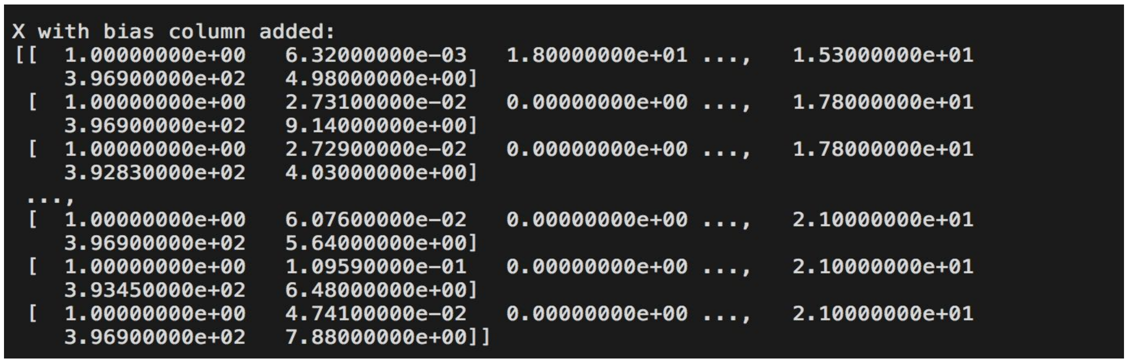
CS 434

Linear Regression

Neale Ratzlaff

1) Given the training data, load the data into the corresponding X and Y matrices, where X stores the features and Y stores the desired outputs. The rows of X and Y correspond to the examples and the columns of X correspond to the features. Introduce the dummy variable to X by adding a column of ones to X.
    - After loading in the data from the file, a bias column of ones was added as the first column

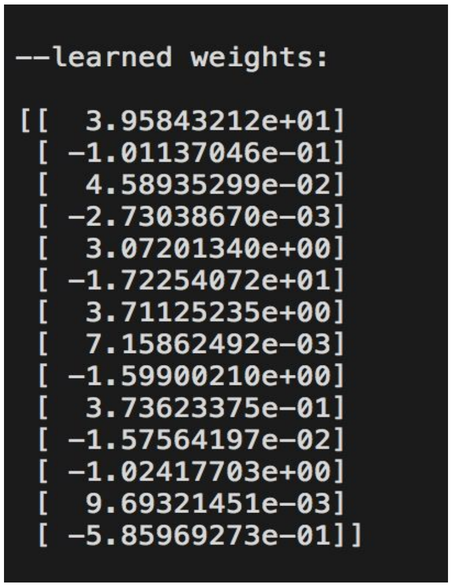


```
X with bias column added:
[[  1.00000000e+00   6.32000000e-03   1.80000000e+01 ...,   1.53000000e+01
    3.96900000e+02   4.98000000e+00]
 [  1.00000000e+00   2.73100000e-02   0.00000000e+00 ...,   1.78000000e+01
    3.96900000e+02   9.14000000e+00]
 [  1.00000000e+00   2.72900000e-02   0.00000000e+00 ...,   1.78000000e+01
    3.92830000e+02   4.03000000e+00]
 ...,
 [  1.00000000e+00   6.07600000e-02   0.00000000e+00 ...,   2.10000000e+01
    3.96900000e+02   5.64000000e+00]
 [  1.00000000e+00   1.09590000e-01   0.00000000e+00 ...,   2.10000000e+01
    3.93450000e+02   6.48000000e+00]
 [  1.00000000e+00   4.74100000e-02   0.00000000e+00 ...,   2.10000000e+01
    3.96900000e+02   7.88000000e+00]]
```

Figure 1: Training data with bias column as column 1

2) Compute the optimal weight vector w using $W = (X^T X)^{-1} X^T Y$. Most programming languages have numerical packages that you can directly use to perform the computation. You don't need to implement your own matrix inversion function. Report the learned weight vector.

    - We see that the learned weight vector has dimensions (, 14) like we expect from 14 dimensional training data



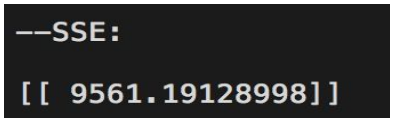
```
--learned weights:

[[  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]]
```

Figure 2: The learned weight vector

3) Apply the learned weight vector to the testing data and compute the sum of squared error(SSE) on the testing data. Report the SSE value.

   ○ SSE was applied in the compacted form of $(Y-XW)^T(Y-XW)$



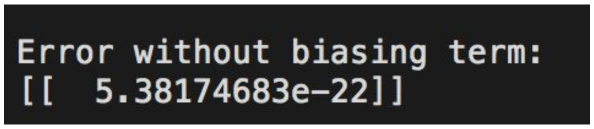
```
--SSE:

[[ 9561.19128998]]
```

Figure 3: Calculated SSE

4) Consider the situation where we do not introduce the dummy variable to X, repeat 2 and 3. How does this influence the performance as measured by testing data SSE?

