Knowledge Representation Machine Problem 2: Logistic Regression: Learning and Inference

Student: Karan Daei-Mojdehi
Instructor: Dr. Guo-Jin Qi
University of Central Florida

November 3, 2015

Introduction

In this Machine Problem, we are requested to implement a Logistic Regression Learning Algorithm on a previously chosen data set from UCI machine learning website. Batch and stochastic gradient descent training are explored and evaluated as well as effect of adding a L2 regularization term to likelihood function. At the end, performance of Logistic Regression on our data set is compared to that of a decision tree which was implemented in previous machine problem.

Step 1: The Data Set

As suggested, I used the same data set for previous assignment which included information about adults salary with goal of predicting whether their salary is +$50k or not. This data set includes both categorical and numerical attributes and is made up of 48000 instances. As suggested, categorical data are coded into binary vector to make implementation of Logistic Regression possible. This problem will be a binary classification task. Output label of +1 corresponds to a +$50k salary, while label of -1 is used for salaries below $50k.The data set can be found online ¡Here¿.

Step 2: Implementation of Batch LR model

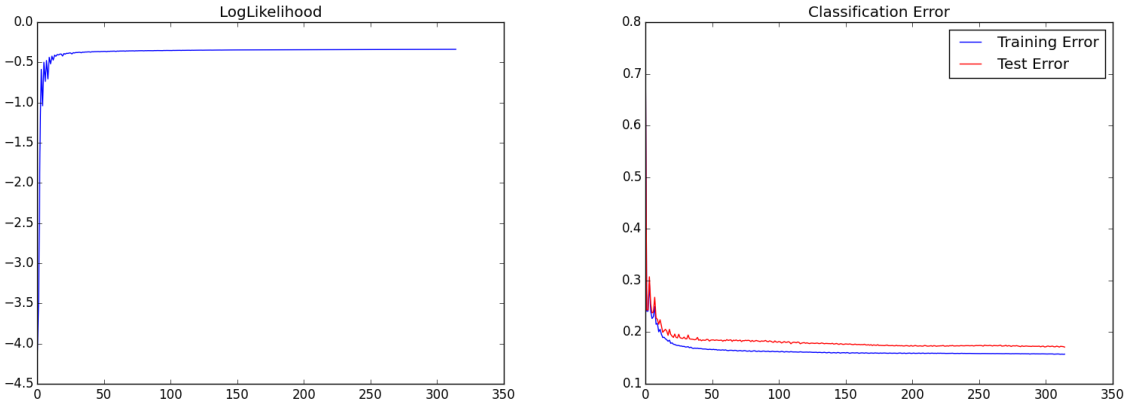
LR model is made up of a number of weights (equal to total number of attributes in our data set after binary vectorization in addition to a bias) and a likelihood function which foretells label of an input data by computing the following probability:

$$P(Y=1|X) = \frac{1}{1+exp(\sum_{i=1}^{N} -W^T X_i)} \tag{1}$$

in which X_i is a border of attributes of entry i and W is a vector of our model weights. For training of the LR model, likelihood is defined as probability of our LR model (weights) successfully predicting probability of labels all samples in our data set.Likelihood can be calculated using following equation:

$$Likelihood(Weights) = \prod_{X_i:Y_i=1} \frac{1}{1+exp(\sum_{i=1}^{N} -W^T X_i)} . \prod_{Xj:Y_j=-1} 1 - \frac{1}{1+exp(\sum_{j=1}^{N} -W^T X_j)} \tag{2}$$

$$= \prod_{i=1}^{N} \frac{1}{1+exp(\sum_{i=1}^{N} -Y_i W^T X_i)} \tag{3}$$



(a) log likelihood

(b) test and train error

Figure 1: Batch Training Results

where Y_i is the ground truth label assigned to X_i which is +1 for True entries and -1 for False entries. Log likelihood will be natural logarithm of above introduced function which will be equal to:

$$LogLikelihood(Weights) = -\sum_{i=1}^{N} ln(1 + exp(-Y_i W^T X_i)) \tag{4}$$

Now we use the gradient descent method to optimize likelihood of our model with respect to our model parameters (Weights). By taking derivative with respect to W and some simplifications, we will have:

