Introduction:

The dataset consists of features extracted from facebook posts. The task associated with the data is to predict how many comments a post will receive in the next H hours. The dataset is comprised of 41949 observations and 54 variables (including target variable). The main aim is to implement a linear regression model using the gradient descent algorithm with batch update. I used the sum of the squared error cost function to build the algorithm. The sum of squared error normalized by 2*number of samples [J(β 0, β 1) = (1/2m)[Σ (y⍰(i) – y(i))2] as cost and error measures, where m is the number of samples. The dataset can be found via
https://archive.ics.uci.edu/ml/datasets/Facebook+Comment+Volume+Dataset
I used Variant 1 from training and 10 test datasets for my analysis. I grouped all of them together initially and used the combined file for my analysis.

Data Exploration and Manipulation:

As I mentioned, the dataset is comprised of 41949 observations and 53 variables and 1 output variable (54th column). I started my analysis by understanding the basic statistics of each variable like their mean, median, min, max, standard deviation, missing values etc., I have noticed these observations:

- There were no missing values in the data.
- The range and magnitude of a few variables are different from the rest.
- I used Z-score > 3 as a condition to identify the outliers in features and found some. I also plotted boxplots to cross-check.

So, I decided to scale the features to normalize the data after splitting them into train and test (70/30). Predominantly I used Pandas, Numpy, Matplot, Sci-kit libraries for data preprocessing and manipulation.

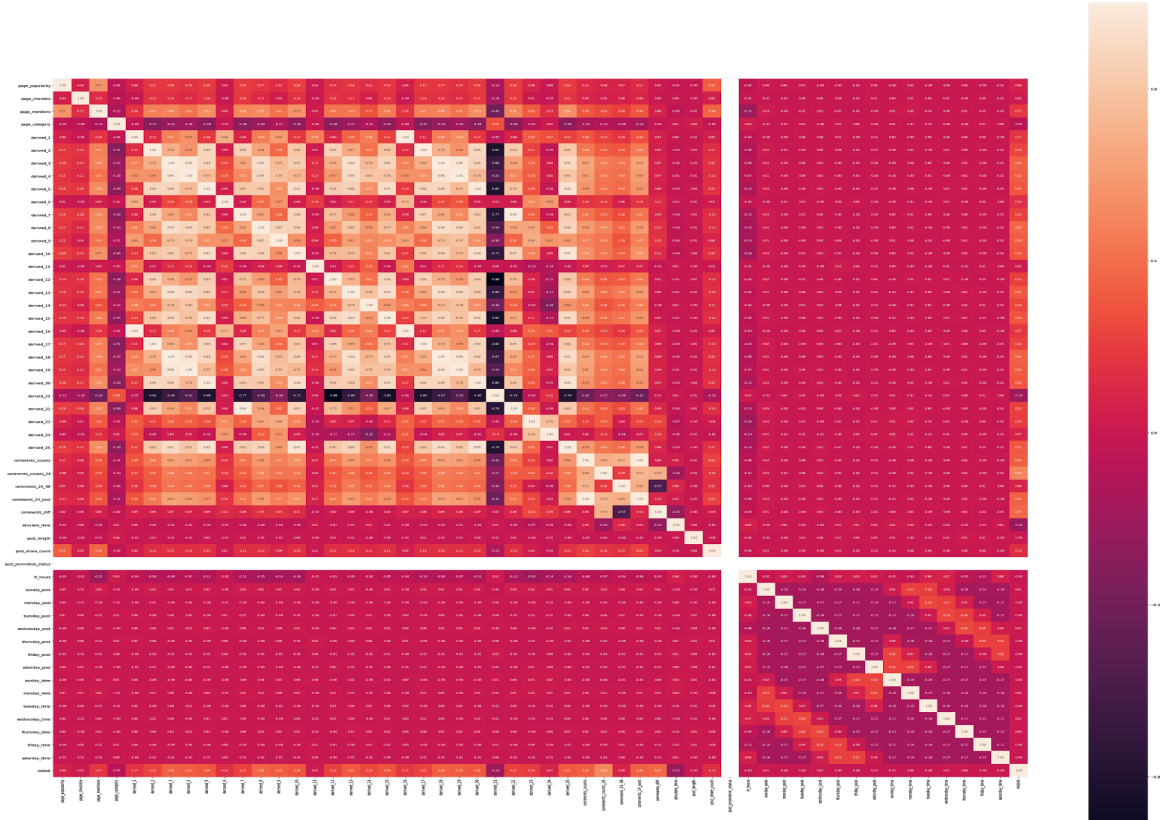
Features Selection:

