Parker Bruni and Brian Blythe
CS434
4/7/2018

Implementation assignment 1

PART 1: LINEAR REGRESSION

1.1.
```
w =
[[ 39.584]
 [ -0.101]
 [  0.046]
 [ -0.003]
 [  3.072]
 [-17.225]
 [  3.711]
 [  0.007]
 [ -1.599]
 [  0.374]
 [ -0.016]
 [ -1.024]
 [  0.01 ]
 [ -0.586]]
```

1.2.
```
Training ASE = 22.081

Test ASE = 22.638
```

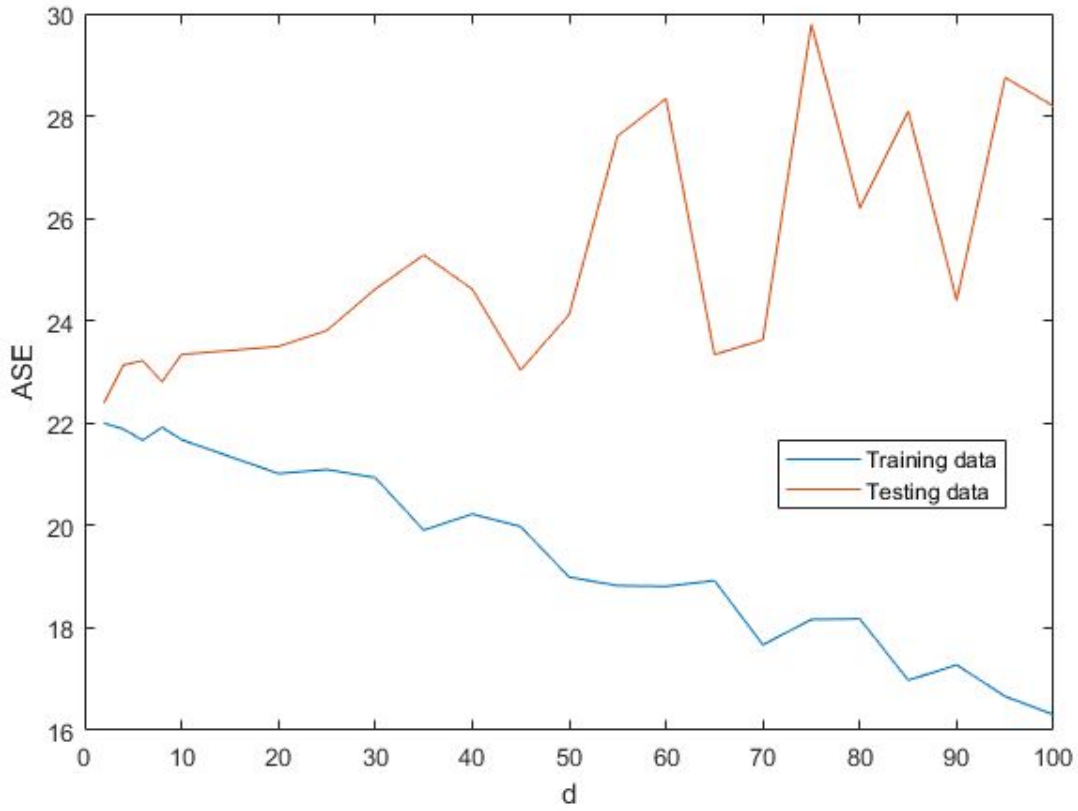
1.3.
```
w =
[[ -0.098]
 [  0.049]
 [ -0.025]
 [  3.451]
 [ -0.355]
 [  5.817]
 [ -0.003]
 [ -1.021]
 [  0.227]
 [ -0.012]
 [ -0.388]
 [  0.017]
 [ -0.485]]
```

```
Training ASE without dummy variable = 24.476

Test ASE without dummy variable = 24.292
```

The ASE for the training and test data change significantly without the dummy variable included in the calculations. Without the dummy variable, we are essentially removing the consideration of the offset in our objective calculation, thus changing our calculation to produce a function that will fit without that offset consideration.

1.4.

