Samuel Atkins
March 14, 2019
1002951754

ROB313 Assignment 3: Gradient Descent

Assignment Objectives:

Learning Objectives:

1. To learn the advantages and disadvantages of using gradient descent versus stochastic gradient descent
2. To understand the effects of increasing and decreasing the learning rate
3. To understand the convergence trends of GD vs SGD
4. To understand how to implement versatile likelihood functions such as the log-likelihood function

Functional Objectives:

1. Implement gradient descent for a variety of learning rates by minimizing the least-squares loss function
2. Implement stochastic gradient descent for a variety of learning rates by minimizing the least-squares loss function
3. Implement a logarithmic likelihood function for gradient descent and stochastic gradient descent

Code Structure Explanation:

Main Functions:

There is only one function that needs to be run to execute the required question 1 implementations:

gradientDescent ()

- This function iterates through 5 different learning rates after acquiring the first 1000 points of the pumadyn32nm training set
- For each learning rate, 1000 iterations of gradient descent and stochastic gradient descent are executed to find the weight vectors that minimize the loss function
- The error at each iteration is tracked in an array
- The optimal weight vectors are updated every time the calculated error is less than the current minimum error
- After 1000 iterations, the error for both stochastic gradient descent and gradient descent are plotted separately as a function of the iteration count
- After iterating through all 5 learning rates, this function predicts on the pumadyn32nm test set and prints the test RMSE as well as the minimum error computed during training

Similarly, to question 1, there is only one function needed for the implementation of question 2:

logisticGradientDescent ()

- This function iterates through 5 different learning rates after initializing the iris classification dataset
- For each learning rate, 1000 iterations of the logarithmic likelihood stochastic and full gradient descent are computed
  - During each computation, several functions are called to compute the logarithmic gradient such as sigmoid(x, w), loglikelihood(x, y, w), and likelihoodGradient(x, y, w). The details of these functions follow in the section below
- Just as in question 1, the error at each iteration is tracked in an array and the optimal weight is constantly updated based on the current negative logarithmic likelihood value
- After 1000 iterations, the negative logarithmic likelihood is plotted as a function of the iteration number
- After iterating through all 5 learning rates, this function predicts on the iris test dataset and prints the test accuracy along with the minimum logarithmic likelihood

Secondary Functions:

accuracy (x, y, w)

- *Arguments*: an NxD x-matrix, an Nx1 y-vector, and a Dx1 weight-vector
- *Function*: outputs the accuracy of the weight vector for classification

RMSE (measurements, actuals)

- *Arguments*: an Nx1 measurement vector and an Nx1 actuals vector
- *Function*: outputs the RMSE of these two vectors

sigmoid (x, w)

- *Arguments*: an NxD x-matrix and a Dx1 weight-vector
- *Function*: computes the mathematical sigmoid function defined in the assignment

loglikelihood (x, y, w)

- *Arguments*: an NxD x-matrix, an Nx1 y-vector, and a Dx1 weight-vector
- *Function*: computes the logarithmic likelihood of the x matrix and the y vector given a weight vector, w

likelihoodGradient (x, y, w)

- *Arguments*: an NxD x-matrix, an Nx1 y-vector, and a Dx1 weight-vector
- *Function*: computes the likelihood gradient using the sigmoid function for the x matrix and the y vector given a weight vector, w

Question 1: Stochastic and Full Gradient Descent

In question 1 a gradient descent algorithm and a stochastic gradient descent algorithm were implemented for the pumadyn32nm dataset by minimizing the least-squares loss function. The first 1000 training data points were used. The following learning rates were tested: [0.1, 0.01, 0.001, 0.0001,

0.00001]. For the gradient descent algorithm, the RMSE decreased as the learning rate increased. The plots of each learning rate for gradient descent are illustrated in the Figure 1 below:

