CS 434 Assignment #1 Report

Group: Tsewei Peng, Marsellus Everard-Kelly

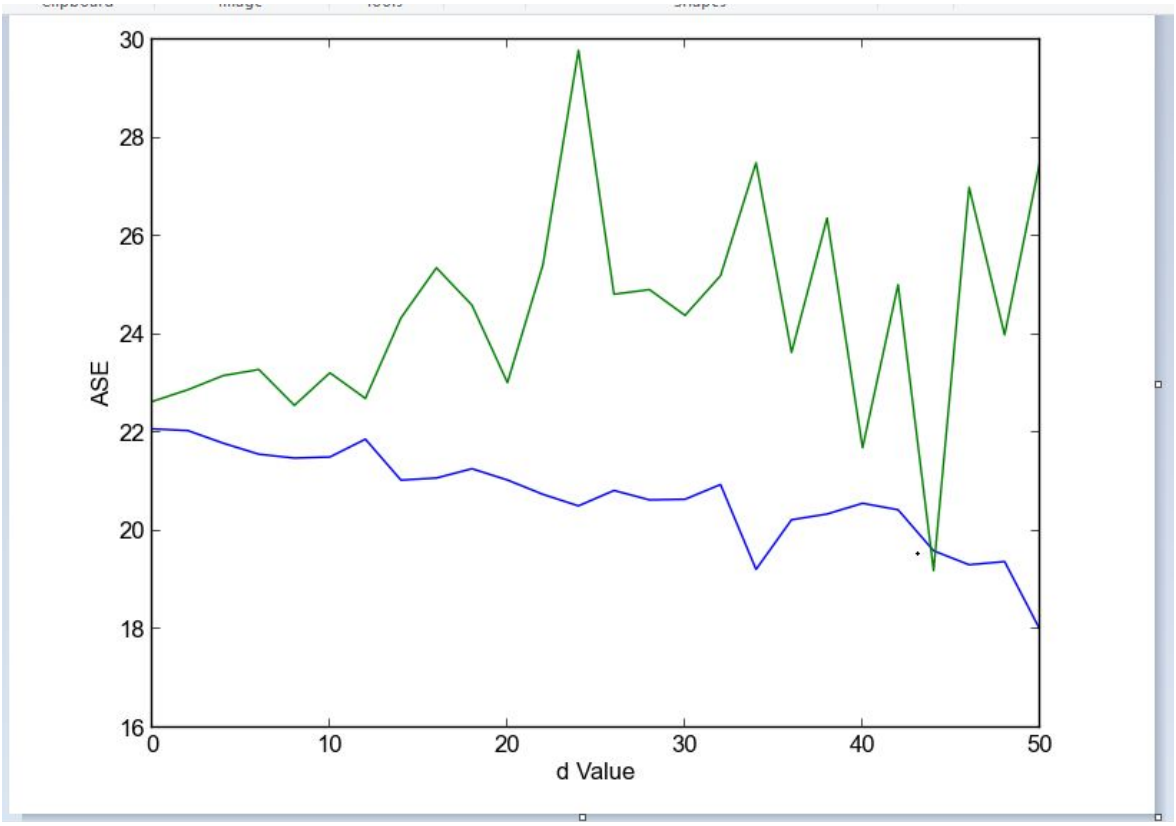
Linear Regression:

1. Learned weight vector:
   [[3.95843212e+01]
   [-1.01137046e-01]
   [4.58935299e-02]
   [-2.73038670e-03]
   [3.07201340e+00]
   [-1.72254072e+01]
   [3.71125235e+00]
   [7.15862492e-03]
   [-1.59900210e+00]
   [3.73623375e-01]
   [-1.57564197e-02]
   [-1.02417703e+00]
   [9.69321451e-03]
   [-5.85969273e-01]]

2. Training data ASE: 22.081273187
   Testing data ASE: 22.6382562966

3. After removing dummy variable:
   Training data ASE: 24.4758827846
   Testing data ASE: 24.2922381757

The resulting ASEs are worse when the dummy variable is removed. This is because the dummy variable accounts for the shift in the y-intercept.

4. 

