**Entire Data Set Formula aka best model:**

**Random Features Formula:**

**My Features Formula:**

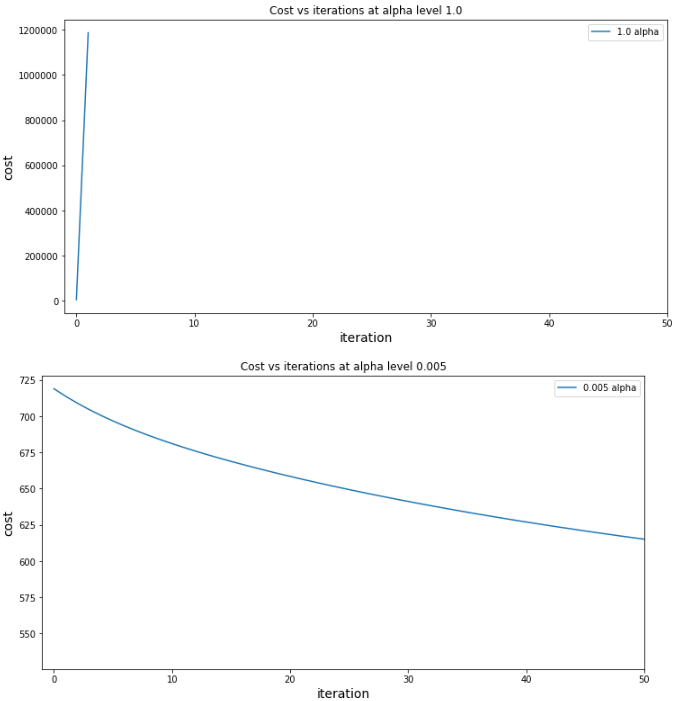
**Initial Parameter values:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Initial Parameter values (seed set to 20190201)**  These numbers will could change for every experiment. We did reset the seed for every experiment to ensure consistent beta values. | | | | | | |
| **Index** | **0** | **1** | **2** | **3** | **4** | **5** |
| Beta\_values | 0.8080 | 0.3905 | 0.3906 | 0.8397 | 0.5692 | 0.2220 |
| **Index** | **6** | **7** | **8** | **9** | **10** | **11** |
| Beta\_values | 0.8738 | 0.1919 | 0.6967 | 0.3981 | 0.0632 | 0.9623 |
| **Index** | **12** | **13** | **14** | **15** | **16** | **17** |
| Beta\_values | 0.4927 | 0.0687 | 0.9054 | 0.3135 | 0.7615 | 0.8265 |
| **Index** | **18** | **19** | **20** | **21** | **22** | **23** |
| Beta\_values | 0.2708 | 0.8867 | 0.1699 | 0.9477 | 0.9136 | 0.8511 |
| **Index** | **24** | **25** | **26** | **27** | **28** | **29** |
| Beta\_values | 0.3391 | 0.6048 | 0.0483 | 0.8954 | 0.9005 | 0.1310 |
| **Index** | **30** | **31** | **32** | **33** | **34** | **35** |
| Beta\_values | 0.6318 | 0.3021 | 0.4458 | 0.3587 | 0.5386 | 0.5553 |
| **Index** | **36** | **37** | **38** | **39** | **40** | **41** |
| Beta\_values | 0.4938 | 0.7637 | 0.2466 | 0.0850 | 0.0093 | 0.7333 |
| **Index** | **42** | **43** | **44** | **45** | **46** | **47** |
| Beta\_values | 0.3523 | 0.9987 | 0.9455 | 0.4863 | 0.4516 | 0.6837 |
| **Index** | **48** | **49** | **50** | **51** | **52** |  |
| Beta\_values | 0.4185 | 0.9485 | 0.0051 | 0.9401 | 0.0003 |  |

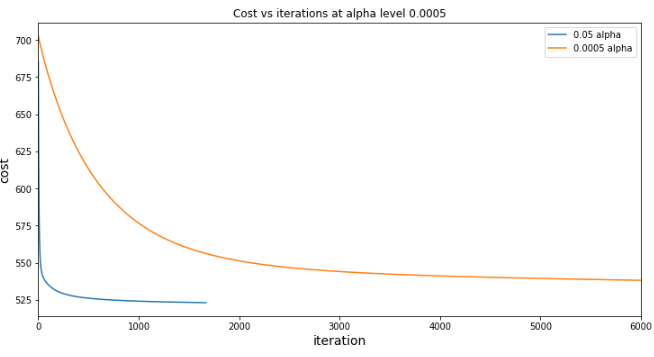
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Errors by Experiment Type** | | | | |
| **EXP** | **Split** | **My\_Features** | **Rand\_Features** | **Full\_Features** |
| EXP 1 | **Training** |  |  | 522.27 |
| **Testing** |  |  | 630.86 |
| EXP 2 | **Training** |  |  | 522.88 |
| **Testing** |  |  | 632.12 |
| EXP 3 | **Training** |  | 659.82 | 525.44 |
| **Testing** |  | 783.53 | 641 |
| EXP 4 | **Training** | 573.69 | 659.82 | 525.44 |
| **Testing** | 731.31 | 783.53 | 641 |

