Yasmin mohamed noureldin 40-2879

Introduction:

Firstly, I went through the previous code to visualize what can be improved. So, in the last assignment we only had a two column data so not a large data plus not enough hypothesis used so the results may not be the most accurate. In this assignment, a new data set is going to be used that has a large number of columns and rows so enough data to train and apply improvements on to get the most fitting output.

Data processing:

I start by exploring the data to see if it has classes, corrupted figures if any and which features are there. New data set is read and visualized to get number of rows, columns and the house features that I can train on.

When viewing data there was a lot of NAN values at the last rows also when running the code, the graphs appeared empty and most of the values were NAN as well, so I had to remove the empty rows using data.dropna() to have a clean dataset and get an output. The house data now has 21 features (columns number) and 17999 rows.

I saved a copy of the data to have a backup data without any edit , split the data into a Training Set(60%), a Validation Set (20%) and a Test Set (20%) using train_test_split which makes random partitions. Next, normalizing data to prepare it for machine learning process and this is a very important step to help the code run properly and Normalization by definition is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1.

My approach:

Linear regression is an algorithm used to predict values that are continuous in nature

I performed gradient descent to learn minimize the cost function J(theta), and monitor the convergence by computing the cost and later on plotting the convergence graph

```
def computeCost(X, y, theta):

    J = (np.dot(((np.dot(X, theta) - y).T), (np.dot(X, theta) - y))) / 2 * m

    return J


def gradientDescent(X, y, theta, alpha, num_iters):

    J_history = []

    for i in range(num_iters):

        theta = theta - (alpha/m)*(np.dot(X,theta.T)-y).dot(X)

        J_history.append(computeCost(X, y, theta))

    return theta, J_history
```

I tried changing alpha to different values but the output wasn't the best as some graphs were reversed so I used 0.01. As for number of iterations, changing it didn't have a noticeable effect on the output so I decreased it to 700 to decrease runtime.

For model selection:

$h\theta\ x = \theta 0 + \theta 1x$

$h\theta\ (x) = \theta 0 + \theta 1x + \theta 2x1\text{^}2$

$h\theta\ x = \theta 0 + \theta 1x + \theta 2x1\text{^}2 + \theta 3\ X2$

I defined 3 models where each model has a different degree

Gradient descent is simply used to find the values of a function's parameters that minimize a cost function as far as possible. I start by defining the initial parameter's values, and from there gradient descent uses calculus to iteratively adjust the values so they minimize the given cost-function.

Gradient descent output values from first run as each time random data is chosen and output differs:

```
theta1 computed from gradient descent:

id                 -450.218671
price            312748.716488
bedrooms         -10893.825958
bathrooms          4735.988164
sqft_living       16312.134665
sqft_lot           1619.433269
floors            -1002.043815
waterfront         9221.586376
```

```
view              4653.344219
condition         2638.869222
grade            20543.403239
sqft_above       15858.271174
sqft_basement     4354.042763
yr_built        -17407.580916
yr_renovated       205.481832
zipcode          -9364.091231
lat              14577.644345
long             -5838.797220
sqft_living15     -707.168085
sqft_lot15       -3468.726374
x0              534755.485526
Name: price, dtype: float64
```

theta2 computed from gradient descent:

```
id                -527.566334
price           313992.817706
bedrooms          1680.116585
bathrooms         -145.897074
sqft_living      10128.057728
sqft_lot          1913.351667
floors           -1267.417914
waterfront        9651.498471
view              5275.671398
condition         3506.924553
grade            27273.402976
sqft_above       10386.382727
sqft_basement     1727.010661
yr_built        -14376.862322
yr_renovated      1401.636832
zipcode          -8280.368874
lat              14456.790667
```

Yasmin mohamed noureldin 40-2879

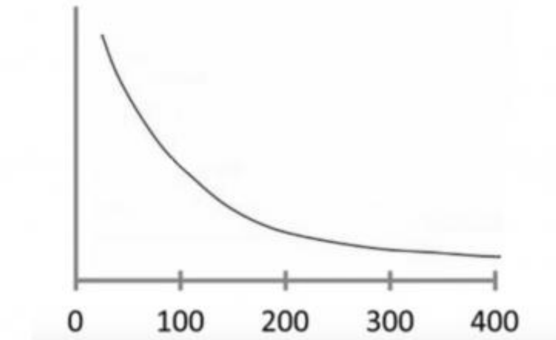
```
long                -5815.583697
sqft_living15         657.512806
sqft_lot15          -2522.057418
x0                 531101.363678
Name: price, dtype: float64
```