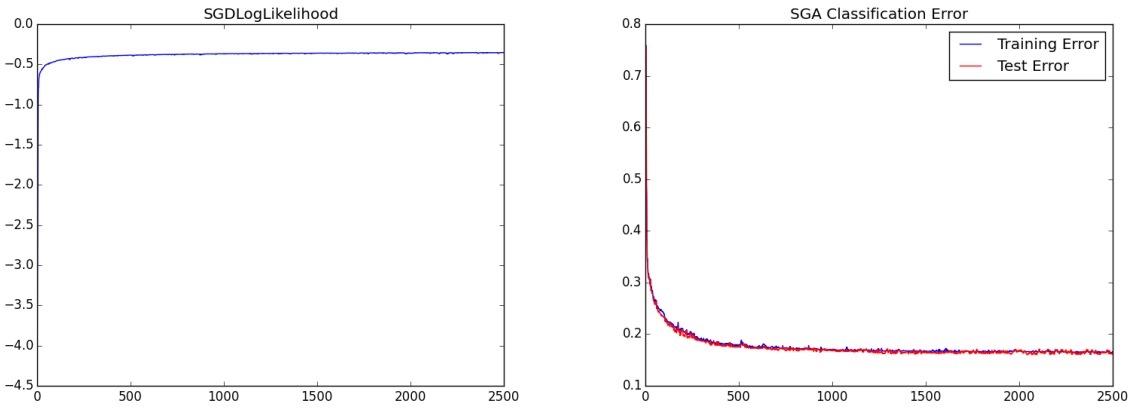
$$\nabla_W LL(W) = \sum_{i=1}^{N} \frac{Y_i X_i}{1 + exp(Y_i W^T X_i)} \tag{5}$$

In order to train the LR model, we update weights of the model iteratively in the direction of gradient ascent ('ascent' since we are maximizing likelihood) with a factor = η, also known as learning rate. In Batch mode training we look at all the training data in each iteration to extract update rule for weights, i.e. our update rule will be:

$$W\text{nextIteration} = W\text{currentIteration} + \eta \nabla_W LL(W) \tag{6}$$

This iterative update continues until one of the below stop criteria are detected:
a) number of iterations exceed a constant (MAX_ITERS)
b) the difference between two consequent iterations' weights drops below a certain threshold (TH_MIN_W_UPDATE) which is equivalent to detecting a certain amount of drop in gradient of log likelihood (we are using $\nabla_W LL(W)$ to update weights).
Also log likelihood in each iteration is compared to it's previous one and in case it is lower that previous step, learning rate temporary shrinks to one fifth of it's original size in that iteration.

Figure 1a shows log likelihood over iterations in a batch training run of our LR model. We observe that over iterations, our LR model become more probable ('likely') to estimate labels of our training data which means the model is learning. By a closer look at this figure, we can tell that learning rate is tuned correctly since log likelihood is monotonically increasing over iteration and converges in relatively short number of iterations. An inappropriate learning rate would had resulted in either fluctuation (a large learning rate) or slow convergence (small learning rate) in log likelihood. A smaller learning rate will smooth out the large fluctuations in early iterations before convergence, but will increase the number of iterations until stagnation. Figure(1b) shows how test and train error decrease over iterations and stagnate when training



