CS 434 Implementation 1

John Warila, Joshua Barringer
Prof. Fern

11 April 2020

1 Linear Regression

1.1

For the first 2 parts, we calculated the linear regression on the data we read in from the data files. We used numpy to perform the weight calculations.

Parameter	Learned Weight
Dummy Var	36.71
CRIM	-0.11
ZN	0.04
INDUS	0.01
CHAS	4.04
NOX	-18.12
RM	3.91
AGE	-0.00
DIS	-1.62
RAD	0.35
TAX	-0.01
PTRATIO	-0.89
B	0.01
LSTAT	-0.59

The results match what we would expect to see from this data. The largest weight of features is NOX (nitric oxides concentration), which, according to our model, drastically reduces the value of a house. The next non-dummy variable with the most weight is CHAS (if the house bounds a river, which increases the value of a house). The feature with the least weight was AGE (proportion of owner-occupied houses build before 1940).

1.2

$\boxed{TrainingASE = 21.636}$

$\boxed{TestingASE = 23.690}$

Our results for linear regression are about what we expected them to be. The inclusion of the dummy variable was very important to allow our trained weights to represent a function that is not centered at (0,0). The difference between the training and testing ASE is small, with the testing ASE only having slightly more error than the training.

1.3

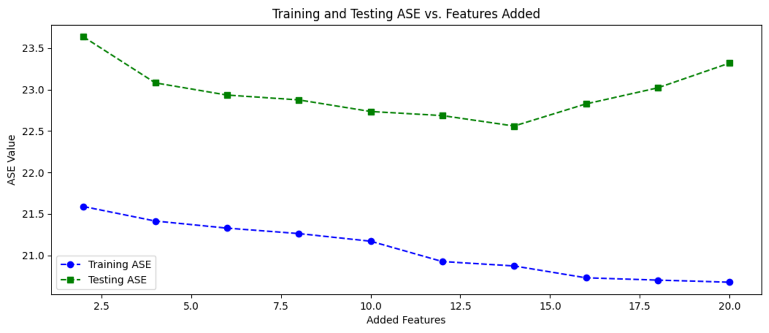
Parameter	Learned Weight
CRIM	-0.10
ZN	0.04
INDUS	-0.01
CHAS	4.44
NOX	-2.94
RM	5.98
AGE	-0.01
DIS	-1.08
RAD	0.19
TAX	-0.01
PTRATIO	-0.33
B	0.01
LSTAT	-0.48

$\boxed{TrainingASE = 23.963}$

$\boxed{TestingASE = 25.939}$

For part 3, we removed the dummy variable column from our features array. Because of this, the weights changed slightly. The average squared error for increased by roughly 2.5 for both the training error and the testing error. The dummy variable is clearly an important part of correctly modeling linear regressions.

1.4



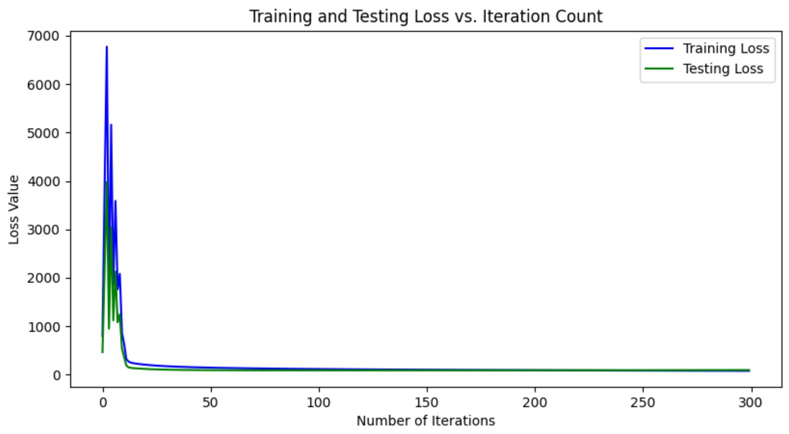
The plot above shows the testing and training ASE's on the Y axis, with the number of randomly sampled added features on the x axis. We ran this script multiple times, and found that the testing error acts mostly randomly, while the training data always reduces error with more random features. For the random normal sampling, we used a μ (mean) of 10 and a σ (standard deviation) of 5. As we add more random features, the training data reduces error because of the increase in parameters. The testing error does not decrease because it doesn't have any real connection to what the model trains with.