   ○ SSE was recalculated using the above method. The X data was changed to exclude the bias term. The resulting weight vector is extremely small, on the order of 10^-14. Without the bias term, the system only takes extremely small steps in the right direction, and the error was correspondingly small.



```
Error without biasing term:
[[  5.38174683e-22]]
```

Figure 4: SSE on X without a biasing term

5) Consider a variant of linear regression, where the optimal weight is computed as $W=\left(X^TX+\lambda I\right)^{-1}X^TY$, where I is the identity matrix of the same size as X^TX and λ is a user specified parameter for learning. Compute the optimal w using this formula with different values of λ (e.g., 0.01, 0.05, 0.1, 0.5, 1, 5. Feel free to explore more choices.)

Evaluate each of the learned w by computing the SSE on the testing data. Plot the SSE value as a function of λ. What behavior do you observe? What do you think is the best λ value for this problem?

- I calculated a new weight vector and its SSE for the learning rates:
  λ = [.000001, .001, .01, .05, .1, .5, 1, 10, 100, 1000, 100000, 10000000]
  I plotted the calculated SSE against the learning rates on a log scale and found an exponential increase in error as the learning rate increased. With a lower bound at a step size of 0.01. Smaller step sizes saw a decrease in error, but not by enough to offset the increase in runtime that a smaller step size demands.

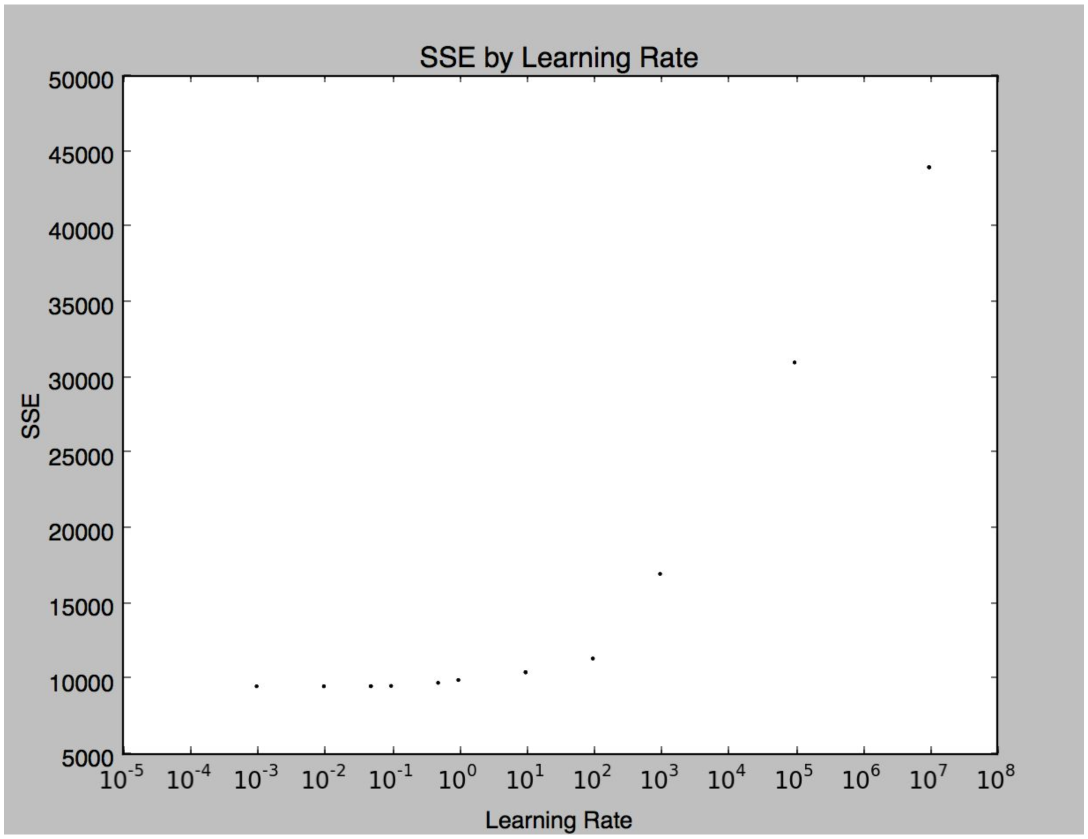


Figure 5: Log plot of SSE against learning rate

6) Compare the different w's that you got in 5. As the λ value gets bigger, what impact do you observe it has on the weight values?

- The weights became increasingly small for each increase in learning rate, decreasing by at least four orders of magnitude from learning rate values: λ = 10^-3 to 10^8

7) This variant that we introduce is the solution for minimizing the following modified objective: $\sum_{i=1}^{n}(y_i - W^T - x_i)^2 + \lambda|W|^2$ is the regular SSE and the second term is called a regularization term and computes the norm of the weight vector w. Can you use this objective to explain the behavior that you observe in [6]?

   - The purpose of regularization is to penalize large weights. A small learning rate will counteract large weights and cause the function to take longer to converge. By squaring the norm (euclidian difference), the cost function here harshly penalizes weights that are larger. This causes the effect seen in the data where the weights tended toward 0, where the step size was too big and the error increased exponentially.