(a) log likelihood

(b) test and train error

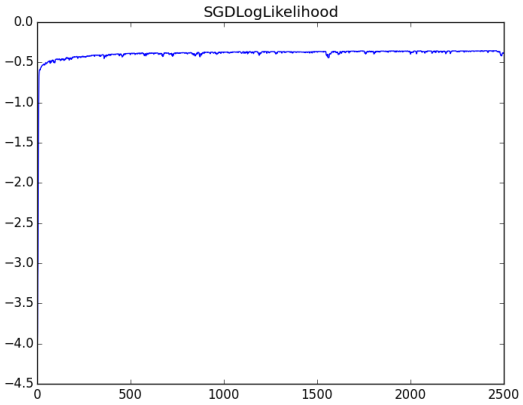
Figure 2: SGA Training Results, SGA_SAMPLE_SIZE = 40 , SGA_ETA = 0.3

converges. In this run all data samples where used for our task and was divided into 10 folds. Nine out of ten folds were used for training and test results are reported for the fold that was left out of training. After convergence accuracy on train and test data data 84% and 83.9% , True Positive Rate (TPR) was 55.35% and 55.31% , and True Negative rate (TNR) was 93.15% and 92.79% for train and test data. These measures shows a bias toward Negative labels in our data set. By comparing performance on test and train data and errors over iterations over those data in Figure(1b), we can assert that our LR model is not being over-fit on our train data. Main reasons behind this is our huge train data size, which we reduce in step 4 in order to observe over-fitting and how it's affected by regularization.

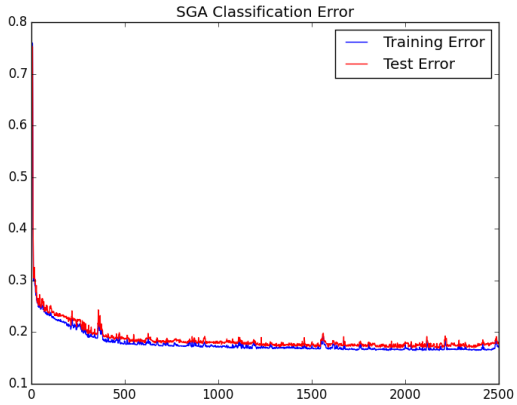
Step 3: Implementation of Stochastic Gradient Ascent (SGA) LR model

In order to change our learning algorithm from Batch learning to SGA, all we need to do is randomly choose a relatively small subset of our training data and compute likelihood and its gradient over this small subset in each iteration. Huge advantage of this method lies in reduction in complexity of computations in each step, since we don't have to compute likelihood in each iteration (although in this assignment, for reporting purposes, still log likelihood is calculated and stored in each iteration). Tuning of learning rate (SGA_ETA) and size of randomly chosen subset (SGA_SAMPLE_SIZE) is essential to convergence of SGA method. A smaller subset would require a smaller learning rate since it's information is less reliable while a larger one includes more reliable information and therefore can exploit a larger learning rate. Rest of the steps are the same as batch learning.

Figure 2 shows results from SGA training and its evaluation on test data set with subset size of 40 and learning rate of 0.3. Log likelihood in 2a is monotonically increasing towards 0 which shows SGA LR is learning with an appropriate learning rate. In this run, Accuracy, TPR, and NPR were 83.38%, 59.74%, and 90.87% on train set and 83.78%, 60.15%, and 91.3% on test set respectively. Again large size of our training data and relatively small test data set has swept away over-fitting problem. In another run with the same learning rate but subset of sizes 10 and 3 (Figures 3 and 4 respectively), we can observe that weights move in more diverse directions in each iteration and therefore result in a fluctuation in likelihood and classification error on both test and train data set. Although Classification evaluation results exceed the ones from runs with larger subset size in some runs, they are not reliable and have a ± 10%

