assignment3

April 26, 2024

0.1 Part-1 Linear regression without scikit-learn

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_diabetes
```

```
/home/oneautumleaf/.local/lib/python3.10/site-
packages/matplotlib/projections/__init__.py:63: UserWarning: Unable to import
Axes3D. This may be due to multiple versions of Matplotlib being installed (e.g.
as a system package and as a pip package). As a result, the 3D projection is not
available.
  warnings.warn("Unable to import Axes3D. This may be due to multiple versions
of "
```

a. Load diabetes dataset from sklearn.

[ ]:
```
diabetes = load_diabetes()
X, y = diabetes.data, diabetes.target
```

[ ]:
```
X
```

[ ]:
```
array([[ 0.03807591,  0.05068012,  0.06169621, …, -0.00259226,
         0.01990749, -0.01764613],
       [-0.00188202, -0.04464164, -0.05147406, …, -0.03949338,
        -0.06833155, -0.09220405],
       [ 0.08529891,  0.05068012,  0.04445121, …, -0.00259226,
         0.00286131, -0.02593034],
       …,
       [ 0.04170844,  0.05068012, -0.01590626, …, -0.01107952,
        -0.04688253,  0.01549073],
       [-0.04547248, -0.04464164,  0.03906215, …,  0.02655962,
         0.04452873, -0.02593034],
       [-0.04547248, -0.04464164, -0.0730303 , …, -0.03949338,
        -0.00422151,  0.00306441]])
```

b. Preprocessing: Null value handling, standardization

[ ]:
```
# Normalization
X_min = X.min()
```

```
X_max = X.max()
X = (X - X_min) / (X_max - X_min)

# Standardization
X_mean = X.mean()
X_std = X.std()
X = (X - X_mean) / X_std
```

c. Data splitting: Split data as 70% train and 30% test.

[ ]:
```
# Shuffle the array
indices = np.arange(X.shape[0])
np.random.shuffle(indices)
```

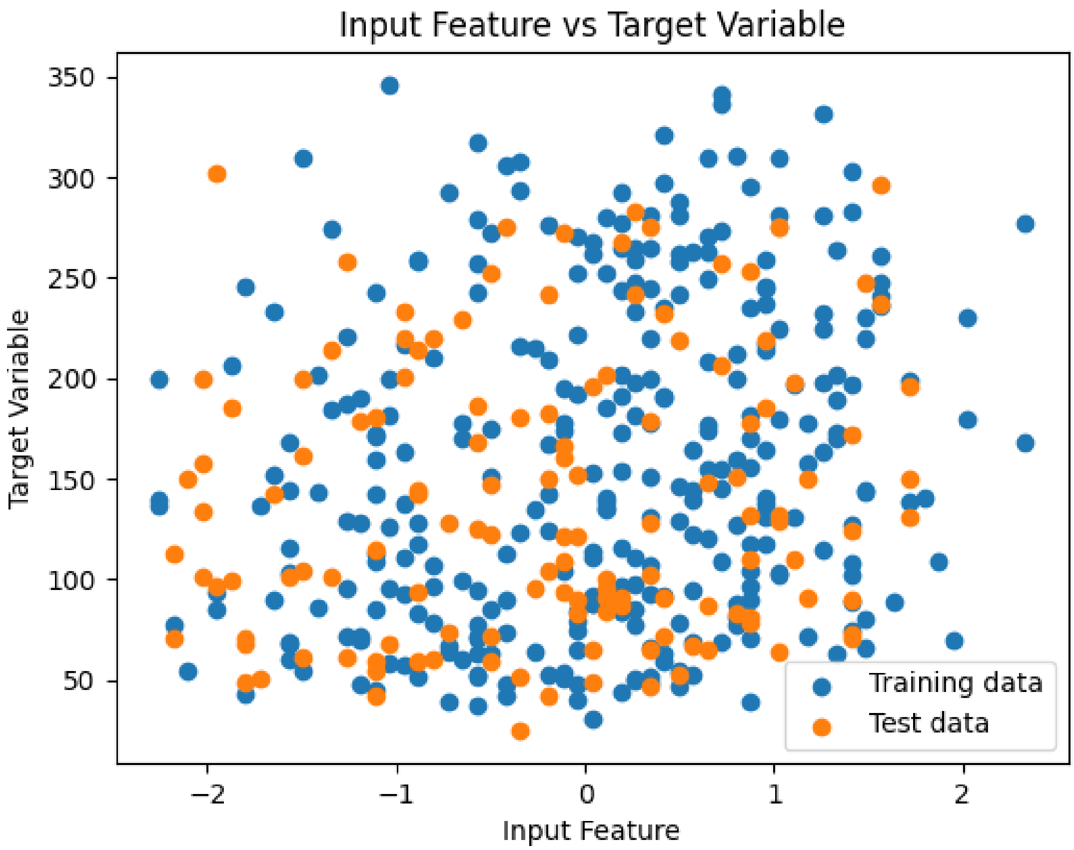
[ ]:
```
train_ratio = 0.7
test_ratio = 1 - train_ratio

