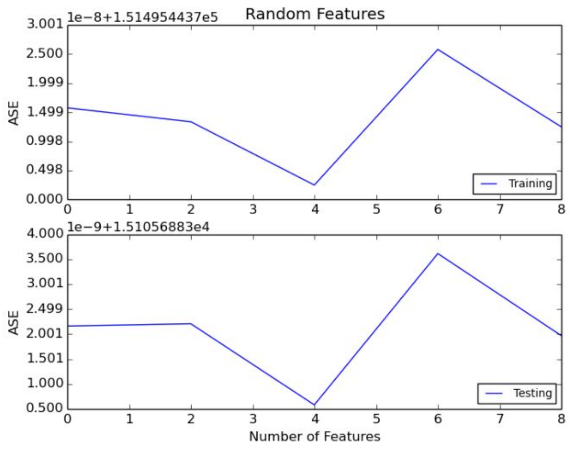
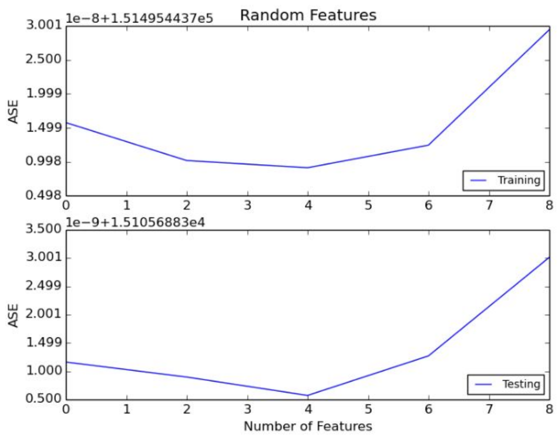
Monica Sek and Arthur Shing
CS 434
Implementation Assignment 1
4/15/2018

1. Linear Regression
    1.1. **Weight Vector:** [3.9584e+01 -1.0114e-01 4.5894e-02 -2.7304e-03 3.0720e+00 -1.7225e+01 3.7113e+00 7.1586e-03 -1.5990e+00 3.7362e-01 -1.5756e-02 -1.0242e+00 9.6932e-03 -5.8597e-01]
    1.2. **Training ASE:** [9561.1913]
    Testing ASE: [1675.231]
    1.3. **Training ASE with no dummy:** [10598.0572]
    Testing ASE with no dummy: [1797.6256]
    Removing the dummy variable increased our average squared error. This is because the dummy variable is correlated with the best fit line, so adding it to the data set would improve predictions.
    1.4. **Weight vector:** [1.2506e-12 5.3291e-15 1.1657e-15 7.7716e-16 -1.6698e-13 3.4106e-13 -3.5527e-14 -8.8818e-16 -4.4409e-14 5.3291e-15 -3.0531e-16 -2.8422e-14 2.2204e-16 1.0000e+00]
    Training ASE: [151495.4437]
    Testing ASE: [15105.6883]

