CS168 Project 3

Henry Lin, Kaylee Bement

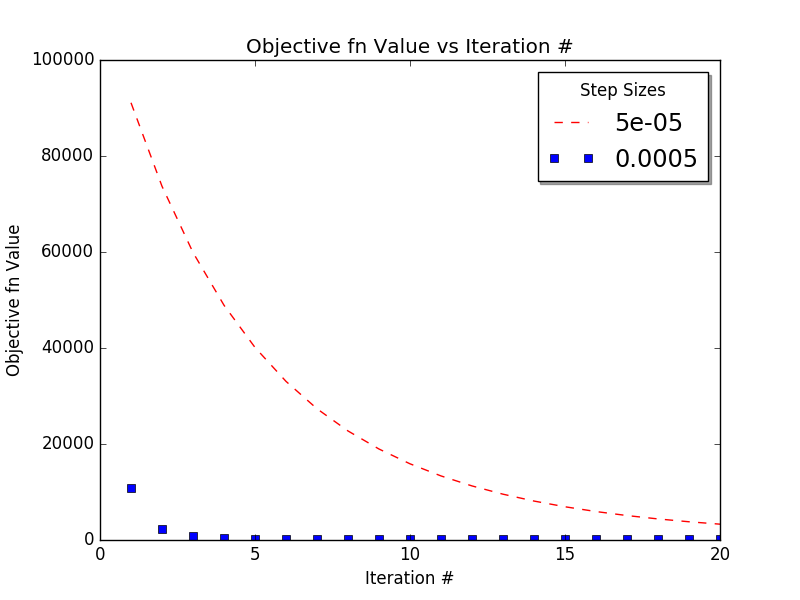
1.

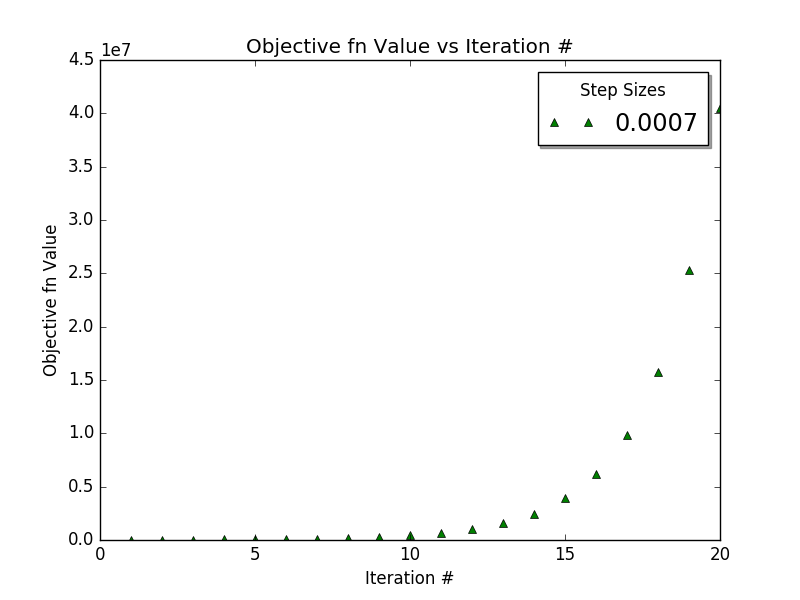
A. Objective function value: 220.34264027

Objective function value with a consisting of all 0s: 91827.65959497

Since there is randomness in some of the function values, these numbers fluctuate, but are close to the numbers given.

B. See code.py





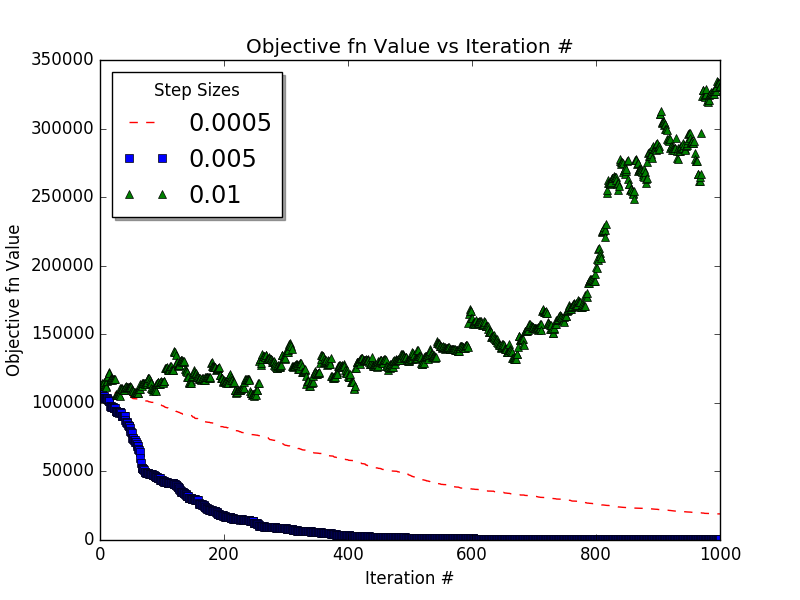
Gradient of f at at:

