Name: Saurabh S. Ramteke
Roll No: 21-27-19
M.Tech: Data Science

Conclusion

- Linear regression is commonly used for predictive analysis as we have used to predict housing price (dependent variable) based on 7 features (Independent Variable).
- Linear Regression attempts to fit the model for linear equation.
- After 300 iterations we can see that the model converges with mean square error (MSE).
- R-squared signifies the goodness of fit measure for Linear Regression models. It indicates percentage of variance in dependent variable that independent variables can explain correctly.
- If dependent and independent variables are having linear relation then R squared value would be closer to 1.
- Else it would be closer to 0, and the data would not have linear relation and hence we need to change try other models for convergence.

Speed

1. Gradient descent model trains the data in **0.09 secs**

```
def model(X_train, y_train, lr, theta):

    for i in range(iterMax):

        h       = np.dot(X_train, theta)
        func    = h[:] - y_train[:, 0]
        cost    = (1/(2*m)) * np.sum(np.square(func))
        d_theta = (1/m) * np.dot(X_train.T , func)
        theta   = theta - (d_theta * lr)

        cost_list.append(cost)
        if i % 100 == 0 :
            pass
#             print(f'Cost associated with iteration no. {i}: {cost_li

    return theta, cost_list
```

```
import time
start_time = time.time()

theta, cost_list = model(X_train, y_train, lr, theta)

end_time = time.time()
print(f'The time taken by algo: {end_time - start_time} seconds')
```

```
The time taken by algo: 0.09003567695617676 seconds
```

2. Normal equation model trains the data **in 0.012 secs**

```
theta_N = np.dot(np.linalg.pinv(np.dot(X_train.T, X_train)), np.dot(X_tr
```

```
import time
start_time = time.time()

for i in range(iterMax):
    y_pred_n = np.dot(X_test, theta_N)

end_time = time.time()
print(f'The time taken by algo: {end_time - start_time} seconds')
```

```
The time taken by algo: 0.012035131454467773 seconds
```

3. Sklearn model trains the data in **0.15 secs**

```
import time
start_time = time.time()

from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)

end_time = time.time()
print(f'The time taken by algo: {end_time - start_time} seconds')
```

```
The time taken by algo: 0.15517878532409668 seconds
```

As we have done 2000 iterations in our gradient descent model still the time required to train the dataset is less than both normal equation and sci-kit learn . As by default sci-kit learn iterates only for 1000 iterations, it still takes a larger time to compute the model.

Accuracy

1. Gradient Descent

```
from sklearn.metrics import mean_squared_error, r2_score
er_GD = mean_squared_error(y_test, y_pred_GD)
r2_GD = r2_score(y_test, y_pred_GD)
print(f'Error: {er_GD}, r2_score: {r2_GD}')
```

```
Error: 0.22450722715454594, r2_score: 0.8017003406283747
```

2. Normal Equation

```
er_n= mean_squared_error(y_test, y_pred_n)
r2_n = r2_score(y_test, y_pred_n)
print(f'Error: {er_n}, r2_score: {r2_n}')
```

```
Error: 0.22453190123505146, r2_score: 0.8016785468455127
```

3. Sklearn

```
er_sk = mean_squared_error(y_test, y_pred)
r2scr_sk = r2_score(y_test, y_pred)
print(f'Error: {er_sk}, r2_score: {r2scr_sk}')
```

```
Error: 0.2245319012350511, r2_score: 0.801678546845513
```