Team Member Zero: Trevor Swope - swopet@oregonstate.edu
Team Member One: William Selbie - selbiew@oregonstate.edu

Language: Python

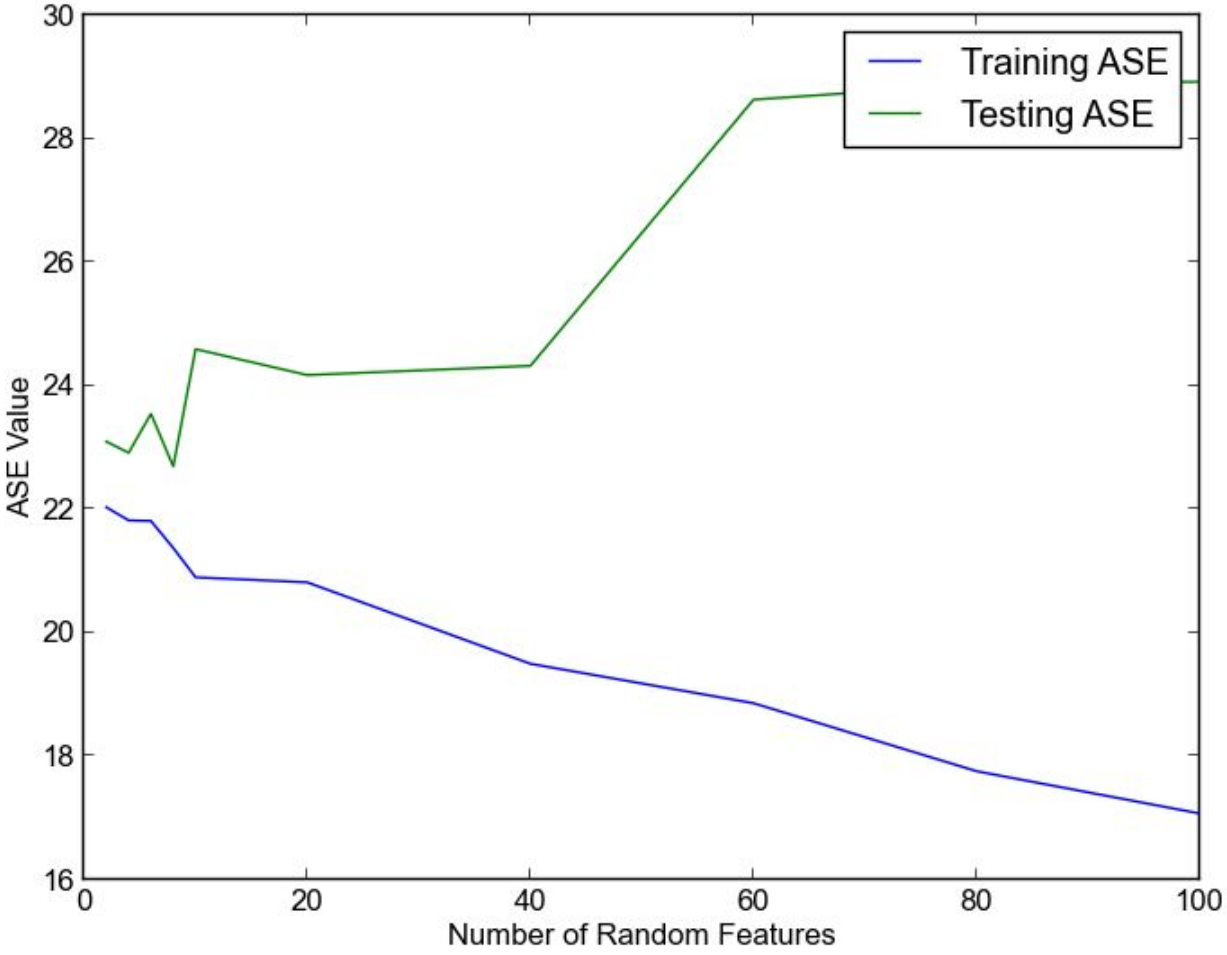
1.

A. **Learned Weight Vector:**
[[-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 3.95843212e+01]]

B. **Training ASE:** 22.081273187 **Testing ASE:** 22.6382562966

C. [[-9.79342380e-02 4.89586765e-02 -2.53928478e-02 3.45087927e+00
-3.55458931e-01 5.81653272e+00 -3.31447963e-03 -1.02050134e+00
2.26563208e-01 -1.22458785e-02 -3.88029879e-01 1.70214971e-02
-4.85012955e-01]]
ASE for training data without dummy variable: 24.4758827846
ASE for testing data without dummy variable: 24.2922381757
Most real linear trends will have a +b component of some kind, no matter how small. Unless that b value is 0, removing the dummy variable disallows for a model that fits that description. So, if the trend should include such a value, excluding it will create a line that follows it closely but does not quite do as well.

D.