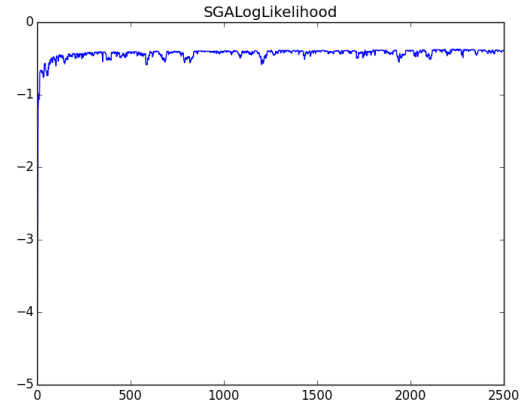


(a) log likelihood

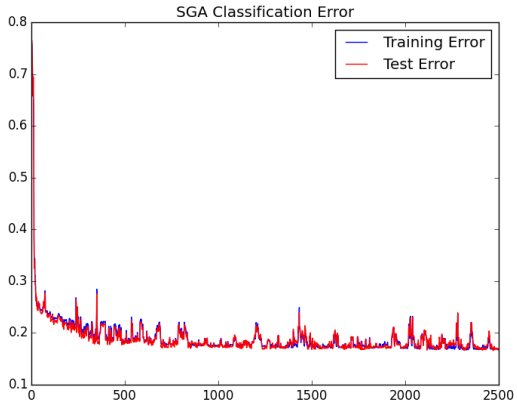


(b) test and train error

Figure 3: SGA Training Results, SGA_SAMPLE_SIZE = 10 , SGA_ETA = 0.3



(a) log likelihood



(b) test and train error

Figure 4: SGA Training Results, SGA_SAMPLE_SIZE = 3 , SGA_ETA = 0.3

fluctuation. A smaller learning rate in these runs would had reduced these fluctuations.

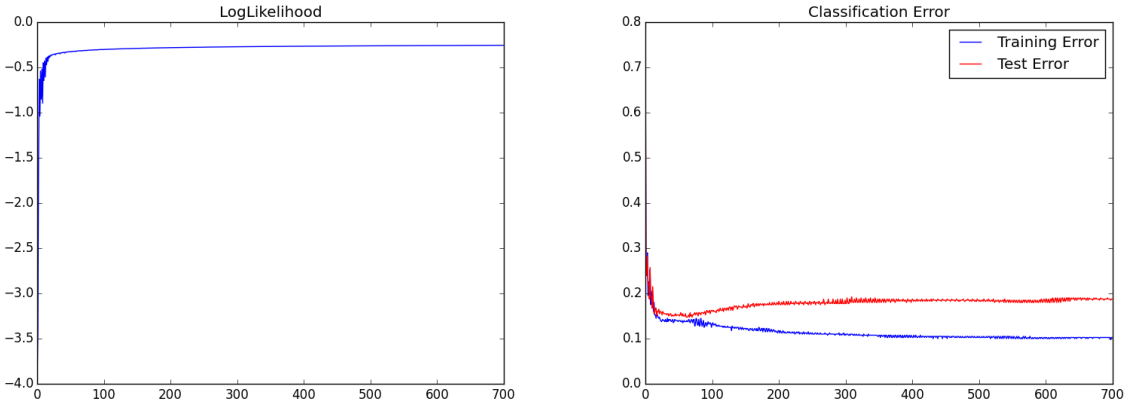
Step 4: Adding Regularization Term

Adding a regularization term will prevent a model from fully fitting a train data set and is mainly used to avoid over-fitting. Regularization is done with goal of reaching a better out of sample performance by avoiding fully 'memorizing' training date, therefore we expect to witness a decrease in in-sample performance.Adding an L_2 regularizer to the log likelihood function will result in the new cost function:

$$LogLikelihood(Weights) = -\sum_{i=1}^{N} ln(1 + exp(-Y_i W^T X_i)) - \frac{1}{2\sigma^2}||W||^2 \tag{7}$$

By taking gradient of the new log likelihood function, we arrive at the new update rule for regularized learning:

$$\nabla_W LL(W) = \sum_{i=1}^{N} \frac{Y_i X_i}{1 + exp(Y_i W^T X_i)} - \frac{1}{\sigma^2} W \tag{8}$$



(a) log likelihood

(b) test and train error

Figure 5: Batch over-fit Training Results, folds = 2 , ETA = 5, no regularizer

$$W\text{nextIteration} = W\text{currentIteration} + \eta \nabla_W LL(W) \tag{9}$$