Optimal step size: 0.0005 with final objective function value of 233.09586745

The step size has a great impact on the convergence of gradient descent. If the step size is too small, gradient descent might be heading in the right direction, but never hit convergence, such as with step size = 0.00005, or it might find a local minimum and converge at the wrong value. With a small step size, it is also very easy to get stuck in a plateau without a large enough update to escape the plateau If the step size is too large, gradient descent might overstep the minimum and step towards an incorrect local minimum, such as with step size 0.0007. A common problem with large step sizes is oscillating around the minimum due to too large of updates. However, step size 0.0005 seemed to work well, and gave us an objective function value close to the one in part A.

C. See code.py

Optimal step size: 0.005 with final objective function value of 499.53798953



Step size influences the convergence of stochastic gradient descent in the same way it affects regular gradient descent, but it takes more iterations for SGD to converge. Regular gradient descent has a better final value than SGD, but in SGD, each data point is only used once on average since there are 1000 iterations to choose a random point and there are 1000 data points, while in regular gradient descent, each data point is used 20 times since every point is used in each iteration.

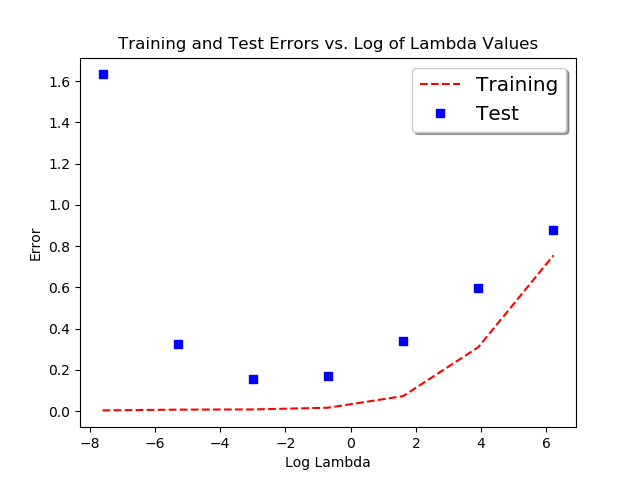
2.

A.

Avg train err: 1.08482770211125681e-14

Avg test err: 0.5823632339833625

B.



Based on the graph, we can see that different values of lambda will drastically affect the test error reported. From part A, we can see a drastic decrease in test error once we implement regularization at the cost of slightly higher training error. This tradeoff is necessary if we want the model to be able to generalize to an unknown dataset however. Not all values of lambda work however, so it is clear that lambda is a hyperparameter that needs to be tuned to have the optimal results. There seems to be an ideal around 0.005, 0.05 where the test error is at its lowest.

C.

Step size: 5e-05

Average Training Error: 0.016675896681903734

Average Test Error: 0.1562577003529706

Step size: 0.0005

Average Training Error: 0.004554721052220292

Average Test Error: 0.2173294324894938

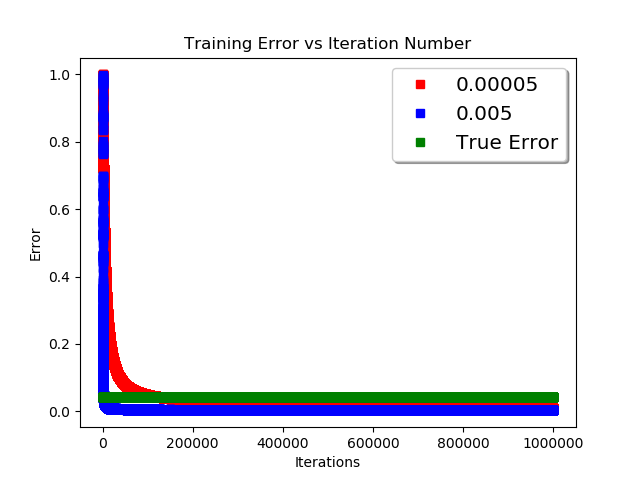
Step size: 0.005

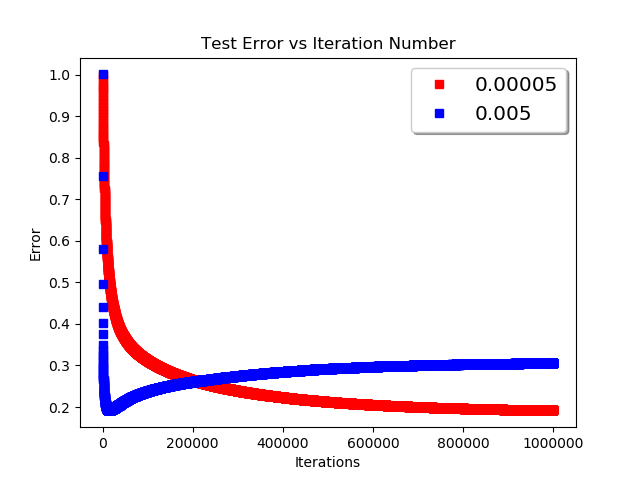
Average Training Error: 0.00036890858831097083