2 Logistic regression with regularization

2.1

For our implementation of batch gradient descent we chose a stopping condition of running a certain number of epochs, in our case we use 300 epochs. We chose this after implementing a condition that stop after the norm of Δ reached a small ϵ, this is for ease of graphing our loss function.

We tried several learning rates from 0.0001 to 10. We found that a learning rate less than 0.01 to work best with our setup. For all subsequent parts we use a learning rate of 0.001.



In the above graph the cross entropy loss is plotted as a function of the number of iterations (epochs) performed. We found that after around 50 iterations the decrease in loss flattens out, and once we reach 200-250 iterations there is minimal improvement.

2.2

The gradient of

$$L(\boldsymbol{w}) = \sum l(g(\boldsymbol{w}^T \boldsymbol{x}^i), y^i) + \tfrac{1}{2}\lambda|\boldsymbol{w}|^2$$

is:

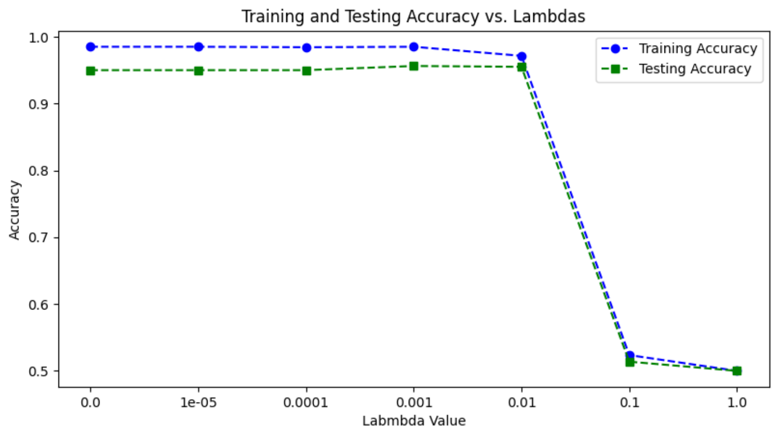
$$\Delta l(\boldsymbol{w}) = (\sigma(\boldsymbol{w}^T \boldsymbol{x}_i) - y_i)\boldsymbol{x}_i + \lambda \boldsymbol{w}$$
$$\Delta L(\boldsymbol{w}) = \sum(\sigma(\boldsymbol{w}^T \boldsymbol{x}_i) - y_i)\boldsymbol{x}_i + \lambda \boldsymbol{w}$$

We use the above gradient to generate new pseudo-code for the batch gradient descent with l2 norm.

```
Given: training examples (x_i, y_i), i = 1,...,n
let w = (0,...,0)
For e number of iterations:
     delta = (0,...,0)
     for i = 1,...,n:
          y_hat = 1 / (1 + exp(-W.transpose().dot(x_i)))
          delta = delta + (y_hat - y_i) + lambda * W
    W = W - learning_rate * delta
```

2.3

When implemented, our batched gradient descent l^2 normalized algorithm performed margnially better in testing than the batch gradient descent algorithm without the l^2 norm (95% compared to 95.7% when using a $\lambda = 0.001$). As we tested different values for λ we found that any lambda larger than 0.01 caused the model to perform no better than a random guess (roughly 50% accuracy). At our best tested $\lambda = 0.001$ we found a training accuracy of 98.5% and a testing accuracy of 95.7%.


Training and Testing Accuracy vs. Lambdas
Training Accuracy
Testing Accuracy
Accuracy
1e-05
Labmbda Value