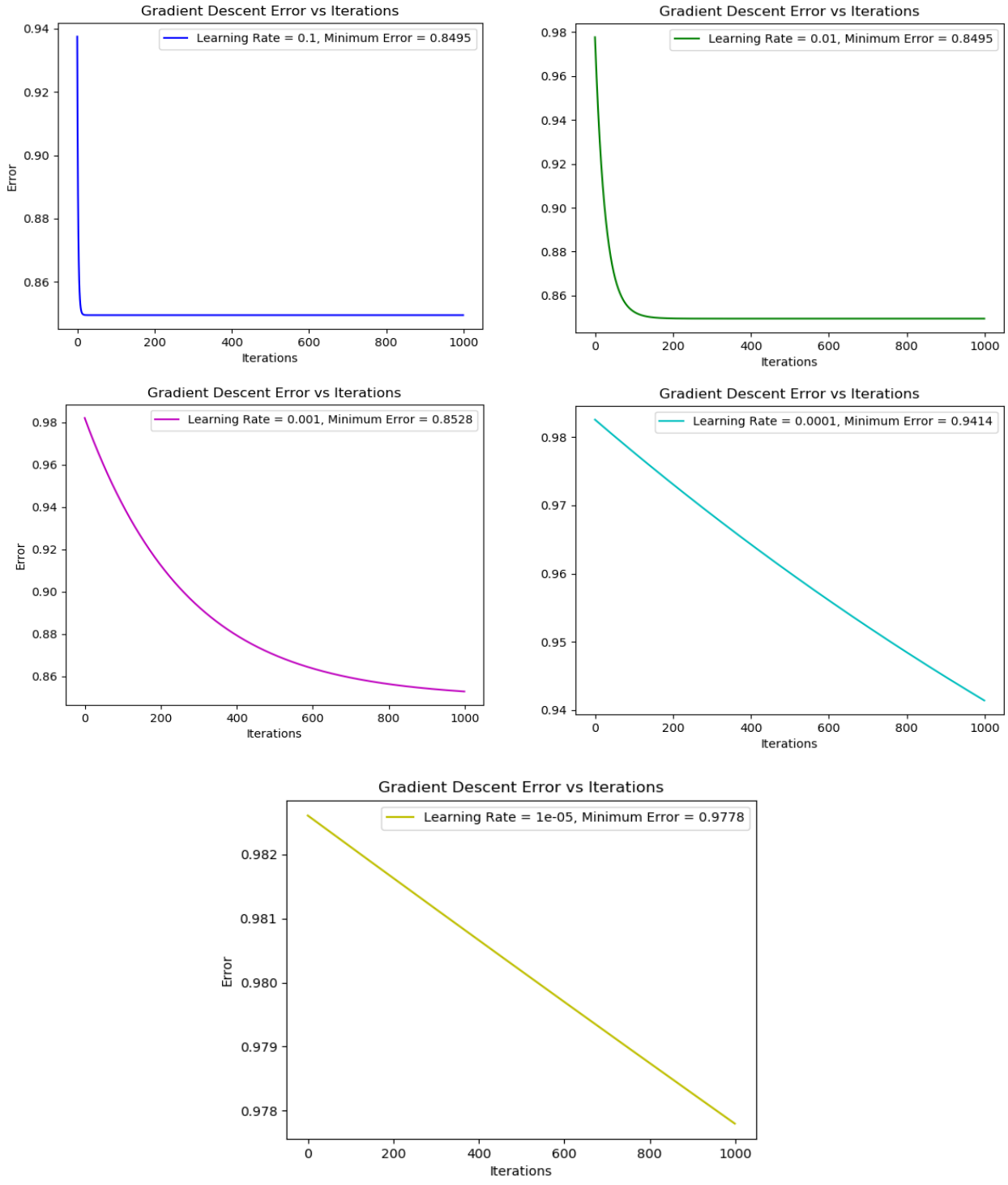


Figure 1: Gradient Descent RMSE for a Variety of Learning Rates

As one can see from the above figure, when the learning rate was very small, i.e. 0.0001 and 0.00001, the algorithm was not able to converge quick enough. The larger learning rates, such as 0.1, encouraged the algorithm to converge very quickly. Convergence was apparent for all of the selected learning rates and there was no oscillation. Based on the above figure, I would select 0.1 as the learning rate because it converges the fastest and without oscillation. The test RMSE calculated after predicting on the test set using the optimal weights was 0.87065. The test RMSE computed in assignment 1 was 0.862. The test RMSE computed using gradient descent was very similar to the test RMSE that was computed through algebraic methods.