train_size = int(train_ratio * X.shape[0])
train_indices = indices[:train_size]
test_indices = indices[train_size:]
```

[ ]:
```
X_train, X_test = X[train_indices], X[test_indices]
y_train, y_test = y[train_indices], y[test_indices]
```

d. Select a single input feature. Plot input feature against target variable.

[ ]:
```
feature_index = 0
X_train_single_feature = X_train[:, feature_index]
X_test_single_feature = X_test[:, feature_index]
```

[ ]:
```
plt.scatter(X_train_single_feature, y_train, label="Training data")
plt.scatter(X_test_single_feature, y_test, label="Test data")
plt.xlabel("Input Feature")
plt.ylabel("Target Variable")
plt.title("Input Feature vs Target Variable")
plt.legend()
plt.show()
```



e. Write functions for computing cost, gradients and gradient descent algorithm.

[ ]:
```
def compute_cost(X, y, theta):
    m = len(y)
    predictions = np.dot(X, theta)
    cost = (1 / (2 * m)) * np.sum(np.square(predictions - y))
    return cost

def compute_gradients(X, y, theta):
    m = len(y)
    predictions = np.dot(X, theta)
    gradients = (1 / m) * np.dot(X.T, (predictions - y))
    return gradients

def gradient_descent(X, y, theta, learning_rate, num_iterations):
    m = len(y)
    cost_history = []

    for i in range(num_iterations):
        gradients = compute_gradients(X, y, theta)
```

```
        theta -= learning_rate * gradients
        cost = compute_cost(X, y, theta)
        cost_history.append(cost)