Batch Regularization

Before adding a regularizer, note that as observed in previous steps, the portion of data that we are using for training of our model is huge and we are not observing any over-fitting in the first place. Therefore, I first decreased training data set size portion and also discarded large portion of available data (reduced total available instances to 800) and used half of those data for training and the other half for test in order to make the conditions more likely for over-fitting the model. I also increased maximum number of iterations as well as threshold in minimum required update in weights for each iteration. Figure 5 shows results of my success in over-fitting the data. By looking at Figure 5b we can see that the model starts over-fitting at iterations around 70 since train error is still decreasing while test error starts rising. Now is the time to observe effect of adding a regularizer. while keeping all other parameters fixed, vary sigma from 1 to 100 , and perform 10 runs of training for each sigma (on random samples from the huge pool of data set), Figure 6 shows the average of error on validation set for different values of sigma. With this values and corresponding average error in mind, we set sigma value to 50 and 70 and evaluate effect of regularizer term on over-fitting.

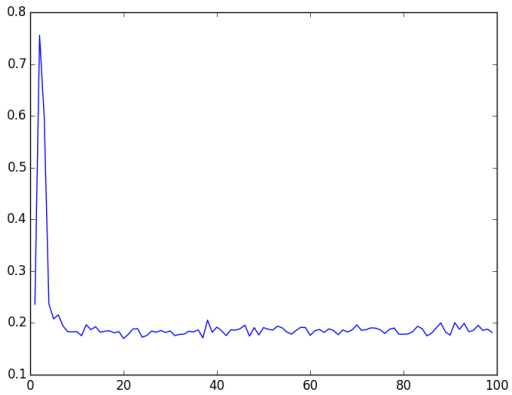
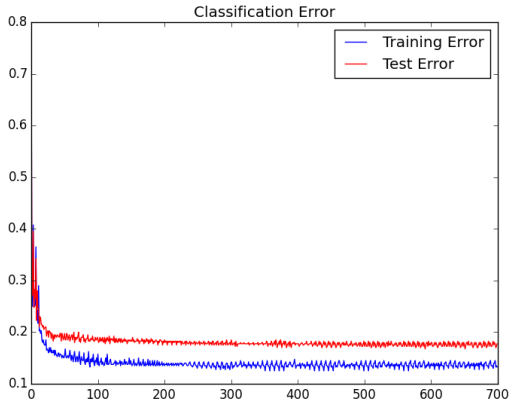
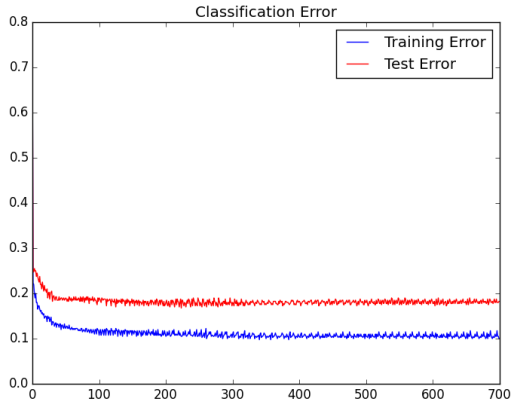


Figure 6: Average Validation Error over Sigma (Batch Regularization)