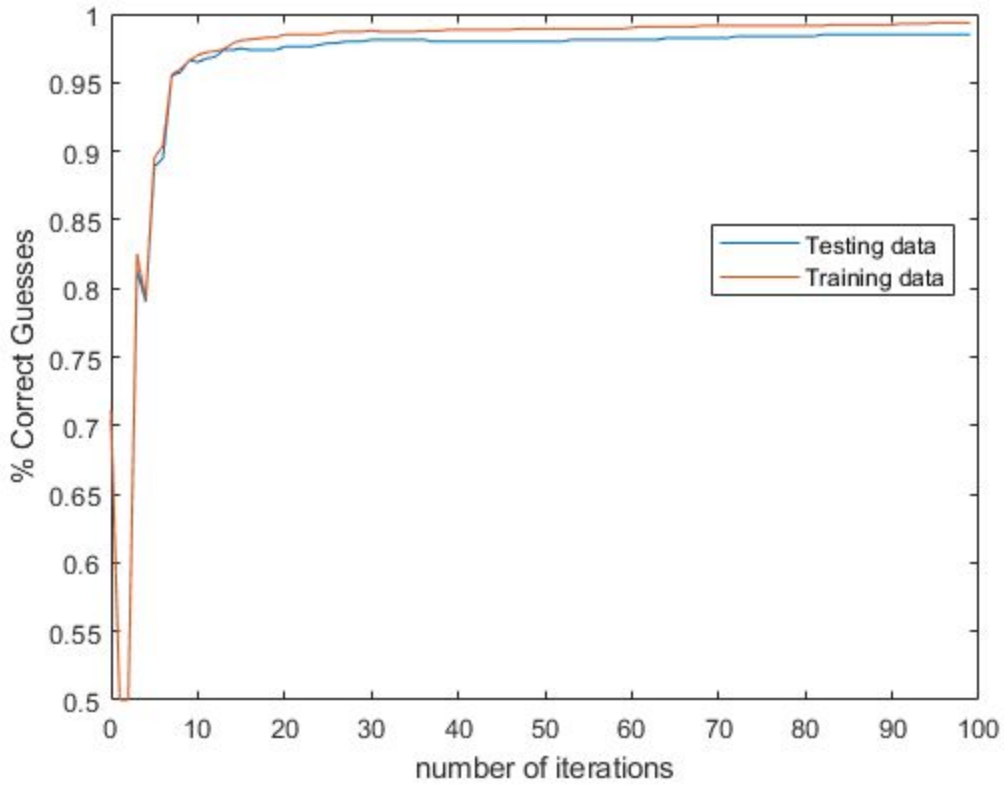


As more random features are added, the ASE for the training data reduces, but the ASE for the testing data tends to increase. This happens because as more random features are added, the training algorithm will fit to their added noise better for every feature added while the testing data, since it is uncorrelated to the noise, will diverge from the trained regression for every feature added.

PART 2: LOGISTIC REGRESSION WITH REGULARIZATION

2.1

As per the graph below, it appears that our model rapidly approaches a high success rate for both the training data and the testing data, which begins to level off after about 10 iterations around the 95% success rate. For both sets of data, the rate that the success rate increases eventually begins to level off. The training data levels off around 99% success rate and the testing data levels off at about 98.5% success rate. This suggests that the training data is better fit to the model, which is to be expected. When experimenting with different learning rates we noticed that a higher learning rate value led to a faster computing algorithm but the values tended to overflow. A lower learning rate value led to a more accurate, slower computing algorithm



2.2. Math and Pseudocode

$$l(w) = -\sum_{i=1}^{n} \left(y_i \log \sigma(w^T x_i) + (1-y_i) \log(1-\sigma(w^T x_i))\right) + \lambda \sum_{i=1}^{n} w_j^2$$

$$\nabla l_i = \frac{-y_i}{\sigma(w^T x_i)} \nabla \sigma(w^T x_i) + \frac{(1-y_i)}{(1-\sigma(w^T x_i))} \nabla \sigma(w^T x_i) + \lambda 2w$$

$$\nabla l_i = (\sigma(w^T x_i) - y_i) x_i + \lambda 2 w$$

Psudo code

1. Compute ∇l_i for each row in X, Y
   a. Compute $\hat{y}$
   b. Compute norm as $\lambda 2w$
   c. Compute grad as grad + $(\hat{y} - y_i) x_i + 0.5 \lambda$ norm
2. $w = w - \alpha \cdot$ grad

2.3

Modified Algorithm Data:

Training Performance Success Rate After 100 Iterations:

λ = 10^-3:	0.994
λ = 10^-2:	0.987
λ = 10^-1:	0.674

Testing Performance Success Rate After 100 Iterations:

λ = 10^-3:	0.985
λ = 10^-2:	0.980
λ = 10^-1:	0.655

Smaller λ values show improved performance success when compared to higher λ values. This is because the larger λ gets, the larger the normalization has an effect on the logistic regression, essentially reducing the possible variations in the regression. Though it isn't shown in our data due to computation time constraints, we would expect to see the testing performance reach of point where it started to decrease with smaller λ values. This is because at a certain point, the regression would start to fit to the noise of the signal instead of to the general shape of it, overfitting the data.