For stochastic gradient descent, the exact same procedure was executed but instead of using the entire dataset to compute the gradient, a random index was selected. The following learning rates were used: [0.01, 0.001, 0.0001, 0.00001]. The RMSE of each learning rate was plotted as a function of the iteration number, just as in the gradient descent case. These results are delineated in Figure 2 below:

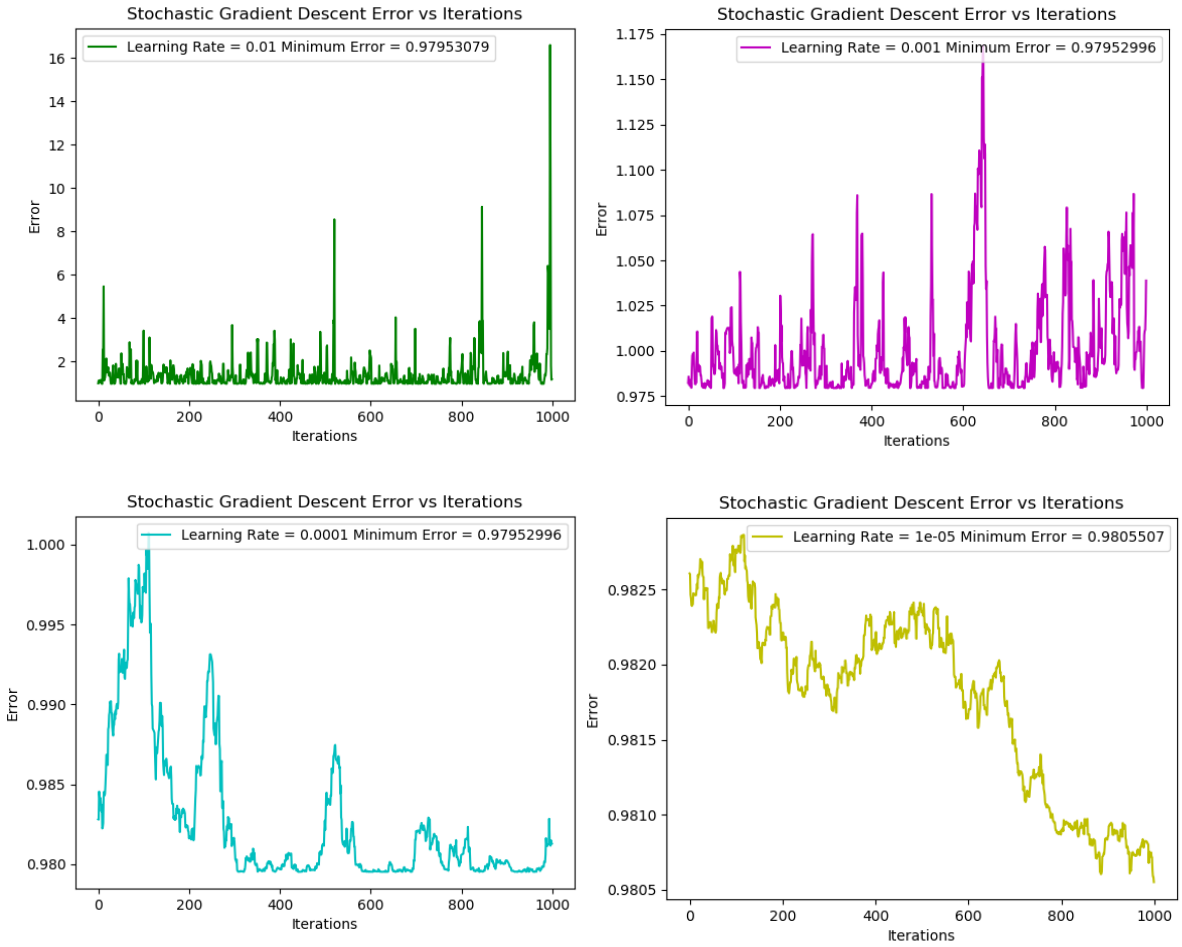


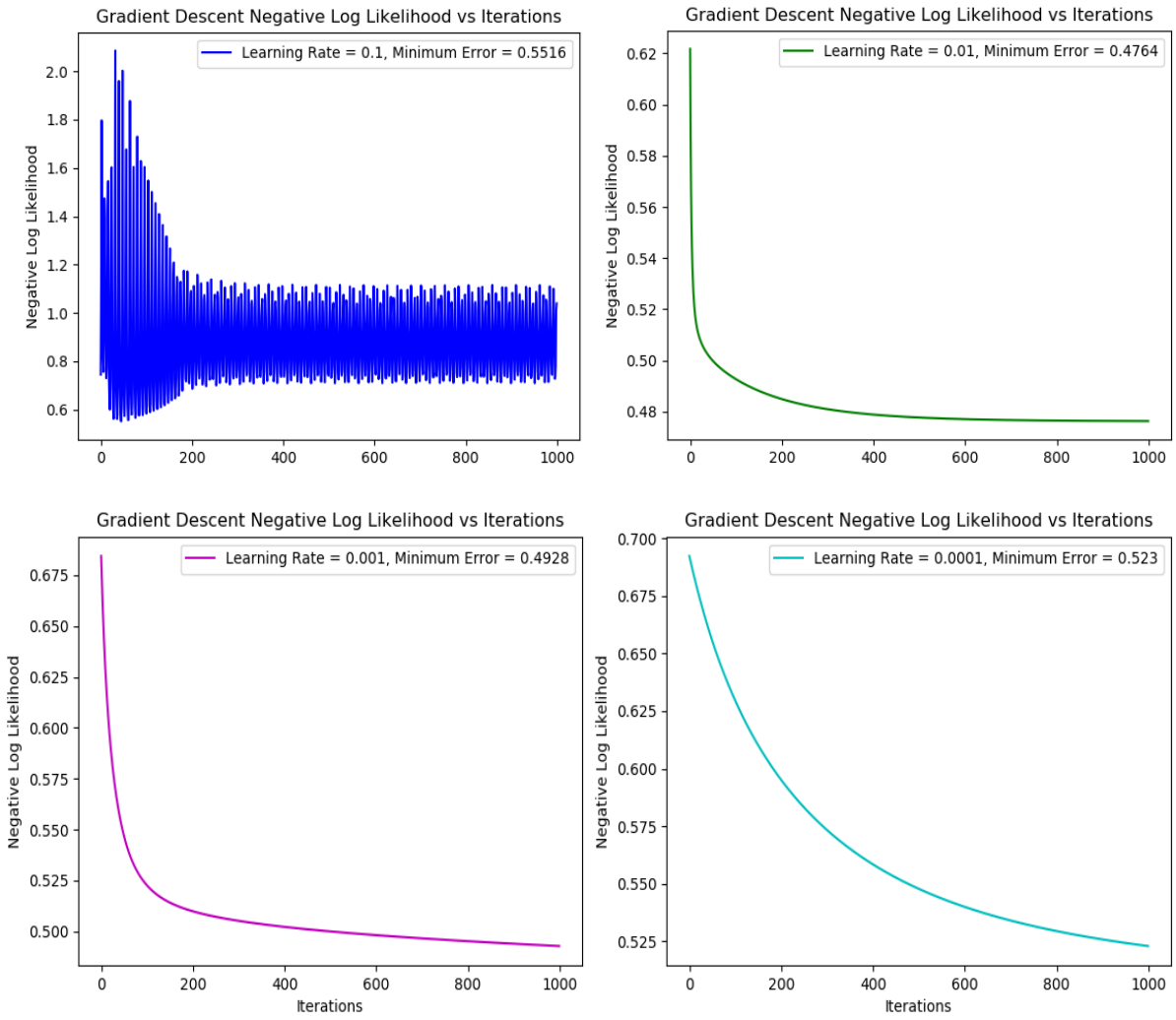
Figure 2: Stochastic Gradient Descent RMSE for a Variety of Learning Rates

Convergence, in this case, is not guaranteed. For a learning rate of 0.01, for example, the error actually grows with the number of iterations. For smaller learning rates, however, the RMSE appears to converge. When comparing SGD with GD, it is evident that GD converges much more reliably. Furthermore, GD does not oscillate whereas SGD does. If I were to select a learning rate for SGD based

on the results shown in Figure 2, I would select a learning rate of 0.0001 because it appears to converge relatively quickly to an error near 0.98. The 0.01 and 0.001 learning rates don't converge at all and the 0.00001 learning rate appears to generally decrease with the number of iterations, but it takes a long time. The test RMSE returned by this method is 0.9624. This is worse that the test RMSE returned by the GD method and significantly worse than the true error of 0.862 computed in assignment 1.