Out of 53 variables, I wanted to see how each variable is correlated to the output variable and how they are correlated among themselves. I plotted correlation matrix using matplotlib to achieve this. I have included the matrix below (fig 1). From the matrix, I handpicked the ones which are correlated heavily with 'output' variable and less correlated among themselves. I was able to include 'page_mentions', 'comments_24_post', 'comments_diff', 'comments_counts_24', 'post_share_count', 'H_hours' features. I also noticed that there are a few derived variables which are heavily correlated with the output variable and also correlated among themselves. So, I picked 'derived_5', 'derived_25', 'derived_13', 'derived_18', 'derived_8' which looked like good ones to include. Thus, I was able to select 11 variables for further analysis and created a new data frame with only selected features. I plotted a correlation matrix with the selected variables as well (fig 2).

Splitting into Training & Test sets:

I split the new data frame into training and test randomly in the 70:30 ratio using pandas lib. The dimensions of the files are as below:

X_train = (29364, 11)
X_test = (12585, 11)
Y_train = (29364, 1)
Y_ test = (12585, 1)



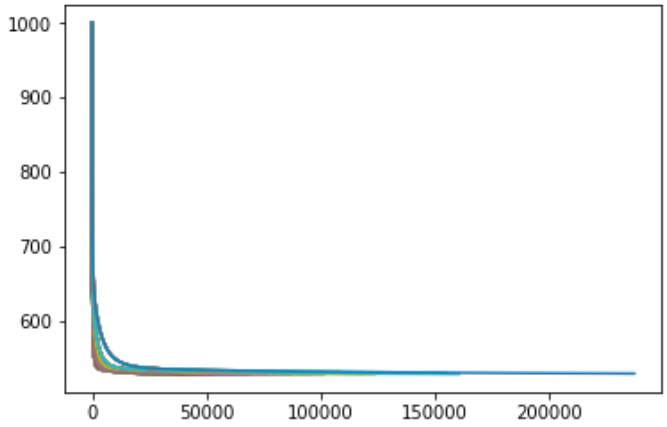

page_mentions 1.00 0.53 0.54 0.43 0.49 0.52 0.30 0.06 0.25 0.36 -0.21 0.17
derived_5 0.53 1.00 0.95 0.86 0.89 0.82 0.57 0.06 0.41 0.15 -0.12 0.30
derived_25 0.54 0.95 1.00 0.86 0.92 0.88 0.58 0.08 0.44 0.15 -0.14 0.31
derived_13 0.43 0.86 0.86 1.00 0.94 0.80 0.60 0.00 0.40 0.14 -0.05 0.31
derived_18 0.49 0.89 0.92 0.94 1.00 0.94 0.63 0.08 0.46 0.15 -0.09 0.34
derived_8 0.52 0.82 0.88 0.80 0.94 1.00 0.59 0.15 0.49 0.13 -0.15 0.35
comments_24_post 0.30 0.57 0.58 0.60 0.63 0.59 1.00 0.10 0.72 0.22 -0.06 0.34
comments_diff 0.06 0.06 0.08 0.00 0.08 0.15 0.10 1.00 0.70 0.03 -0.03 0.37
comments_counts_24 0.25 0.41 0.44 0.40 0.46 0.49 0.72 0.70 1.00 0.16 -0.07 0.50
post_share_count 0.36 0.15 0.15 0.14 0.15 0.13 0.22 0.03 0.16 1.00 -0.00 0.13
H_hours -0.21 -0.12 -0.14 -0.05 -0.09 -0.15 -0.06 -0.03 -0.07 -0.00 1.00 -0.03
output 0.17 0.30 0.31 0.31 0.34 0.35 0.34 0.37 0.50 0.13 -0.03 1.00
page_mentions
derived_5
derived_25
derived_13
derived_18
derived_8
comments_24_post
comments_diff
comments_counts_24
post_share_count
H_hours
output
1.00
0.75
0.50
0.25
0.00