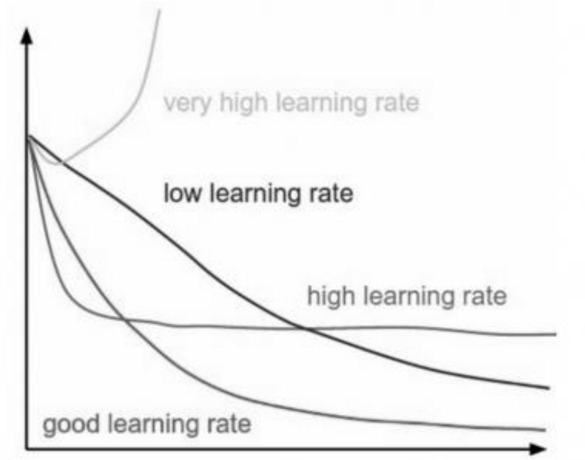
theta3 computed from gradient descent:

```
id                   -470.969591
price              314194.383508
bedrooms             1713.207678
bathrooms             106.239904
sqft_living         10014.189504
sqft_lot             -395.521772
floors              -1356.792336
waterfront           9559.778017
view                 5313.461647
condition            3456.520694
grade               27383.147252
sqft_above          10333.488148
sqft_basement        1592.122933
yr_built           -14473.761362
yr_renovated         1363.638482
zipcode             -8287.397223
lat                 14416.356731
long                -5877.061988
sqft_living15         399.498292
sqft_lot15             12.383836
x0                 530994.604082
Name: price, dtype: float64
```

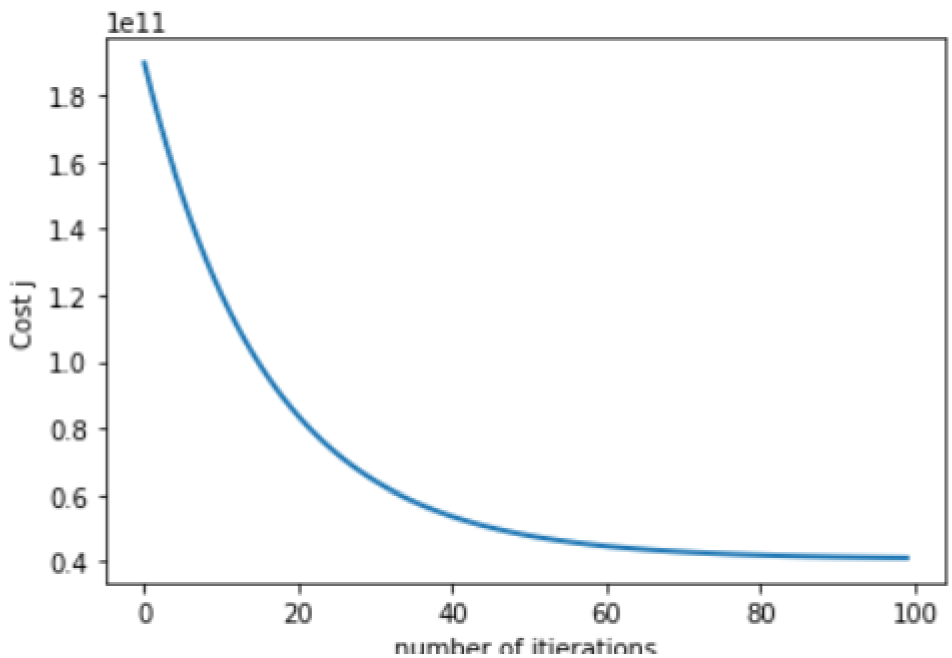
To make sure gradient descent runs properly is by plotting the cost function against the number of iterations.

Yasmin mohamed noureldin 40-2879





If gradient descent is working properly, the cost function should decrease after every iteration and if the learning rate the plot should look like the left image. In my code I tried different learning rate as 0.1,0.3,0.05,0.01 and 0.01 gave the best graph and it look exactly like the left graph here.



Yasmin mohamed noureldin 40-2879

Validation and testing

training proceeds on the training set, after which evaluation is done on the validation set:

(In data validation on thetas, Call (computecost) function, this function gives me the J cost of validating data/error)

and when the experiment seems to be successful, final evaluation can be done on the test set also by calling computecost function

output of Validation on thetas:

J1 2.011634422453688e+16

J2 1.994286460212963e+16

J3 1.9831734600179532e+16

Output of Test on thetas:

J1 8.208082287709699e+18

J2 2.9968289279477108e+16

J3 3.0297619581220904e+16