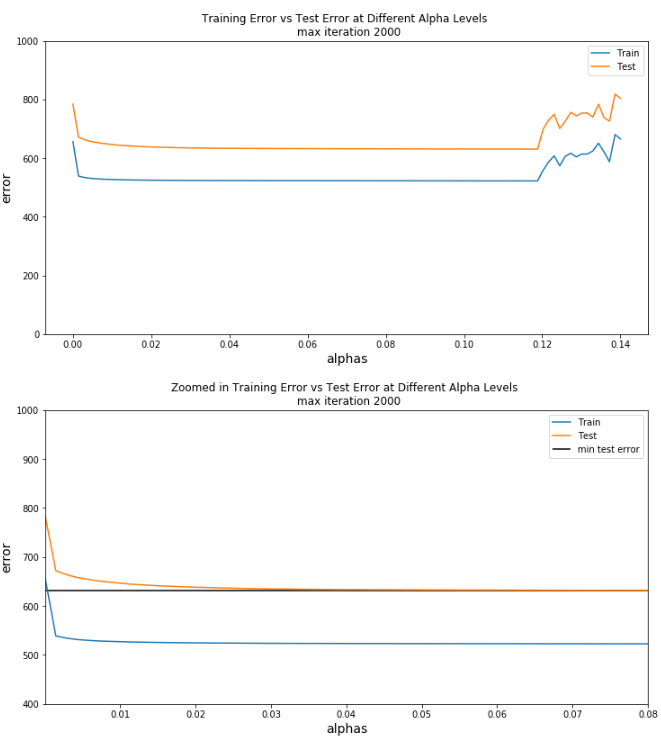
1. **Experiment with various values of learning rate ∝ and report on your findings as how the error varies for train and test sets with varying ∝. Plot the results. Report your best ∝ and why you picked it.**

After experimenting with a few hundred alpha levels I put a couple plots here that I thought would be interesting to discuss.

In the first two plots to the right, you will see the cost function with respect to the alpha levels are completely different. In the top plot you will see the alpha rapidly begin to explode in cost and stopping after iteration 2. This stopping after iteration 2 is by design. We have programmed our algorithm to stop if the cost begins to increase by a substantial rate. This is fine for a convex function but for non-convex functions we would want to remove this.

The reason our cost increases is due to the learning rate being too large for our model. The result of this causes our beta to bounce around and overshoot the global minimum. Ultimately, it won’t reach the minimum and will continue to increase in cost. I struggled with this as I wasn’t initially sure why it was happening. First, I thought my gradient descent algorithm was incorrect. I initially created a separate pseudo-data set to vet this out. This dataset allowed me to create my own multivariate regression equation so I knew what my beta coefficients should be. Once I verified that all was working correctly, I knew it was then turning my attention to the gradient being too big. After experimenting and going back over the lecture notes it was clear a large learning rate was the issue.

The third plot really does the best to summarize this experiment. As we decrease the learning rate the amount of iterations for the betas to converges increases. At each iteration, the cost continues to decrease until we reach our minimum. A larger learning rate will reach the minimum faster, but we must be careful that we don’t overshoot our minimum and start bouncing around on the betas’ gradient. Smaller learning rates need more time because they are taking smaller steps towards the bottom of the convex function.