Question 2: Logarithmic Likelihood SGD and GD

In question 2 a gradient descent algorithm and a stochastic gradient descent algorithm were implemented for the iris dataset by minimizing the negative logorithmic-likelihood function. Prior to discussing the results, an interesting topic question was raised: "What will be the value of the log-likelihood if $f(x_i, w) = 1$ but the correct label is $y_i = 0$ for some i?" If this occurs, then that means we have computed log(0) which is negative infinity. This makes sense because it is impossible to have absolute certainty in a model. For the gradient descent algorithm the following learning rates were tested: [0.1, 0.01, 0.001, 0.0001, 0.00001]. The negative logarithmic likelihood was plotted as a function of the iteration number for each learning rate. These results are shown in Figure 3 below:



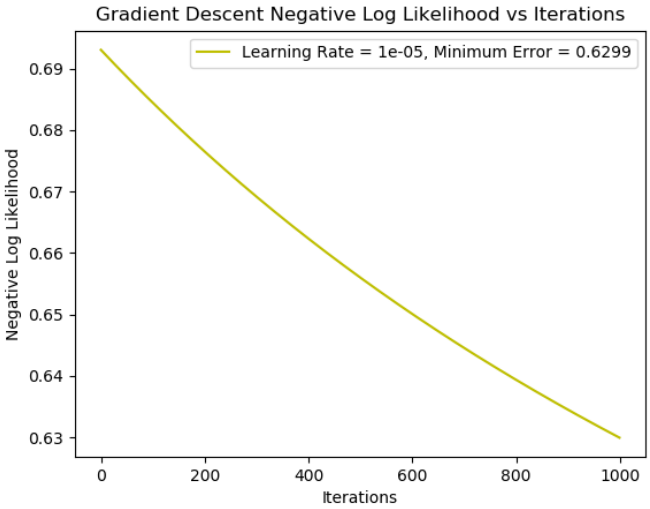
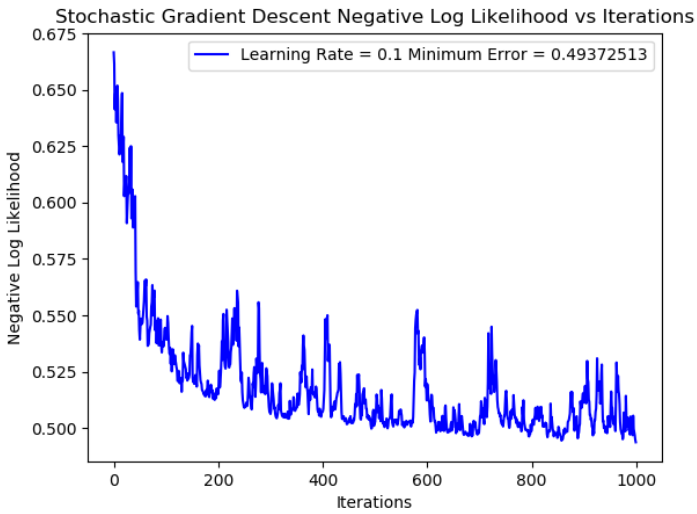
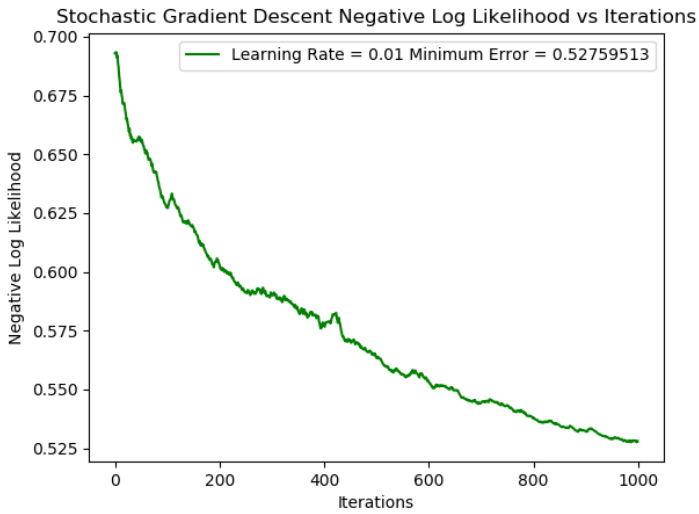


Figure 3: Negative Logarithmic-Likelihood for Gradient Descent

From the figures above we can see that convergence is apparent for all of the learning rates except for the aggressive 0.1 learning rate. When we select a larger learning rate such as 0.1, oscillation occurs. For gradient descent the accuracy after finding an optimal weight vector was 80%.