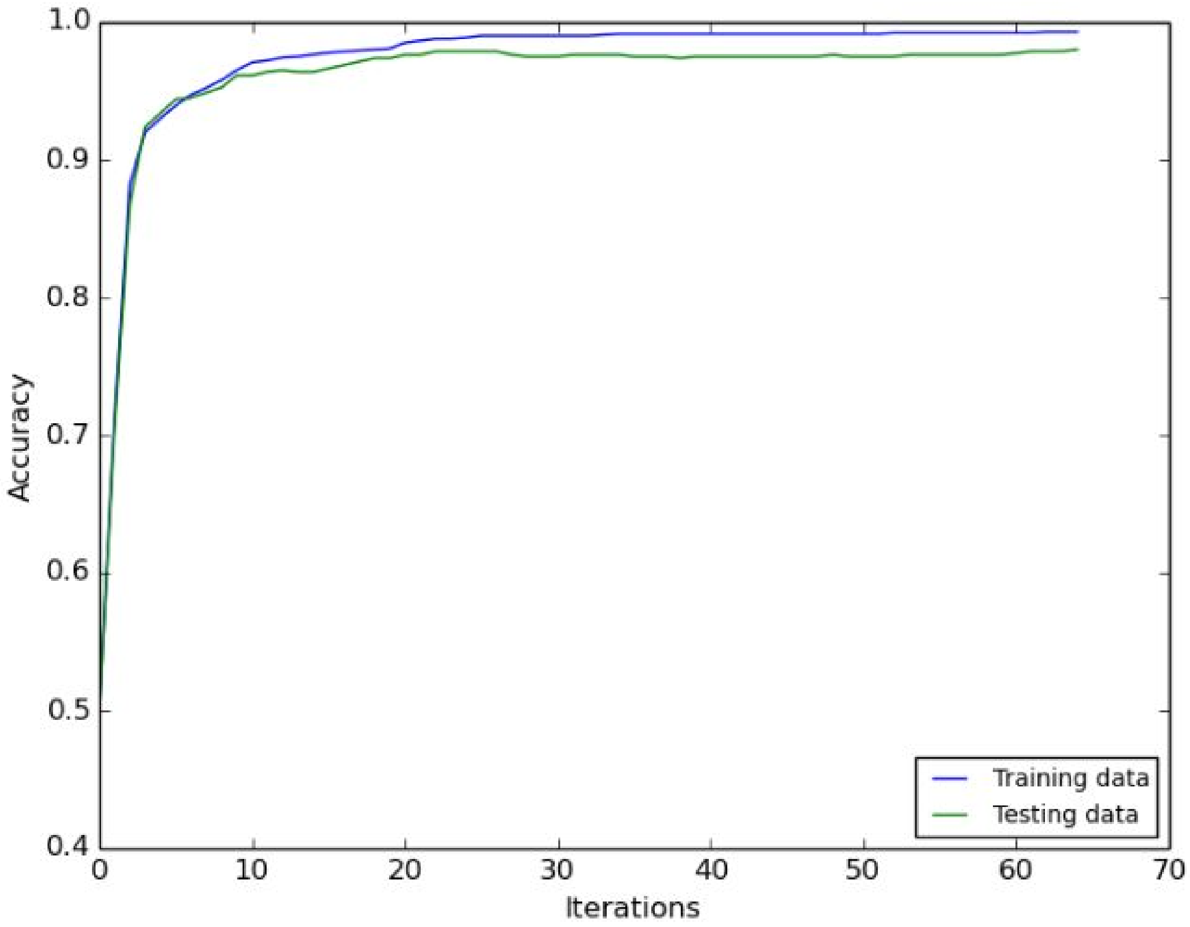




As you can see from the graphs above, our random feature generator outputs different values. Generally, we can see a decrease in ASE when we added 4 random features. As we continue, ASE increases. The first graph shows that the ASE decreases again however the second graph demonstrates that the ASE continues to gradually increase.

2. Logistic Regression with Regularization

2.1.

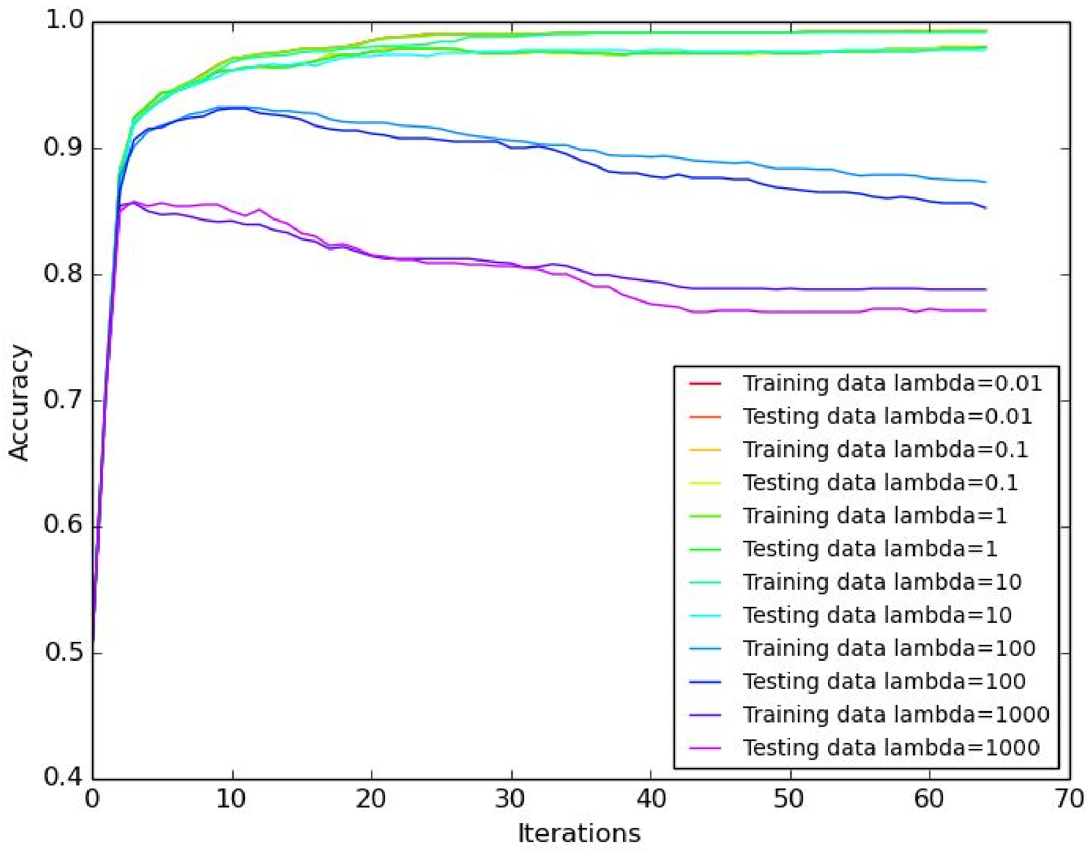


We used a fixed number of iterations (65) to determine our stopping point after playing around with learning rates. We found that a learning rate too large would end up not training the data, and one too small would take too long. We determined a good learning rate to be 0.000001. Our graph displays an asymptotic function. It begins at 50% random guessing and as we continue to update the coefficients we get closer to 100% accuracy to the expected value cost.

2.2.

```
Def batch_gradient_desc ( w, x, actual)
      Lambda = 0.1
      Epoch = 65
      learnRate = 0.0005
      For e in epoch:
            Delta = w
            For row in x[e]
                  Prediction = (1/1 + e^-($w^t$x))
                  Error = actual - prediction
                  Regulator = lambda * w
                  Error = error + regulator
                  Odds = y * (1-y)
                  Delta = delta + (row * error * odds * learnRate)
            w = delta
      Return w
```

2.3.



Our graph shows that the smaller lambda gives us a more accurate predictor at first. As we go through iterations, a lambda of 0.1 through 1 converge to close values at ~0.99. As we increased our lambda exponentially, the accuracy gets lower. The accuracy also got lower as iterations continued, for lambda values above 10. This probably means that the regularization began overcompensating. Between lambda=10 to lambda=1000 there was a drastic decrease in accuracy.