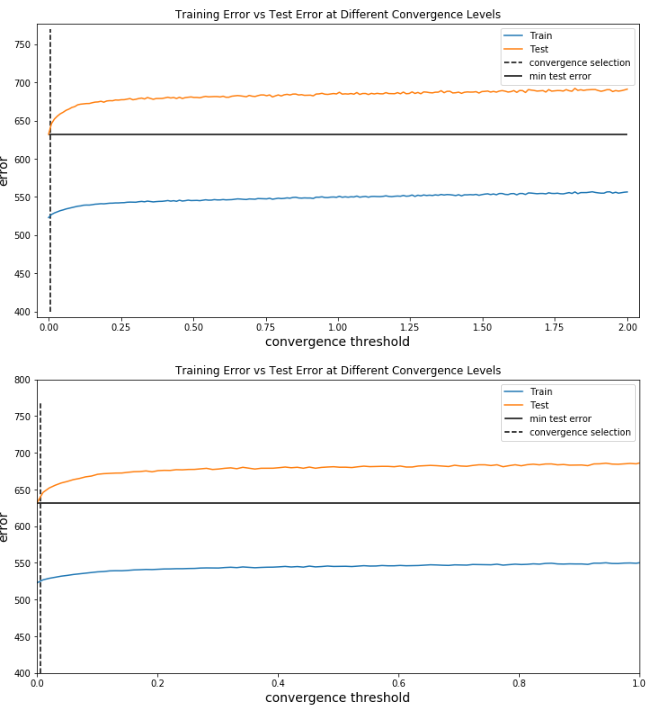
The fourth chart (right) represents the effect of different alpha levels on our training set and testing set. We know that the smaller and smaller our alpha level becomes (moving right to left on the chart) the more and more iterations you’ll need before it converges. At the same time, these smaller steps get closer and closer to the actual global minimum. We can see this displayed in chart 4. Also, we can see, based on our iterations and convergence, and alpha level between .12 and .02 produce similar train/test errors. We balance a fine line with choosing alphas. We don’t want to choose one that’s so big that could overshoot the minimum. At the same time, we don’t want to choose one that’s too small and takes too much time to converge.

What’s interesting on chart 4 & 5 (right), as our gradient gets super small our cost begins to increase. My hypothesis on why this happens, my learning rate is too small for my threshold parameter. If I made the threshold parameter even smaller this cost number should stay at its remaining level.

As a future machine learning engineer/data scientist we must weigh the time it takes for getting an extremely precise beta estimate with getting something that’s “pretty close” to the global minimum**. In this scenario, I will choose .05** as my alpha level. While I could have chosen something between .12 and .02, .05 I feel allows me a good balance. This allows me to balance the line of obtaining a learning rate which takes small steps towards our global minimum at the same time won’t take much time to converge.

1. **Experiment with various thresholds for convergence. Plot error results for train and test sets as a function of threshold and describe how varying the threshold affects error. Pick your best threshold and plot train and test error (in one figure) as a function of number of gradient descent iterations.**

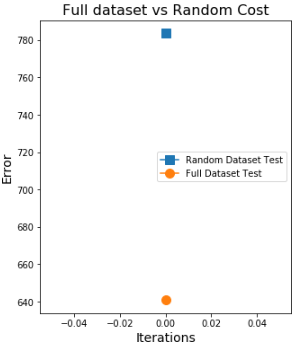
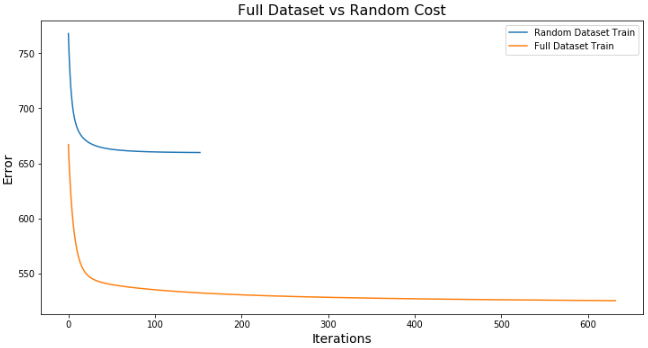
I found out that the smaller the convergence threshold the smaller the cost on the testing set will be, up to a certain point.

The first chart to the right is an extreme example but provides a good illustration. I wanted to show that at higher levels of convergence the higher your cost will be. This is driven by the fact that there’s still optimization to be had. What’s interesting is we can see the slope of errors for both train and test is continues to decrease up until we get to a threshold of 2. At this point we can really see it begin to level out. This affect is barley visible in our first plot but more representative in our second plot.

The second and third chart were instrumental in finding our ideal convergence threshold. We tested 200 convergence thresholds between 2 and 10e-7. **We eventually settled on .005 for a couple of reasons**. As we move lower and lower on convergence, we see the slope of the testing errors flat lines then begins to increase. This shows us, we are beginning to near the bottom of our convex function. This implies, there isn’t much to be gained by decreasing the convergence any further. It almost looks like if you had a mirror image of the plot you would get the full convex curve. If we were to decrease it further, we would be waiting longer to get estimates which would produce results equivalent to this level. One thing to keep in mind, we as future data scientist/machine learning engineers must balance both time and resources. Time here is how long it takes to converge, and resources is both computational resources and other system resources.