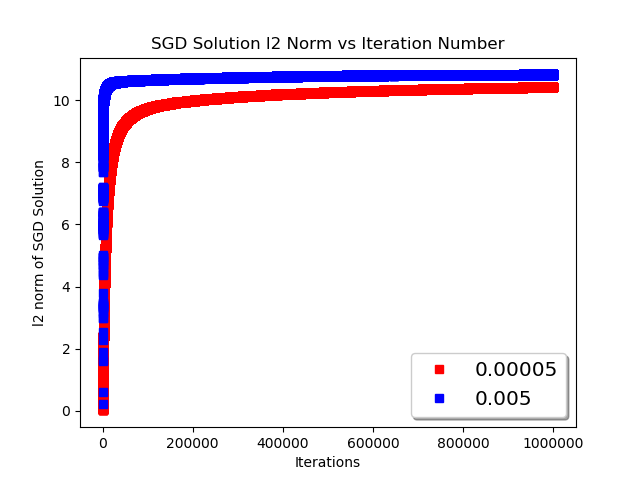
Average Test Error: 0.38507917451441664

For each step size, we can see a different training/test error along with a clear trend for the relationship between the two. As the training error decreases, the test error increases, meaning that the model is overfitting the training set with certain step sizes. This is ultimately undesirable because we want the model to be able to generalize to the unknown, so it is clear that step size is a hyperparameter that needs to be tuned to have optimal performance on a test set. Regardless of the step size chosen here however, in comparison to a, the training error is higher in exchange for a higher capability to generalize as shown by the lower test error.

D.



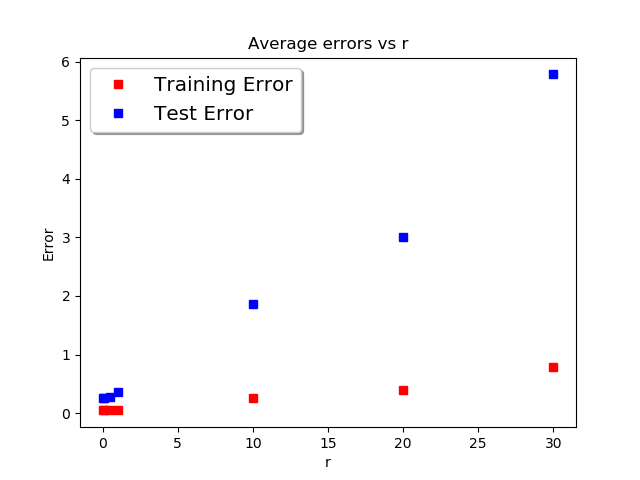




From the plots, it seems that the generalization ability of a model decreases as the step size increases. With the step size of 0.005, the model begins to overfit pretty quickly as shown by the increase in test error around the 200,000th iteration. While test error increases, the training error continues to decrease, confirming that when the training error gets too low, it is a sign of the model possibly overfitting.

TODO: answer How does the generalization ability of the final solution depend on the l2 norm of the final solution?

E.



From the graph, we can see the drastic effect large weight initializations have on the ability to generalize. There is a small gradual increase in training error with larger weights, and we hypothesize this is because bad, large initializations are harder to correct, but with 1,000,000 iterations, this is generally not a big problem. For testing however, small weight initializations are ideal. Regularization from part b penalizes large weights, so it makes sense that regularization helps generalize since it would work against large weights through training and/or initialization.

3.

Average test error: 0.7062

We decided to use l2 regularization because the dimensionality is greater than the size of the training data, and we wanted to avoid overfitting. After trying lambdas ranging from 5e-8 to 5e2, we found that a lambda of 5e-6 worked best. We were disappointed that the test error was so high given the results we found in 2b, thus we believe that a better test error is achievable. However, the test error won’t be as low as the results in 2b given the challenge that our dimensionality is so high relative to the training dataset size.

Here is our pseudocode:

set λ to 0.000005

set training error and testing error to 0

for each trial:

pick a true, X train and test, and y train and test

set a to (X\_trainTX\_train + λI)-1X\_trainTy\_train

find current normalized train and test error for a

increment training and testing error by current train and test error

divide training and testing error by number of trials to find average training and testing error