Feature scaling and Preprocessing:

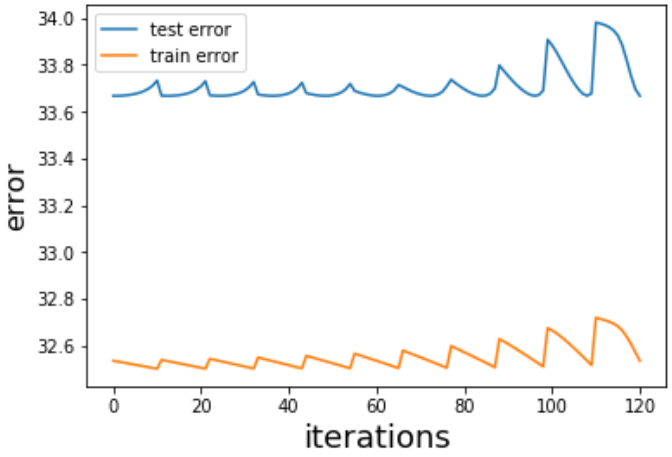
Then I used training and test feature sets and normalized them using min.max formula:
data[i] = (data[i]-data[i].mean())/(data[i].max() - data[i].min())
After I normalized, I added a column of ones to X_train and X_test for matrix multiplications. Since there are 11 features but 12 betas, we need to add another column to make it compatible for matrix multiplications with beta matrix. And I converted y_train and y_test into an array for further analysis.

Building the model and initial experimentation:

I built the gradient descent model by using matrix multiplication of beta and x transpose to calculate predicted y. Then I defined a cost function for sum of mean squared errors and used gradient method to modify the beta values by varying the learning rates. Initially, I trained the model on training set of 11 features using various learning rates (0.1 to 10.0) and threshold values (0.000001 to 0.0001) to monitor the behavior of model. I was able to define a limited range of learning rate and threshold values where the cost function is converging local minima. I considered RMSE values as the error for train and test sets to pick the best learning rate (alpha). In the below plot, x-axis is number iterations and y-axis is cost function value:

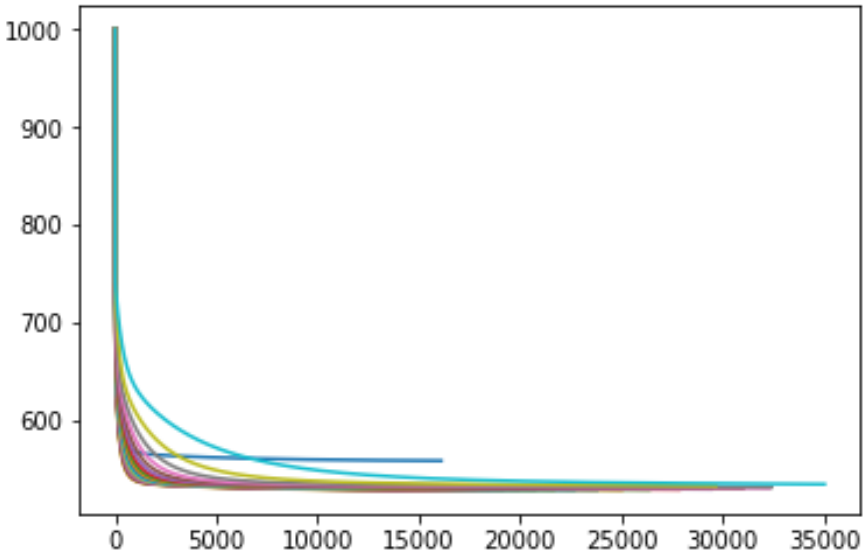


The below graph represents a total of 121 combinations (11 different alpha values with 11 threshold values) to monitor the cost function. For each alpha value, I changed threshold values, that's the reason why the graph looks like a step function.



