal weight to the cost associated with each training example and not reasonable to divide the sum of costs by N. In this case, we should weigh each per-example cost differently based on the likelihood of x.

For example, if we want to train a classification model with 300 images of cats and 700 images of dogs, which has an imbalance distribution in the training sample. If the model uses equally weighted costs, the model would focus more on minimizing the costs of identifying dogs, which would has better performance in identifying dogs and poorer performance in identifying cats. Hence, simply dividing the sum of costs by 1000 is not reasonable, since the model focuses more on dogs, so we need to add more weights on cats to reduce the bias.

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 $\mathbb{E}[Y|X=0]$, $\mathbb{E}[Y|X=-2]$ and $Var[Y|X]$?
2. What is the probability of $Y > 5$, given $X = 2$?
3. If X has a mean of zero and variance of 10, what are $\mathbb{E}[Y]$ and $Var[Y]$?
4. What is $Cov(X,Y)$?

Answer:

1.
$\mathbb{E}[Y|X=0] = \mathbb{E}[5+0.5X+\epsilon|X=0] = 0.5\,\mathbb{E}[X] + 5 + \mathbb{E}[\epsilon] = 0+5+0 = 5$
$\mathbb{E}[Y|X=-2] = \mathbb{E}[5+0.5X+\epsilon|X=0] = 0.5\,\mathbb{E}[X]+5+\mathbb{E}[\epsilon] = 0.5\cdot(-2)+5+0 = 4$
$Var[Y|X] = Var[5+0.5X+\epsilon|X] = Var[0.5X+\epsilon|X] = Var[\epsilon|X] = Var[\epsilon] = 1$

2.
$\mathbb{E}[Y|X=2] = \mathbb{E}[5+0.5X+\epsilon|X=0] = 0.5\,\mathbb{E}[X] + 5 + \mathbb{E}[\epsilon] = 0.5\cdot(2)+5+0 = 6$
$Z = \frac{5-\mathbb{E}[Y|X=2]}{\sigma} = (5-6)/1 = -1$
$P(Y>5|X=2) = 1 - \phi(Z=-1) = 1 - 0.15866 = 0.84114 \approx 0.84$

3.
Given that $\mathbb{E}[X] = 0$ and $Var[X] = 10$,
$\mathbb{E}[Y] = \mathbb{E}[5+0.5X+\epsilon] = 0.5\,\mathbb{E}[X] + 5 + \mathbb{E}[\epsilon] = 0+5+0 = 5$
$Var[Y] = Var[5+0.5X+\epsilon] = Var[0.5X+\epsilon] = 0.5^2 Var[X] + Var[\epsilon] + 2Cov(0.5X,\epsilon) = 0.5^2\cdot 10 + 1 + 0 = 3.5$

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

Question3: Least Squares Regression (10 points)

Consider the linear regression model:

$$y = \theta_1 x_1 + \theta_2 x_2 + ... + \theta_k x_k + \epsilon, \epsilon \tag{2}$$

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 $\mathbf{x} = [x_1, x_2, ..., x_k]$ in the dataset. This error is captured by $\epsilon \sim N(0, \sigma^2)$, 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 $Var(\epsilon_i|X_i) = \sigma^2$, where σ is a constant value. However, when $Var(\epsilon_i|X_i) = f(X_i) \neq \sigma^2$, 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.

Answer:

$y_i = \hat{y_i} + \epsilon = x_i^T\theta + \epsilon$

The closed form ordinary least squares estimator for θ:

Suppose a candidate of θ is c, the mean squared error is $MSE(c) = \frac{1}{N}\sum_{i=1}^{N}(y_i - x_i^T c)^2$, and the c minimizing this sum is called the OLS estimator for θ.

Thus, $\hat{\theta} = argminMSE(c) = (\frac{1}{N}\sum_{i=1}^{N} x_i x_i^T)^{-1} \cdot \frac{1}{N}\sum_{i=1}^{N} x_i y_i) = (X^T X)^{-1} X^T y$.

The closed form weighted least squares estimator for θ:

Add weight w_i to x_i, y_i.

Suppose a candidate of θ is c, the mean squared error is $MSE(c) = \frac{1}{N}\sum_{i=1}^{N} w_i(y_i - x_i^T c)^2$, and the c minimizing this sum is called the OLS estimator for θ.

Thus, $\hat{\theta} = argminMSE(c) = (\frac{1}{N}\sum_{i=1}^{N} x_i w_i x_i^T)^{-1} \cdot \frac{1}{N}\sum_{i=1}^{N} x_i w_i y_i) = (X^T W X)^{-1} X^T W y$.