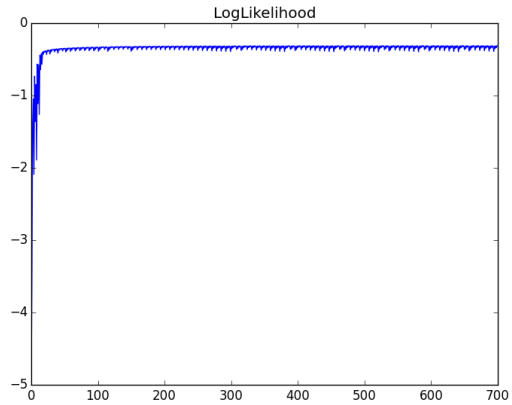


(a) Error with Regularization, SIGMA = 50

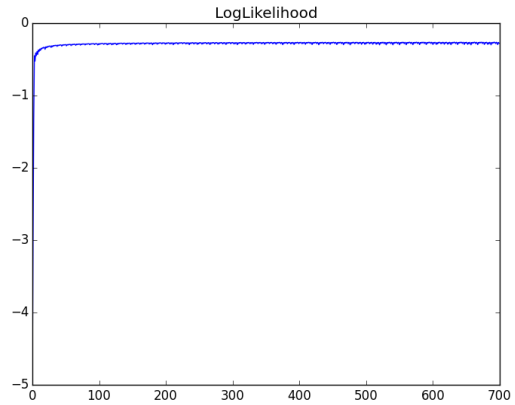


(b) Error with Regularization, SIGMA = 70

Figure 7: Regularized Batch Training Error Results, ETA = 5



(a) SIGMA = 50



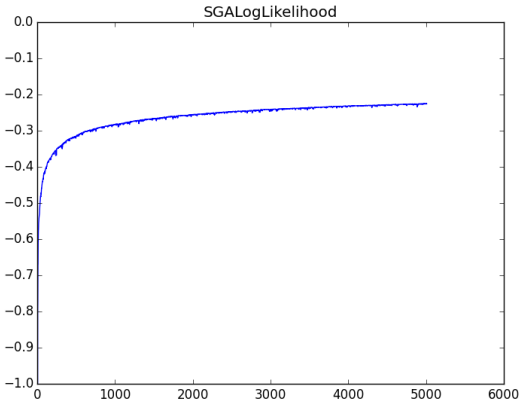
(b) SIGMA = 70

Figure 8: Regularized Batch Training LogLikelihood Results, ETA = 5

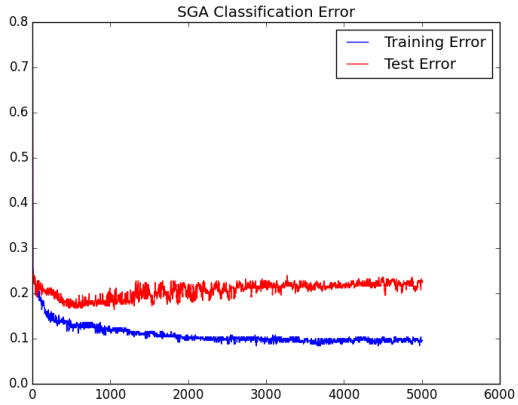
Figures 7a and 7b show classification error over iterations. We can observe that regularization is taking effect: although error on training set has slightly increased, (compare it to Figure 5b) , error on test data set is lower and it does not increase beyond any iteration (so far). Clearly we can assert that regularization doesn't permit our model to over-fit. Log likelihood in these runs are plotted in Figure 8 and show that the regularizer with sigma =70 results in less fluctuation in loglikelihood and is more stable after convergence.

Stochastic Gradient Ascent Regularization

Same as Batch regularized section, first we try to arrive at over-fit results without a regularizer and then add a regularizer term to examine effect of regularization. Every step for regularization of SGA is same as Batch method, except that for reporting purposes, we calculate error and likelihood not only for randomly selected subset of training data, but also for whole training data and test data. I reduced total available number of instances to 500 and maximum number of iterations to 5000 in order to make the conditions suitable for over-fitting. Figure 9 shows log likelihood and error in this run. It is clear that our SGA method is over-fitting the model since test error starts rising after iteration 800.



(a) log likelihood



(b) test and train error (note over-fitting)

Figure 9: SGA Training Results, SGA_SAMPLE_SIZE = 40 , SGA_ETA = 0.3

Now that we know our model over-fits in these conditions, we fix our training parameters, and follow the same procedure as Batch regularization. Figure 10 shows results of adding a regularizer to our update rule for sigmas in range of 1 to 100.