    return theta, cost_history
```

[ ]:
```
# Add a column of ones to account for the intercept term (bias)
X_train_single_feature_with_bias = np.c_[np.ones((len(X_train_single_feature),␣
 ↪1)), X_train_single_feature]
X_test_single_feature_with_bias = np.c_[np.ones((len(X_test_single_feature),␣
 ↪1)), X_test_single_feature]
```

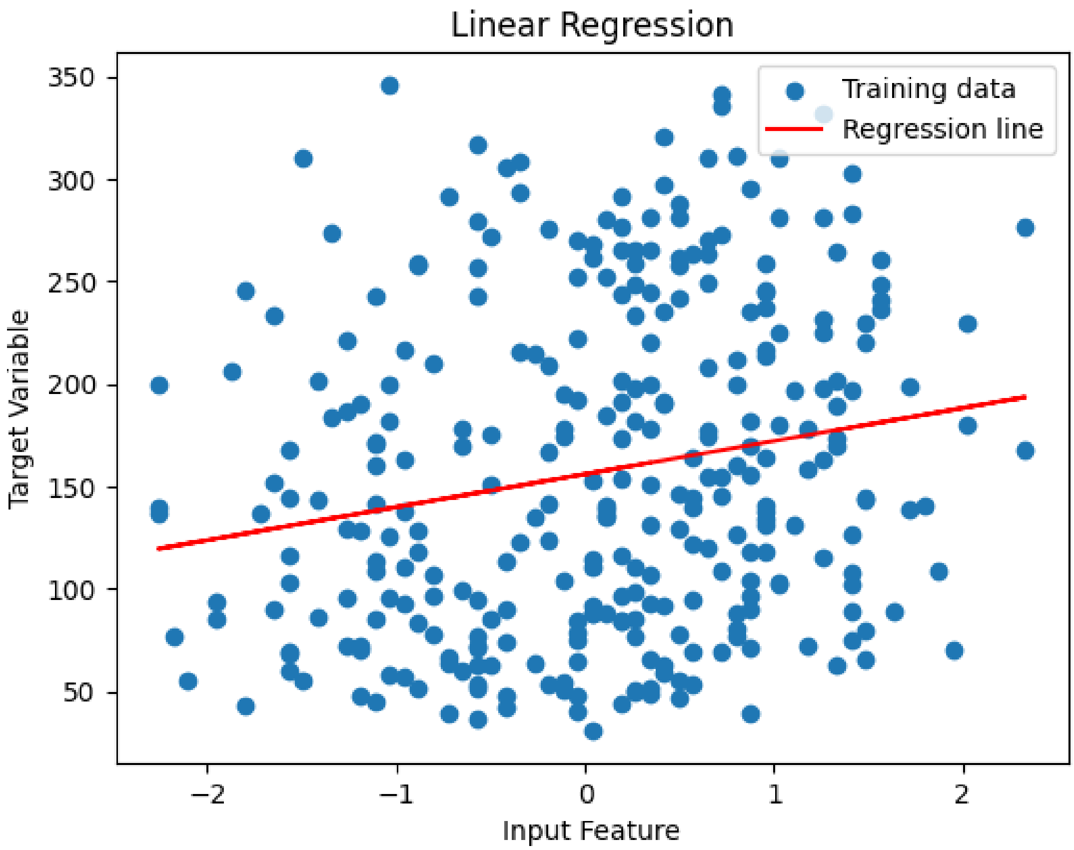
[ ]:
```
# Initialize theta (parameters)
theta = np.random.rand(X_train_single_feature_with_bias.shape[1])
```

[ ]:
```
# Set hyperparameters
learning_rate = 0.1
num_iterations = 1000
```

[ ]:
```
# Perform gradient descent
theta_final, cost_history = gradient_descent(X_train_single_feature_with_bias,␣
 ↪y_train, theta, learning_rate, num_iterations)
```

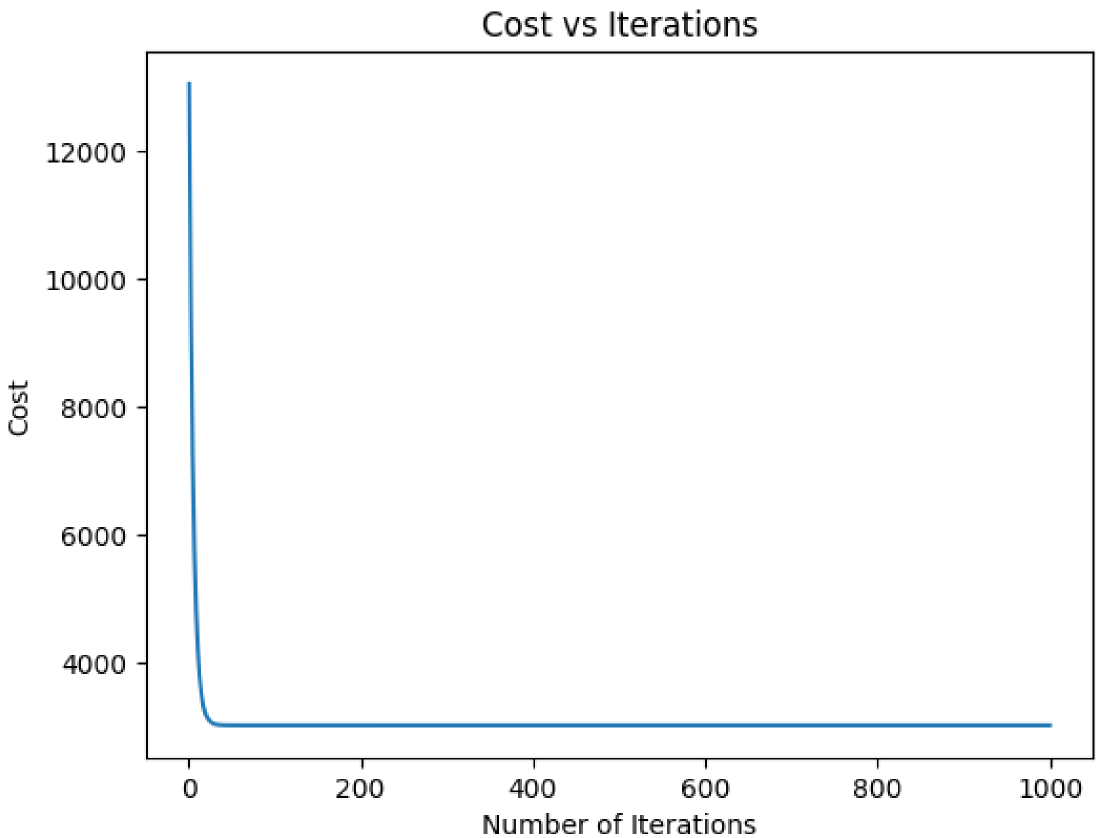
f. Plot regression line on scatter plot of feature vs target.

[ ]:
```
# Plot regression line on scatter plot
plt.scatter(X_train_single_feature, y_train, label="Training data")
plt.plot(X_train_single_feature, np.dot(X_train_single_feature_with_bias,␣
 ↪theta_final), color='red', label="Regression line")
plt.xlabel("Input Feature")
plt.ylabel("Target Variable")
plt.title("Linear Regression")
plt.legend()
plt.show()
```



g. Plot cost vs #iterations.

[ ]:
```
plt.plot(range(1, num_iterations + 1), cost_history)
plt.xlabel("Number of Iterations")
plt.ylabel("Cost")
plt.title("Cost vs Iterations")
plt.show()
```



h. Report parameter values, training error, test error and model accuracy.

[ ]:
```
# parameter values
print("Parameter values (theta):", theta_final)