Question 4: Linear vs Logistic Regression (5 points)

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

Answer:

Linear regression is used to handle regression problems, which provides a continuous output. One example of linear regression could be poverty rate as input and birth rate as output.

The equation of linear regression is $\hat{y} = \hat{w_0} + \hat{w_1}x_1 + \hat{w_2}x_2 + ... + \hat{w_n}x_n$. And our goal is to find the best fit line that can accurately predict the output, which is minimizing the mean squared error $\frac{1}{n}\sum(y - \hat{y})^2$, where y is the actual output and $\hat{y}$ is the predicted output.

Logistic regression is used to handle the classification problems, which provides discreet output. One example of logistic regression could be poverty rate as input and birth rate > 2 or < 2 as output.

The equation of linear regression is $log[\frac{y}{1-y}] = b_0 + b_1x_1 + b_2x_2 + ... + b_nx_n$. And our goal is to find the S-curve by which we can classify the samples, which is minimizing the log loss $\frac{1}{n}\sum_{i=1}^{n} -y_i log(p_i) - (1 - y_i)log(1 - p_i)$, where y is the actual output and p is the probability predicted by the logistic regression.

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.

```
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.

```
# Load dataset
from sklearn.datasets import fetch_california_housing
housing = fetch_california_housing()
```

Part 1 : Analyse the dataset

```
# 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()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitu
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	–122.
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	–122.
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	–122.
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	–122.
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	–122.

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.

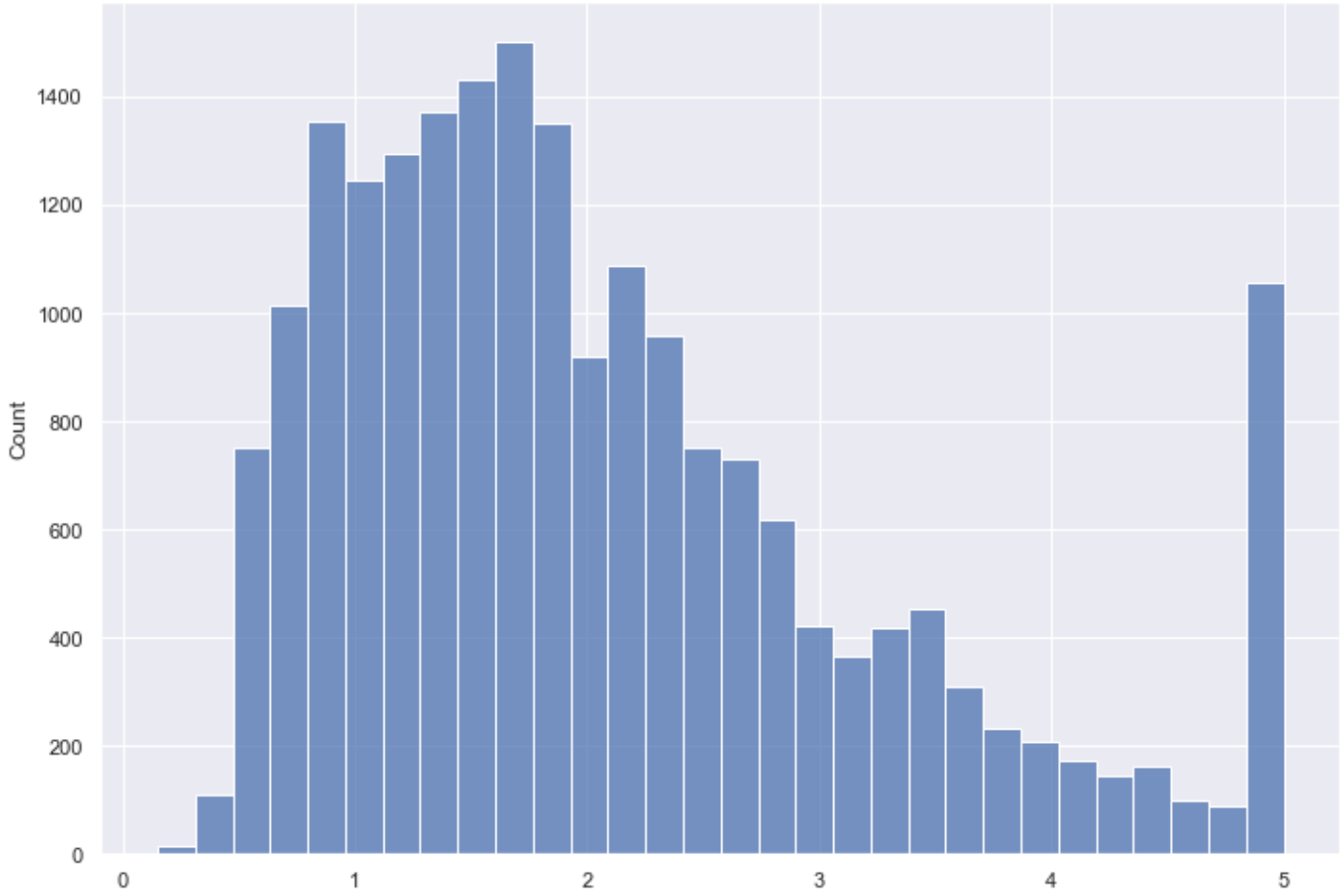
```
# Check for missing values
def is_null(dataframe):
    """
    This function takes as input a pandas dataframe and outputs whether the
    dataframe has missing values. Missing values can be detected by checking
    for the presence of None or NaN. inf or -inf must also be treated as a missing val

    Input:
        dataframe: Pandas dataframe
    Output:
        Return True is there are missing value in the dataframe. If not, return False.
    """
    # YOUR CODE HERE
    count = np.isinf(dataframe).values.sum() + dataframe.isnull().sum().sum()
    if count == 0:
      return False
    else:
      return True
    # raise NotImplementedError()


# === 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).

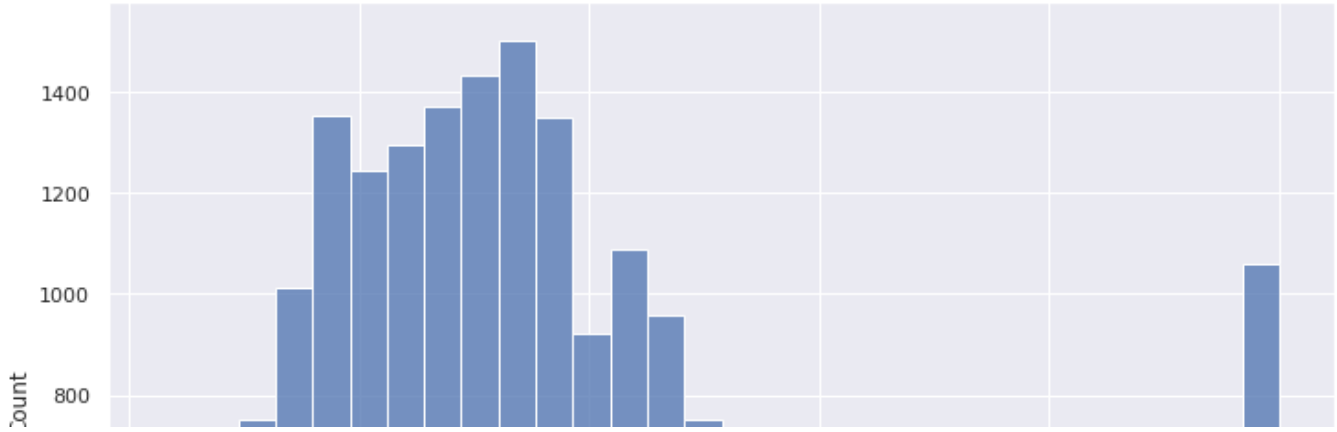
Example histogram output:



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