According to Figure 10 , we choose sigma value of 15 (SIGMA = 15)and plot the results in Figure 11. As we were hopping, we can perceive that again adding a regularizer has decreased our performance on training error (compare with Figure9b, but results in a better classification on test data set. We can also note that signs of over-fitting have vanished in these runs thanks to regularizer term. Log likelihood and classification error are plotted for other values of sigma in figures 12 to 14 . By observing log likelihood, we can note that smaller values of sigma will result in a poor log likelihood (which is a measure of how much our model fits the training data) and increasing sigma will enhance log likelihood over training data and allows our model to better fit training data, while increasing chances of over-fitting.

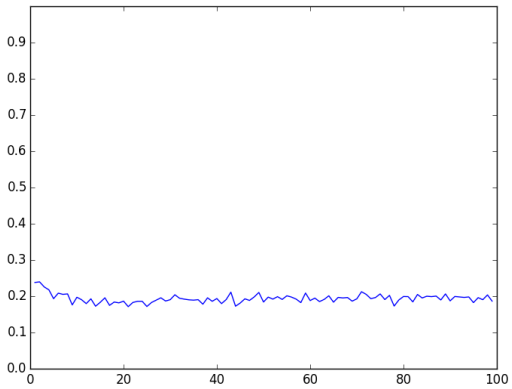
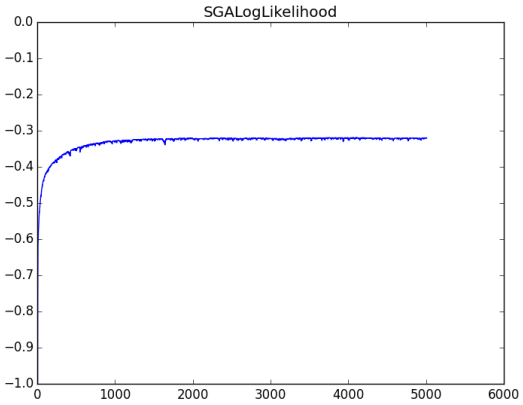
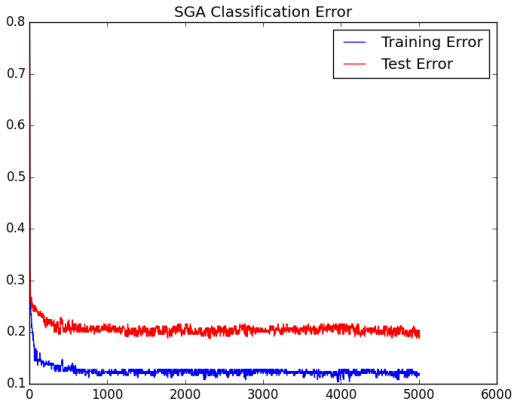


Figure 10: Average Validation Error over Sigma (SGA Regularization)