Using the logarithmic-likelihood function with SGD, the negative logarithmic-likelihood was plotted as a function of the iteration number just as above. These results are presented in Figure 4 below:





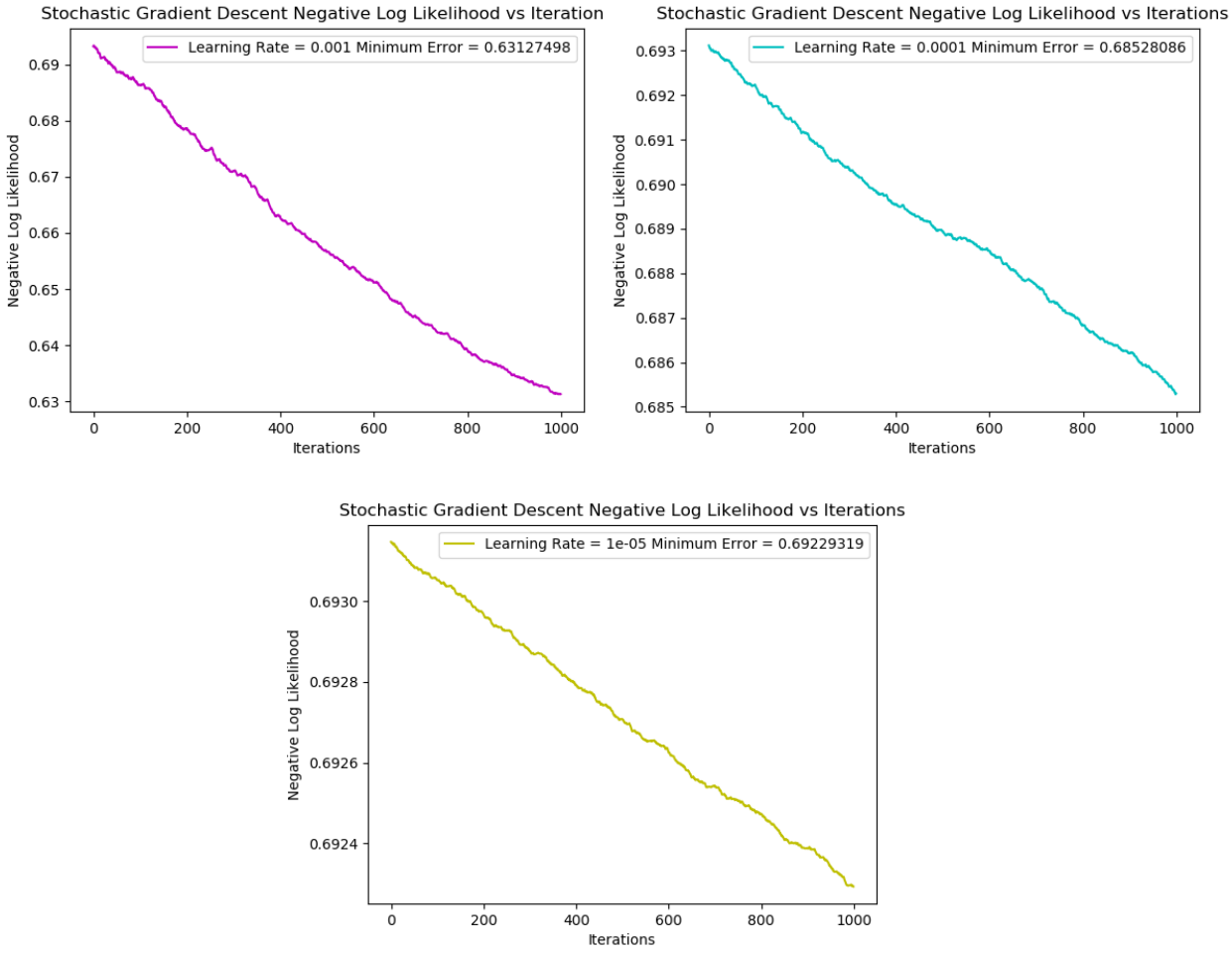


Figure 4: Negative Logarithmic-Likelihood for SGD for a Variety of Learning Rates

The results displayed in Figure 4 above show that the logarithmic-likelihood converges for all of the values in our learning rate testing set. Once again, it is apparent that higher learning rates converge faster but are more prone to oscillation. A good learning rate to select in this situation would be 0.1 because despite the oscillation, the convergence is apparent and aggressive. The logarithmic-likelihood is an attractive performance method because by using logarithms we are able to add factors as opposed to multiplying them together. This is computationally easier and also allows us to isolate factors easier.