# Training error (MSE)
train_predictions = np.dot(X_train_single_feature_with_bias, theta_final)
train_error = np.mean(np.square(train_predictions - y_train))
print("Training error (MSE):", train_error)

# Test error (MSE)
test_predictions = np.dot(X_test_single_feature_with_bias, theta_final)
test_error = np.mean(np.square(test_predictions - y_test))
print("Test error (MSE):", test_error)
```

```
Parameter values (theta): [155.96722889  16.12561645]
Training error (MSE): 6059.809455423056
Test error (MSE): 4990.306212719012
```

0.2 Part 2: Linear regression with scikit-learn

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

[ ]:
```
df = pd.read_excel('./datasets/Real estate valuation data set.xlsx',␣
 ↪index_col="No")
```

[ ]:
```
df.head()
```

[ ]:
```
    X1 transaction date  X2 house age  X3 distance to the nearest MRT station  \
No
1           2012.916667          32.0                                84.87882
2           2012.916667          19.5                               306.59470
3           2013.583333          13.3                               561.98450
4           2013.500000          13.3                               561.98450
5           2012.833333           5.0                               390.56840

    X4 number of convenience stores  X5 latitude  X6 longitude  \
No
1                                10     24.98298     121.54024
2                                 9     24.98034     121.53951
3                                 5     24.98746     121.54391
4                                 5     24.98746     121.54391
5                                 5     24.97937     121.54245