```
# Plot histogram of target variable

# YOUR CODE HERE
sns.set(rc={"figure.figsize":(12, 8)})
sns.histplot(data = data , x = "target", bins = 30)
# raise NotImplementedError()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58de49bc50>
```



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`.

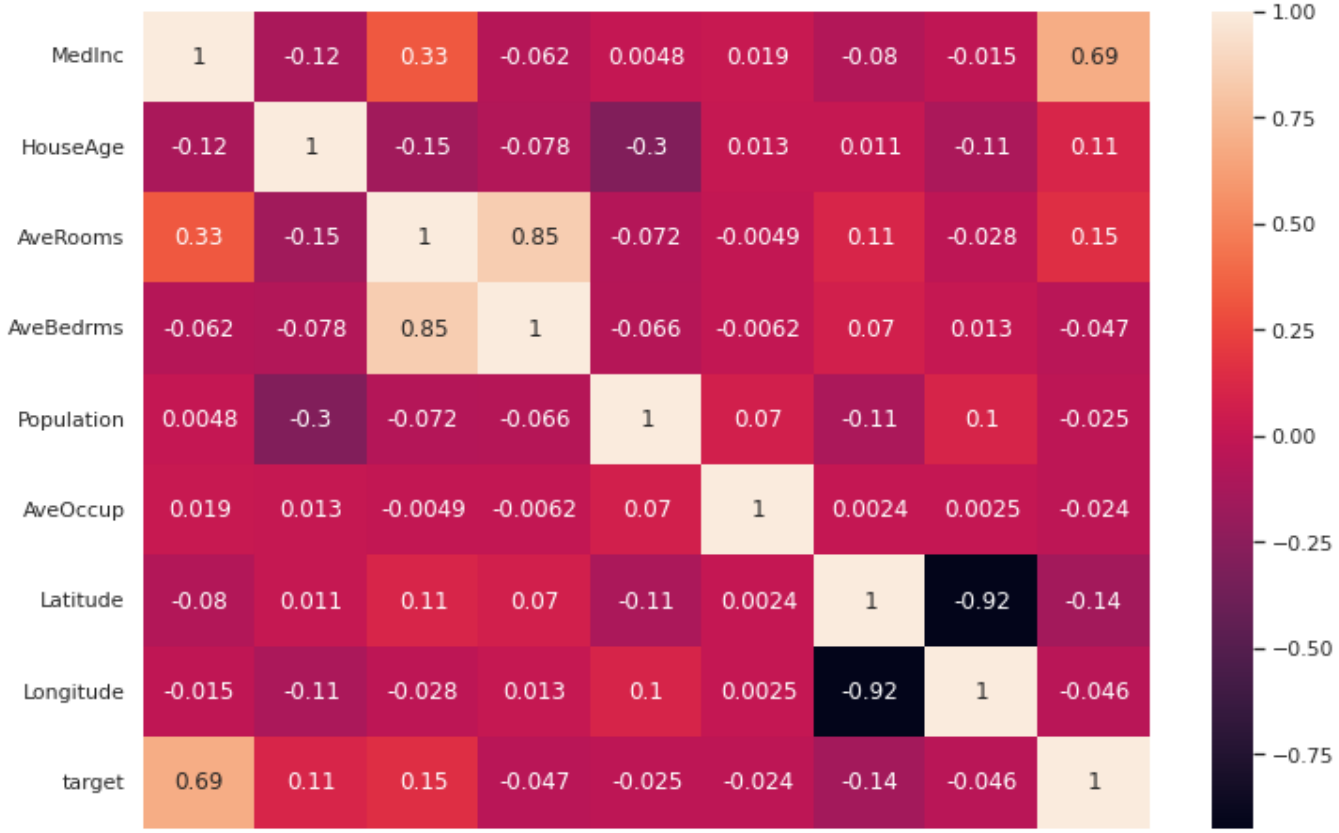
Hint: You may use the correlation matrix computation of a dataset provided by the `pandas` library.

Link: [What is a correlation matrix?]

target

```
# 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 2 decimal
    """
    # YOUR CODE HERE
    correlation_matirx = dataframe.corr()
    return correlation_matirx
    # 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)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58dd8bb8d0>
```




```
# === 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(), rtol=1e-2, a
```

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.

1. MedInc
2. AveRooms

3. Latitude
4. HouseAge

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.

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

# Normalize data
def normalize(features, target):
    # YOUR CODE HERE

    features_std = np.std(features, axis = 0)
    features_mean = np.mean(features, axis =0)
    features_normalized = (features - features_mean)/(features_std)

    target_std = np.std(target, axis = 0)
    target_mean = np.mean(target, axis =0)
    target_normalized = (target - target_mean)/(target_std)

    return features_normalized, target_normalized
    # raise NotImplementedError()

features_normalized, target_normalized = normalize(features, target)


# === 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 be close to
assert all(np.abs(features_normalized.std(axis=0) - 1) < 1e-2), "Standard deviation sh
assert np.abs(target_normalized.mean(axis=0)) < 1e-2, "Mean should be close to 0"
assert np.abs(target_normalized.std(axis=0) - 1) < 1e-2, "Standard deviation should be
```

Part 2b: Train-Test Split

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

```
# YOUR CODE HERE
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(features_normalized, target_normal
# raise NotImplementedError()


# === 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.

```
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 linear regre
        # YOUR CODE HERE
        # Compute (X^T X)^(-1)
        part1 = np.dot(X.T, X)
        # Compute (X^T Y)
        part2 = np.dot(X.T, Y)
        # Compute theta = (X^T X)^(-1) (X^T Y)
        self.theta = np.linalg.solve(part1, part2)
        # raise NotImplementedError()

    def predict(self, X):
```

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

# Train the model on (X_train, Y_train) using Linear Regression
my_model = MyLinearRegression()
my_model.fit(X_train, Y_train)


from sklearn.metrics import mean_squared_error, r2_score

# Compute train RMSE using (X_train, Y_train)
def train_metrics(X_train, Y_train, my_model):
    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)
def test_metrics(X_test, Y_test, my_model):
    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))

train_metrics(X_train, Y_train, my_model)
test_metrics(X_test, Y_test, my_model)
```

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


The model performance for testing set
--------------------------------------
RMSE is 0.6302717773501297
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.

```
# YOUR CODE HERE
from sklearn.linear_model import LinearRegression
model = LinearRegression().fit(X_train, Y_train)
# raise NotImplementedError()
```

```
# 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-4 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.