Experimentation 1:

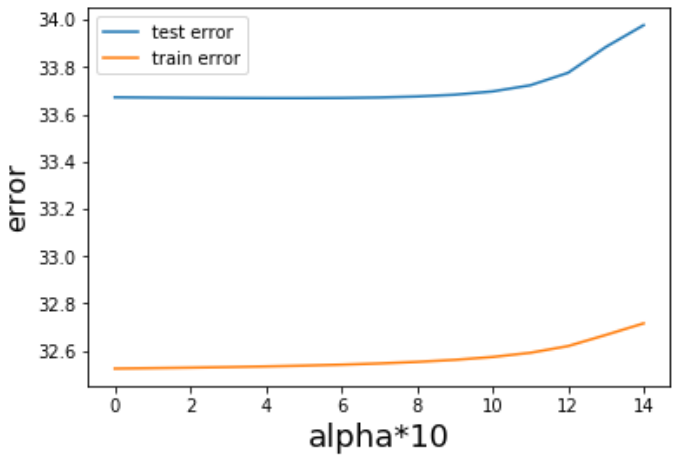
After doing the experiment, I noticed that error is minimum across train and test at threshold = 0.00001. In order to monitor the behavior change in cost function, I plotted a graph with varying alpha values from 2.0 to 0.1 with a decrement of -0.1, helding the constant threshold of 0.00001. The initial parameter values I started is beta values of zero. Below screenshot is error plot of train and test sets in a single graph for above experiment. Cost functions for different alpha values w.r.t iterations:



As the learning rate decreases, the error also decreased. For both the training and test sets, the error is minimum at alpha = 1.1. As we are trying to minimize the error using the cost function, since error is minimum at alpha is 1.1, I picked it as the best learning rate. Between test and train errors, I prefer the alpha value where test error is minimum because we can only validate the accuracy of our model when we run it with test data.

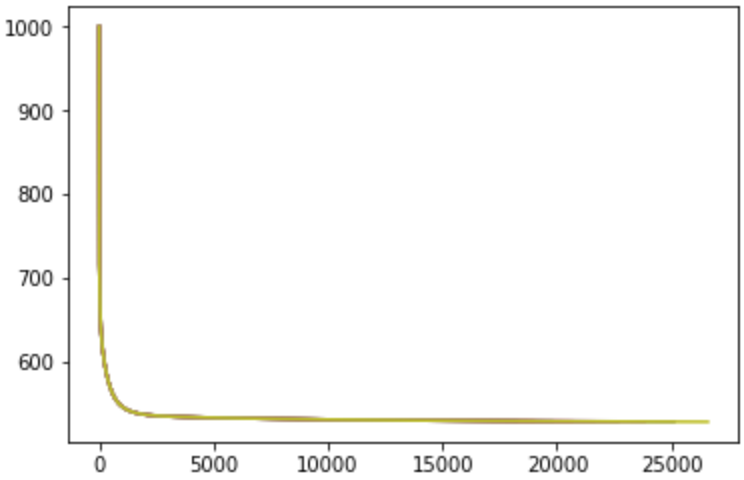
At learning rate (alpha) = 1.1,
RMSE of test data = 33.668149559972115
RMSE of train data = 32.53313680985653
Beta values = array([7.80322844, -27.31813434, 48.49325765, -71.3514097 , 40.12334852, 22.76656885, 187.05183831, -53.76901983, 127.81187241, 546.34702377, 179.88680516, 3.98664829])

Please notice the below graph. (**Note:** The X-axis is not alpha*10, it is the number of iterations)



Experimentation 2:

I already calculated train and test errors with respect to various thresholds w.r.t various alpha values initially. In the previous question, I hold threshold constant as 0.00001 depending on the overall behavior of the cost function. Likewise, I picked the best alpha value i.e., 1.1 to test various thresholds in this case. The below graph is an output when I tried to change the threshold from 0.1 to 0.00001 by the decrement of 0.001 to identify the range of threshold where the error is decreasing. From the graph, I notice that the best threshold lies between 0.0001 - 0.00001.