    Y house price of unit area
No
1                         37.9
2                         42.2
3                         47.3
4                         54.8
5                         43.1
```

[ ]:
```
df.columns
```

[ ]:
```
Index(['X1 transaction date', 'X2 house age',
       'X3 distance to the nearest MRT station',
       'X4 number of convenience stores', 'X5 latitude', 'X6 longitude',
       'Y house price of unit area'],
      dtype='object')
```

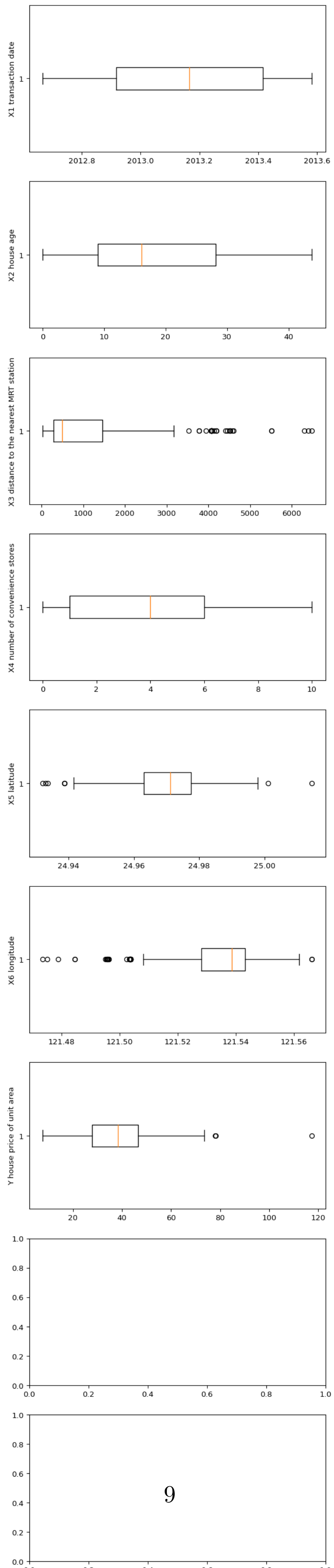
[ ]:
```
print(f"Number of null values: {df.isnull().sum()}")
```

```
Number of null values: X1 transaction date                       0
X2 house age                              0
X3 distance to the nearest MRT station    0
X4 number of convenience stores           0
```

```
X5 latitude                               0
X6 longitude                              0
Y house price of unit area                0
dtype: int64
```

[ ]:

```
# Box Plots
fig, axs = plt.subplots(9,1,dpi=95, figsize=(7,37))
i = 0
for col in df.columns:
    axs[i].boxplot(df[col], vert=False)
    axs[i].set_ylabel(col)
    i+=1
plt.show()
```


X1 transaction date
2012.8
2013.0
2013.2
2013.4
2013.6
X2 house age
0
10
20
30
40
X3 distance to the nearest MRT station
0
1000
2000
3000
4000
5000
6000
X4 number of convenience stores
0
2
4
6
8
10
X5 latitude
24.94
24.96
24.98
25.00
X6 longitude
121.48
121.50
121.52
121.54
121.56
Y house price of unit area
20
40
60
80
100
120
0.0
0.2
0.4
0.6
0.8
1.0

0.2.1 Standardization and Normalization

[ ]:
```python
from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

[ ]:
```python
standard_scaler = StandardScaler()
min_max_scaler = MinMaxScaler()
```

[ ]:
```python
# Normalize X
df_normalized = pd.DataFrame(min_max_scaler.fit_transform(df), columns=df.
 ↪columns)

# Standardize X
df_standardized = pd.DataFrame(standard_scaler.fit_transform(df_normalized),␣
 ↪columns=df.columns)

df_preprocessed = df_standardized
```

[ ]:
```python
print(f"Min:\n{df_preprocessed.min()}")
print('-------------------')
print(f"Max:\n{df_preprocessed.max()}")
print('-------------------')
print(f"Std:\n{df_preprocessed.std()}")
print('-------------------')
```

```
Min:
X1 transaction date                        -1.712334
X2 house age                               -1.556639
X3 distance to the nearest MRT station     -0.841279
X4 number of convenience stores            -1.391638
X5 latitude                                -2.981805
X6 longitude                               -3.903223
Y house price of unit area                 -2.235474
dtype: float64
------------------
Max:
X1 transaction date                         1.542244
X2 house age                                2.292652
X3 distance to the nearest MRT station      4.287008
X4 number of convenience stores             2.007407
X5 latitude                                 3.675611
X6 longitude                                2.146891
Y house price of unit area                  5.851328
dtype: float64
------------------
Std:
```

```
X1 transaction date                        1.00121
X2 house age                               1.00121
X3 distance to the nearest MRT station     1.00121
X4 number of convenience stores            1.00121
X5 latitude                                1.00121
X6 longitude                               1.00121
Y house price of unit area                 1.00121
dtype: float64
------------------
```