```
# (1) Traning unnormalized data on all features
print("Model: unnormalized data(all features)")
X_train_unnor, X_test_unnor, Y_train_unnor, Y_test_unnor = train_test_split(features,
my_model.fit(X_train_unnor, Y_train_unnor)

train_metrics(X_train_unnor, Y_train_unnor, my_model)
test_metrics(X_test_unnor, Y_test_unnor, my_model)
```

```
Model: unnormalized data(all features)
The model performance for training set
--------------------------------------
RMSE is 0.7763476606405088
R2 score is 0.5496648800413839


The model performance for testing set
--------------------------------------
RMSE is 0.7823476431899948
R2 score is 0.5306065925145123
```

```
# (2) training unnormalized data on the top-4 most relevant features (MedInc, HouseAge
print("Model: unnormalized data(top4 relevant features)")
X_train_unnor_new = X_train_unnor[:,[0,1,2,6]]
X_test_unnor_new = X_test_unnor[:,[0,1,2,6]]
my_model.fit(X_train_unnor_new, Y_train_unnor)

train_metrics(X_train_unnor_new, Y_train_unnor, my_model)
test_metrics(X_test_unnor_new, Y_test_unnor, my_model)
```

```
Model: unnormalized data(top4 relevant features)
The model performance for training set
--------------------------------------
RMSE is 0.8030689808406747
R2 score is 0.5181309615487499


The model performance for testing set
--------------------------------------
RMSE is 0.8170413397548135
R2 score is 0.48805242444865515
```

```
# (3) training normalized data on the top-4 most relevant features (MedInc, HouseAge,
print("Model: normalized data(top4 relevant features)")
X_train_new = X_train[:,[0,1,2,6]]
X_test_new = X_test[:,[0,1,2,6]]
```

```
my_model.fit(X_train_new, Y_train)

train_metrics(X_train_new, Y_train, my_model)
test_metrics(X_test_new, Y_test, my_model)
```

```
Model: normalized data(top4 relevant features)
The model performance for training set
--------------------------------------
RMSE is 0.6913056757787523
R2 score is 0.5245321658472841


The model performance for testing set
--------------------------------------
RMSE is 0.7025733697847857
R2 score is 0.4959444812537642
```

YOUR ANSWER HERE

```
from tabulate import tabulate
evaluation_data = [["normalized data(all features)", 0.62698, 0.60889, 0.63027, 0.5943
                    ["unnormalized data(all features)", 0.77635, 0.54966, 0.78235, 0.53
                    ["normalized data(top4 relevant features)", 0.80307, 0.51813, 0.817
                    ["unnormalized data(top4 relevant features)", 0.69131, 0.52453, 0.7

col_names = ["Model", "RMSE_train", "R2_train", "RMSE_test", "R2_test"]
print(tabulate(evaluation_data, headers=col_names))
```

```
Model                                  RMSE_train    R2_train    RMSE_test
-------------------------------------  ------------  ----------  -----------
normalized data(all features)                0.62698     0.60889      0.63027
unnormalized data(all features)              0.77635     0.54966      0.78235
normalized data(top4 relevant features)      0.80307     0.51813      0.81704
unnormalized data(top4 relevant features)    0.69131     0.52453      0.70257
```

1. Difference between training between unnormalized and normalized data:

- The RMSE of training set: unnormalized data > normalized data
- The R^2 of training set: unnormalized data < normalized data
- The RMSE of testing set: unnormalized data > normalized data
- The R^2 of testing set: unnormalized data < normalized data

---

2. Difference between training on all features versus training on the top-4 most relevant features in the dataset.

> (Normalized data)

- The RMSE of training set: top4 > all

- The R^2 of training set: top4 < all
- The RMSE of testing set: top4 > all
- The R^2 of testing set: top4 < all

> (Unnormalized data)

- In all cases: top4 < all

---

3. Difference between (1) training on all features (unnormalized), (2) training on top-4 unnormalized features, and (3) training on top-4 normalized features.

- The RMSE of traning set: (3) > (1) > (2)
- The R^2 of traning set: (1) > (2) > (3)
- The RMSE of testing set: (3) > (1) > (2)
- The R^2 of testing set: (1) > (2) > (3)