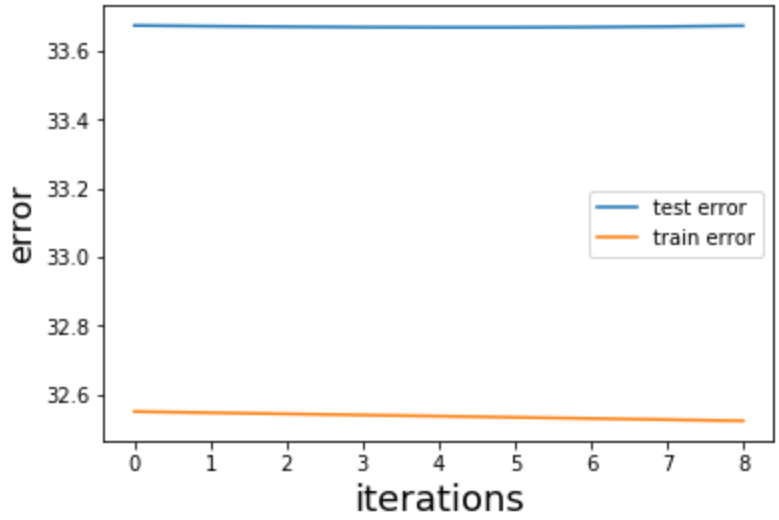
Below graph represents the variation of train and test errors by changing the threshold value between 0.00015 to 0.00005 with a decrement of 0.00001. When the error is minimum at threshold = 0.00001,

RMSE of test data = 33.668149559972115

RMSE of train data = 32.53313680985653

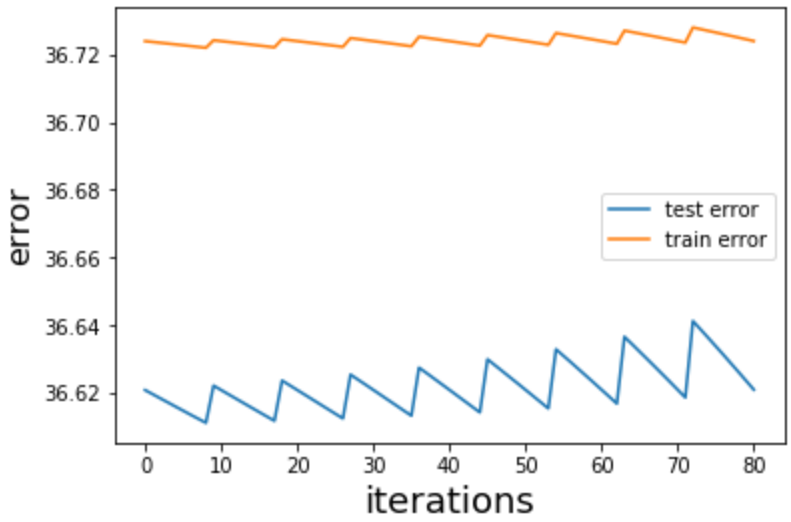
Alpha = 1.1

Beta values = array([7.80322844, -28.47124104, 48.86134667, -71.9046171 , 40.81440464, 20.82090589, 190.65409605, -56.6619778 , 122.13007199, 552.74030028, 184.94534109, 3.99276499])



Experimentation 3:

For experimentation 3, I randomly chose 'page_mentions', 'derived_5', 'derived_25', 'derived_13', 'H_hours' features and calculated train and test error. It seemed like the randomly selected features were not efficient in predicting the number of Facebook comments. I ran the gradient descent model on the new random data and with the increase in the alpha values also increased error value. The model converged at alpha = 1.5 and threshold of 0.00007.