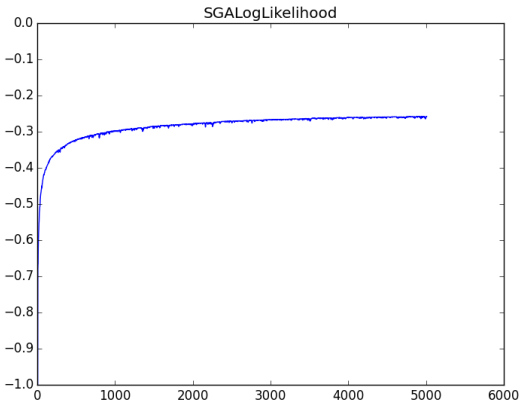


(a) log likelihood

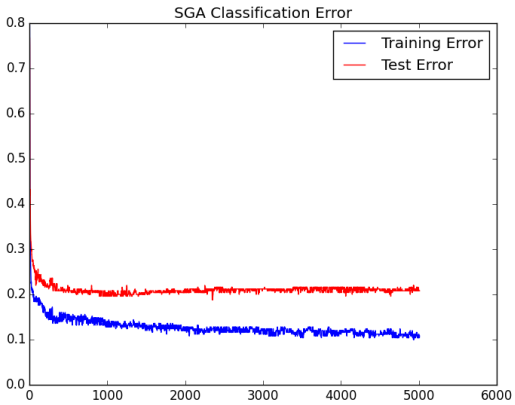


(b) test and train error

Figure 11: SGA Regularized Training Results, SGA_ETA = 0.3, SIGMA = 15



(a) log likelihood

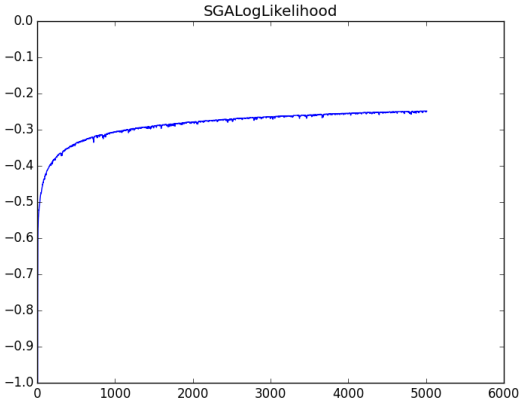


(b) test and train error

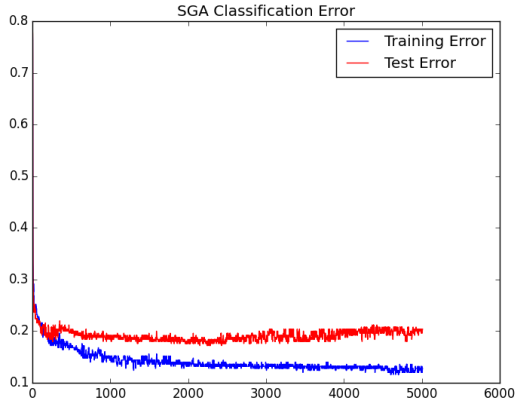
Figure 12: SGA Regularization Training Results, SGA_ETA = 0.3 , SIGMA = 45

Step 5: Report and Comparison with Decision Tree

By comparing evaluation results from original runs (no the runs in which we had intentionally over-fit the models) we observe that logistic regression has a slightly lower overall accuracy, while it has a better TNR (True Negative Rate). Having in mind that our data set is more biased toward Negative samples, we can tell that performance of logistic regression is more better on the selected data set. This can be mainly due to inability of decision to monitor all of attributes at the same time and make a decision on a combination of them, i.e. correlation between attributes of our data set is captured better by the LR model.

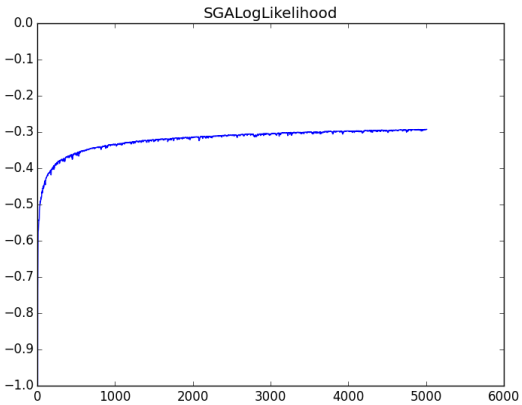


(a) log likelihood

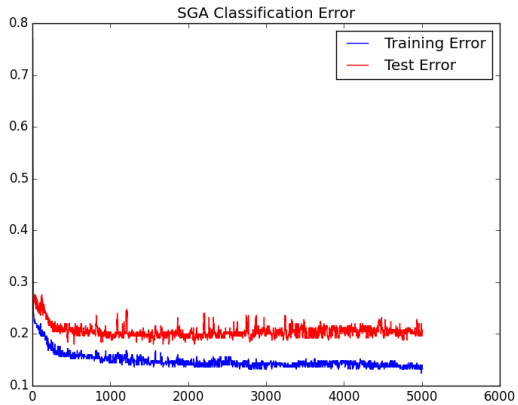


(b) test and train error

Figure 13: SGA Training Results, SGA_ETA = 0.3, SIGMA = 75



(a) log likelihood



(b) test and train error

Figure 14: SGA Regularized Training Results,SGA_ETA = 0.3, SIMA = 90