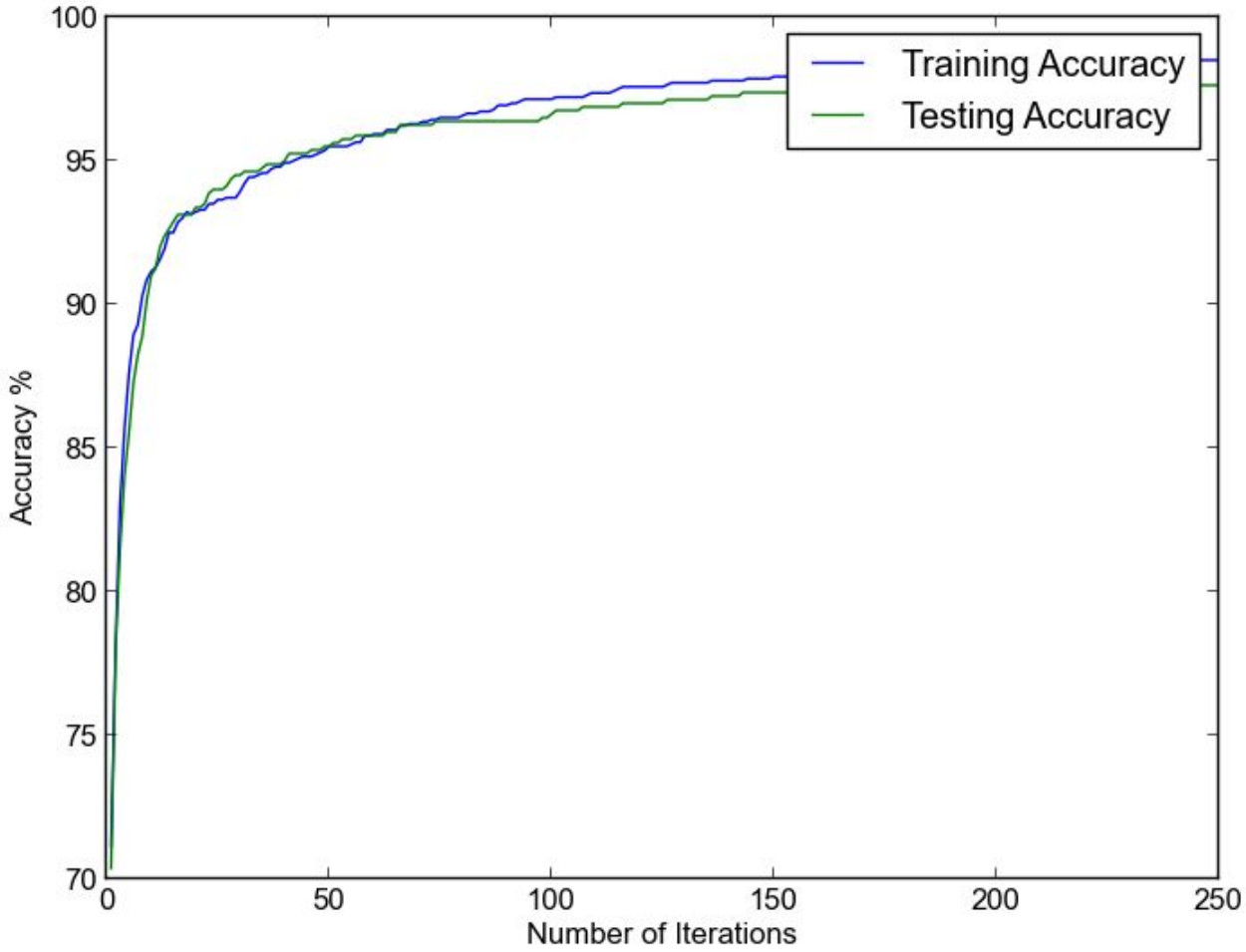


As can be seen above, the training ASE decreases with increasing random features (d) while the testing ASE increases. This is because on the training data the weights for the shifts to match the actual correlation, regardless of the fact that dummy variables are not actually linked in any way to the target. However, the correlation/weights are only accurate for the training set because the dummy variables are not related to the target variable, so the testing stage prediction is thrown off because of this.The end result is more accurate training stage and a less accurate testing stage.

2.

A. The following two graphs depict the relationship between accuracy and iterations for learning rates of 0.000000001 and 0.01 respectively. As can be seen, with a much smaller learning rate, convergence

takes more iterations to reach.



B .

Given: training examples $(\boldsymbol{x}_i, y_i), i = 1, \ldots n$

Let $\boldsymbol{w} \leftarrow (0, \ldots, 0)$ //initialization

Repeat

$\nabla = (0, \ldots, 0)$

for $i = 1, \ldots, n$

$$\hat{y}_i \leftarrow \frac{1}{1+e^{-\boldsymbol{w}^T \boldsymbol{x}_i}}$$

$$\nabla \leftarrow \nabla + (\hat{y}_i - y_i)\boldsymbol{x}_i$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta\nabla$$

Until $|\nabla| \leq \epsilon$

The pseudocode we used is almost identical to the pseudocode provided with the exception that in order to implement regularization, the w(transpose)Xi had lambda*0.5*w_norm^2 added to it.
C.

Lamda	Training Converge %	Testing Converge %
64	50	50
32	70.4	68.4
16	89.7	88.5
8	96.9	95.6
4	98.8	97.25
2	99.2	97.4
1	99.5	97.5
0.5	99.92	97.875

0.25	100	98
0.01	100	97.875
0	100	97.875

As can be seen in the above table, super high lambdas that indicate heavily penalizing higher weights eventually cause the of fit to be a straight line dividing the points in half, leaving a 50% accuracy rate. As lambda gets smaller, the accuracies increase because high weights are less penalized so the curve can fit the data more accurately. Eventually this will cause a 100% fit for the training set as it can create a curve using weights that correctly guess each target. However, this is impossible for the testing set as the weights are from the training set, and therefore the weights will not be perfectly fitted to the testing data set.