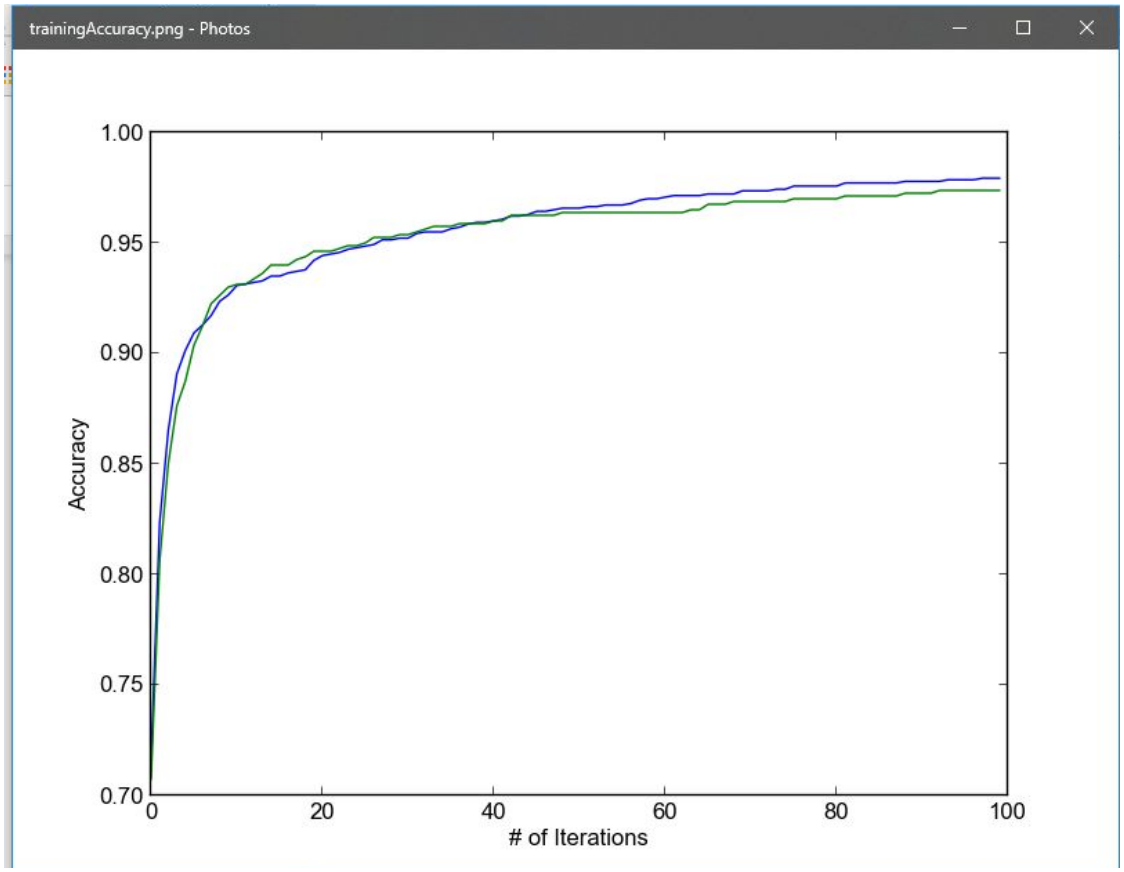
Green is testing data
Blue is training data.

For testing data the ASE tends to shoot up or down as the number of new random attributes increases passed 10. This is because we are feeding the algorithm data that is irrelevant, but it can't tell the difference. Therefore, it factors all the data into the weight vector. This throws off the ASE because the bad data can affect the predictions.

This can be seen in the training data ASE which shows a downward trend when adding new random attributes. It's showing there is less error because the algorithm is fitting to the training data even though the new features are random.

Logistic Regression:

1. With a small learning rate such as (0.0000001 or 10*10^-8) it takes 12 iterating to achieve accuracy 90% or above. Too small of a learn rate and the logistic regression algorithm can't improve over iterations. For example .000000001 or 10*10^-10 resulted in a constant 50% accuracy. The best learning rates are .1 to .0001 which need about 4-7 iteration s achieve 90% accuracy. The trend is the accuracy increases as time goes on because the algorithm can take in the data over the iterations and improve its predictions.



Green is testing data
Blue is training data

2. Given training example (xi, yi), i = 1, … , n
   Let w ← (0, … , 0) // initiation, length is the number of features
   Repeat:
   Gradient ← (0, … , 0) // length is the number of features
   For i in range of data (From 1 to n where n is the number of samples)
   resultYi = 1 / [1 + e ^ (-wT * xi)]
   Gradient ← Gradient + [(resultYi - yi)*xi + lambda * w]
   // Lambda * w is the gradient term for ½ lambda * |w|^2
   w ← w - LearningRate * Gradient
   Until condition satisfied
   // Condition: Completing an amount of iterations, or when magnitude of gradient is small
   // enough

3. Learning Rate: .000001
   lambda values:

   [1.00000000e-03 1.00000000e-02 1.00000000e-01 1.00000000e+00

1.00000000e+01 1.00000000e+02 1.00000000e+03]

Training data accuracies:
[0.8021428571428572, 0.8021428571428572, 0.8014285714285714,
0.7978571428571428, 0.7878571428571428, 0.8014285714285714,
0.6628571428571428]

Testing data accuracies:
[0.79125, 0.79125, 0.79125, 0.79125, 0.785, 0.79, 0.6425]

When the amount of iterations and the learning rate remain as a constant with increasing lambda The accuracy shows a downward trend. This is because in each iteration of the algorithm you are multiplying lambda to the weight vector. However larger lambda means changing the weight vector more drastically since the resulting multiplication will be bigger. For example say the weight vector has a value of .1. If lambda is .1 the resulting multiplication is .01. If lambda is 1000 the resulting multiplication is 100. This means using a smaller lambda would be better to get more accurate training and testing values.