1. **Pick five features randomly and retrain your model only on these five features. Compare train and test error results for the case of using your original set of features (greater than 10) and five random features. Report which five features did you select randomly.**

I randomly selected 'num\_comm\_pre', 'base\_mon', 'l24\_avg', 'l24\_48\_avg', 'l24\_48\_std' as my variables. Using the entire data set appears to have both a lower training cost and a lower testing cost.

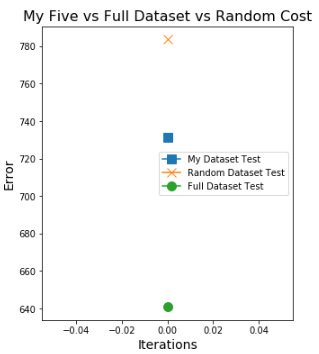
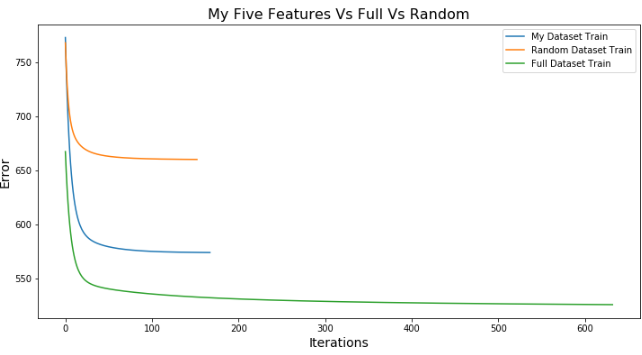
1. **Now pick five features that you think are best suited to predict the output, and retrain your model using these five features. Compare to the case of using your original set of features and to random features case. Did your choice of features provide better results than picking random features? Why? Did your choice of features provide better results than using all features? Why?**

I choose the following five features, ['num\_comm\_diff\_24\_48', 'num\_comm\_l24', 'h\_hours', 'pst\_shre\_cnt', 'page\_talk\_bt']. These were choose given some quick exploratory analysis and after consulting the research paper associated with this dataset. After reading the research paper, what stood out to me was looking at the shape of comments by hour on a post. After reading this, I performed a quick correlation analysis to see what was correlated with our target variable. After looking at the highest correlated variable I started creating hypothesis. I knew that comments in the next H hours were somehow related to most recent base hours. I also wanted a way to capture comment momentum. We logically know every post is not created equal. Some posts become viral and others got lost in the noise of all the crap on Facebook. Here was the thought process behind selecting these variables:

* num\_comm\_diff\_24\_48 – this would give us an indication of comment momentum. Were the comments picking up or slowing down?
* num\_comm\_l24 – the previous 24 hours should be an indicator for how much we will receive in the next H hours
* h\_hours – this is related to our target variable as we need to know what number in the future, we will need to predict
* pst\_shre\_cnt – the more the post is being shared the higher the likelihood that it will receive more comments as its seen by more people. Th inverse of this is true as well
* page\_talk\_bt– the more people are talking about this page the higher the likelihood that it will receive more comments as more people are talking about it. The inverse of this is true as well

My choice features did seem to perform pretty well but not as well as using all the features. The shape of my error curve mirrors that of the full dataset. This means to tell me that I’m picking up most of the variation in the dataset by just a couple of variables.

My choice of features also performing better than random. This is due to looking at the data and gathering some insights about the data rather than picking variables randomly. There’s always the chance that your random choice can gather good results. As an example, if I choose 5 variables, trained the model and applied it to the test set and repeated this process 1000 times, there’s a chance the random will do perform better than mine. We always want to ensure our model performs better than randomly selecting variables. If not, we may be out of a job!



**Discussion**

**Describe your interpretation of the results.**

See each experiments page as this is covered on those pages.

**What do you think matters the most for predicting the number of comments?**

I think the 5 features I picked mattered the most to it. If we look at the difference in errors between the full dataset and the ones I picked, they are fairly close to each other. My error is only 14% more than the full dataset and I’m using 48 less parameters. I think this just goes to show us how important feature engineering and feature selection is.

**What other steps could have been taken with regards to modeling to get better results?**

Polynomials:

* I could have played around with adding polynomials to my top 5 features. This may have brought down the error even more.

Interaction effects:

* The one that really comes to mind is the interaction between the post share count and the number of comments in the last 24 hours. The thinking here is trying to account for viral posts. I don’t believe the relationship between post share and number of comments in the last 24 has a linear relationship. The reason being, as more and more people share the post the more and more opportunity there is for the post to “gain steam” and start exhibiting a quadratic relationship.

Regularization:

* The reason being we would have penalized coefficients that don’t contribute much to the function. I think this is important to selecting features.

Testing different models:

* As an example, we could have tried this with a decision tree. This would allow us to see which model does a better job in picking up the true target function.