The error was minimum at alpha = 1.5

Threshold = 0.00007

RMSE of test = 36.61128685764466

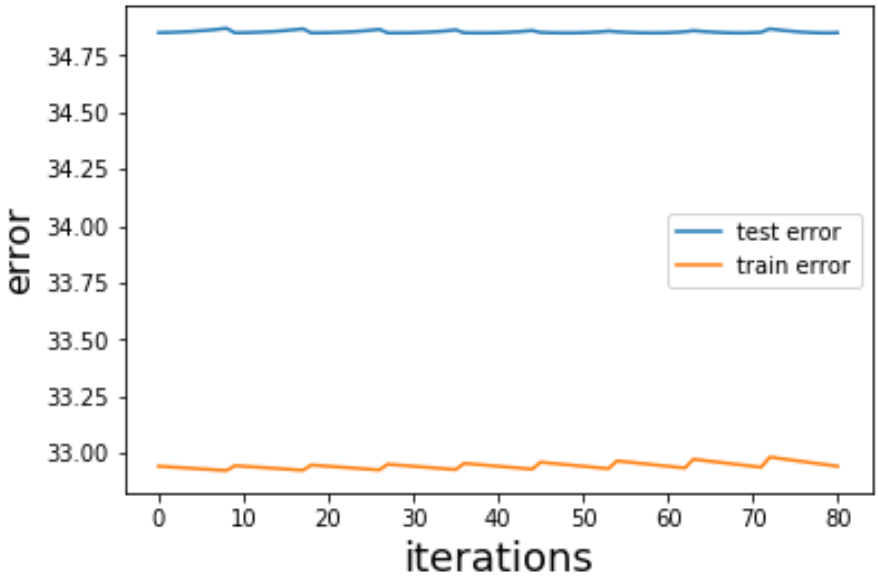
RMSE of train = 36.721915629187684

Beta = array([7.80322844, 13.28107212, 43.98939924, 77.22845312, 82.61821955, 3.86384727])

The RMSE error values of both test and train indicate that they are way less compared to the original model. The reason behind this could be that some of the best features which are crucial to predict facebook comments were not included, like comments in 24 hours, comments counts etc., And we are not aware of the features included to create derived features.

Experimentation 4:

For experimentation 4, I chose my personal best features which I believe would impact the facebook comments more compared to the rest. They are 'comments_24_post', 'comments_diff', 'comments_counts_24', 'post_share_count', 'H_hours'. It seemed like the top selected features were not efficient in predicting the number of Facebook comments compared to the original features set either.. I ran the gradient descent model on the new data and with various alpha (from 1.5 to 0.5 with decrement of 0.1) and threshold values (from 0.00002 to 0.000001 with a decrement of 0.00001). From the observations after running the model, the decrease in the alpha values doesn't decrease the error by large amount. The model converged at alpha = 1.2 and the threshold of 0.00004.



The error was minimum at alpha = 1.2
Threshold = 0.00004
RMSE of test = 34.84968365410827
RMSE of train = 32.921506502041076
Beta = array([7.80322844e+00, 4.09550429e+01, 1.58830642e+02, 5.23272664e+02, 1.46607309e+02, 3.72353020e-01])

The RMSE error values of both test and train indicate that they are less compared to the original model. The reason behind this might be that none of the dereived features were included in the model. I didn't include them because I am unaware of the qualities of dereived features and they are highly correlated among themselves. So, I excluded them to avoid collinearity. But, I was successful in achieving better results compared to randomly picked features. Maybe including one derived features might have improved this model abit pulling it closer to the original values.

Discussion:

The final linear regression equation is
Y = 7.80322844 -27.31813434 * page_mentions + 48.49325765 * derived_5 - 71.3514097 * derived_25 + 40.12334852 * derived_13 + 22.76656885 * derived_18 + 187.05183831 * derived_8 - 53.76901983 * comments_24_post + 127.81187241 * comments_diff + 546.34702377 * comments_counts_24 + 179.88680516 * post_share_count + 3.98664829 * H_hours

At learning rate (alpha) = 1.1
Threshold = 0.00001
RMSE of test data = 33.668149559972115
RMSE of train data = 32.53313680985653

By training the model on datasets with various features, I realized that the model best behaved when I included more features (more derived features). I was skeptical to use derived features in my model given that they are heavily correlated among themselves. But since they are derived from other significant variables, I realised that they are also important in predicting the facebook comments. Other features which were crucial in predicting facebook comments were CC1, CC2, CC3, CC4.

After realising the significance of derived features, I would have included more derived variables in my model to improve it. From the above linear regression equation, the parameter for page_mentions is a negative, which is theoritically wrong. Because with more page_mentions, comes more page views, which increases the visibility of that facebook page increasing the chances of attracting more facebook comments. And among the 5 derived features I have in my model, one of them has negative paramater. All derived features has strong impact on predicting the number of facebook comments. So, I would definitely include more derived features to make my model more robust.

One more thing I would have done to improve my model is that I would have removed the outliers before scaling the features to come up a more efficient model. Removing the outliers leads to decreasing the error by huge margin, thus making my model more robust.