[ ]:
```
target_col = df.columns[6]
print(f"Target col: {target_col}")
X = df_preprocessed.drop(target_col, axis=1)
y = df_preprocessed[target_col]
```

```
Target col: Y house price of unit area
```

0.2.2 Train test split

[ ]:
```
from sklearn.model_selection import train_test_split

# Split the data into train and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[ ]:
```
print(f"X:{X.shape}\ntrain:{X_train.shape}\ntest:{X_test.shape}")
print("-----------------")
print(f"y:{y.shape}\ntrain:{y_train.shape}\ntest:{y_test.shape}")
```

```
X:(414, 6)
train:(331, 6)
test:(83, 6)
-----------------
y:(414,)
train:(331,)
test:(83,)
```

[ ]:
```
X_train.min()
```

[ ]:
```
X1 transaction date                       -1.712334
X2 house age                              -1.556639
X3 distance to the nearest MRT station    -0.841279
X4 number of convenience stores           -1.391638
X5 latitude                               -2.981805
X6 longitude                              -3.796886
dtype: float64
```

[ ]:
```
X_train
```

[ ]:
```
     X1 transaction date  X2 house age  \
192              0.062891      2.292652
234              0.358761     -0.853573
5               -1.712334     -0.932668
45              -0.232980      1.659892
245              0.950503     -0.897514
..                    …           …
71              -0.232980      1.563220
106             -0.232980     -0.045046
270              0.654632     -0.607499
348             -1.120593     -1.152376
102             -0.232980     -1.459968

     X3 distance to the nearest MRT station  X4 number of convenience stores  \
192                               -0.814143                         0.987694
234                                0.898572                        -0.032020
5                                  0.865586                        -0.371925
45                                -0.472056                         1.327598
245                               -0.352429                         0.307885
..                                      …                               …
71                                -0.351541                        -0.371925
106                               -0.709486                         1.327598
270                               -0.659459                        -1.051734
348                               -0.653844                         0.647789
102                               -0.706261                         0.647789

     X5 latitude  X6 longitude
192    -0.123441      0.478119
234    -0.722866     -1.288509
5      -0.482451     -1.358313
45      0.090352      0.755378
245     0.286395      0.964137
..          …             …
71      0.532458      0.247179
106     0.648632      0.634036
270     0.449362     -0.189259
348     0.550207      0.769730
102    -0.267851      0.491167

[331 rows x 6 columns]
```

[ ]:
```
y_train
```

[ ]:
```
192    0.347299
234   -1.036067
5     -0.432684
45     0.023532
```

```
245     0.207491
          …
71      0.207491
106     0.671065
270     5.851328
348     1.156715
102     1.208223
Name: Y house price of unit area, Length: 331, dtype: float64
```

0.2.3 Train a linear regression model

[ ]:
```python
from sklearn.linear_model import LinearRegression
```

[ ]:
```python
model = LinearRegression()
```

[ ]:
```python
model.fit(X_train, y_train)
```

[ ]:
```
LinearRegression()
```

[ ]:
```python
from sklearn.metrics import mean_squared_error

y_pred = model.predict(X_train)
test_mse = mean_squared_error(y_train, y_pred)
print(f"Train Mean Squared Error (MSE): {test_mse}")

y_pred = model.predict(X_test)
test_mse = mean_squared_error(y_test, y_pred)
print(f"Test Mean Squared Error (MSE): {test_mse}")
```

```
Train Mean Squared Error (MSE): 0.45000423774495135
Test Mean Squared Error (MSE): 0.2896878551924998
```