CS 334 — Homework 3 Solutions

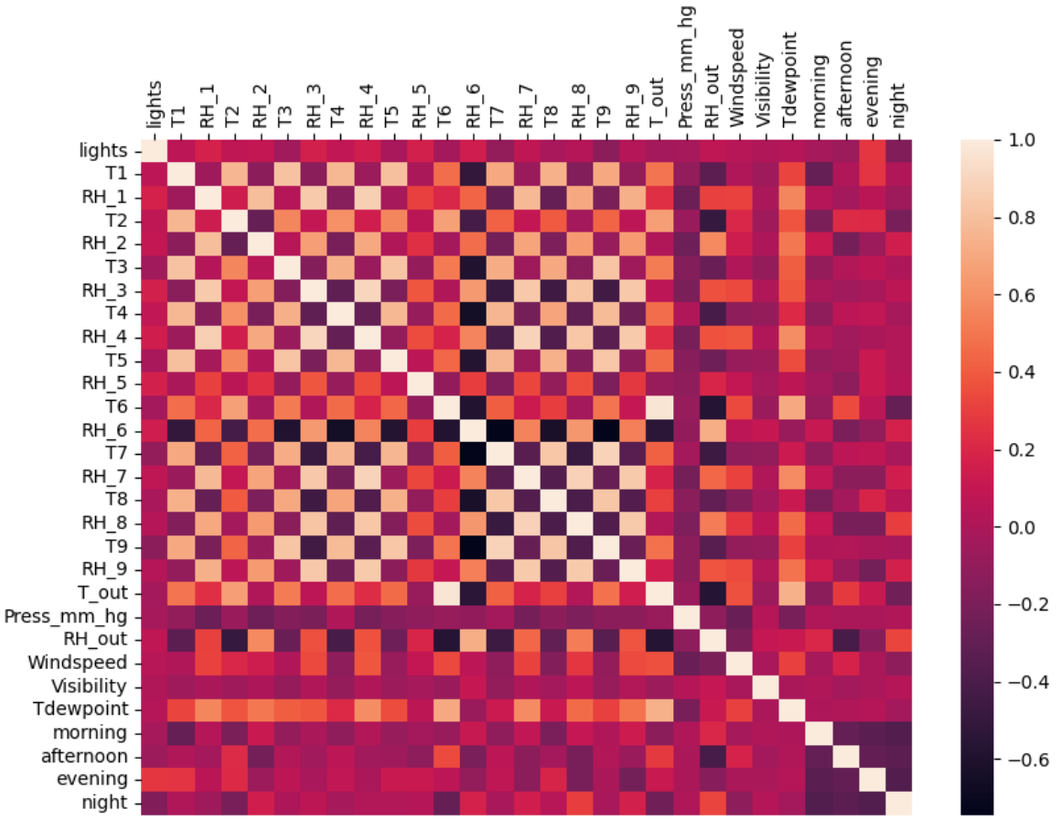
Jack Anstey

Problem 1: Feature Extraction + Feature Selection

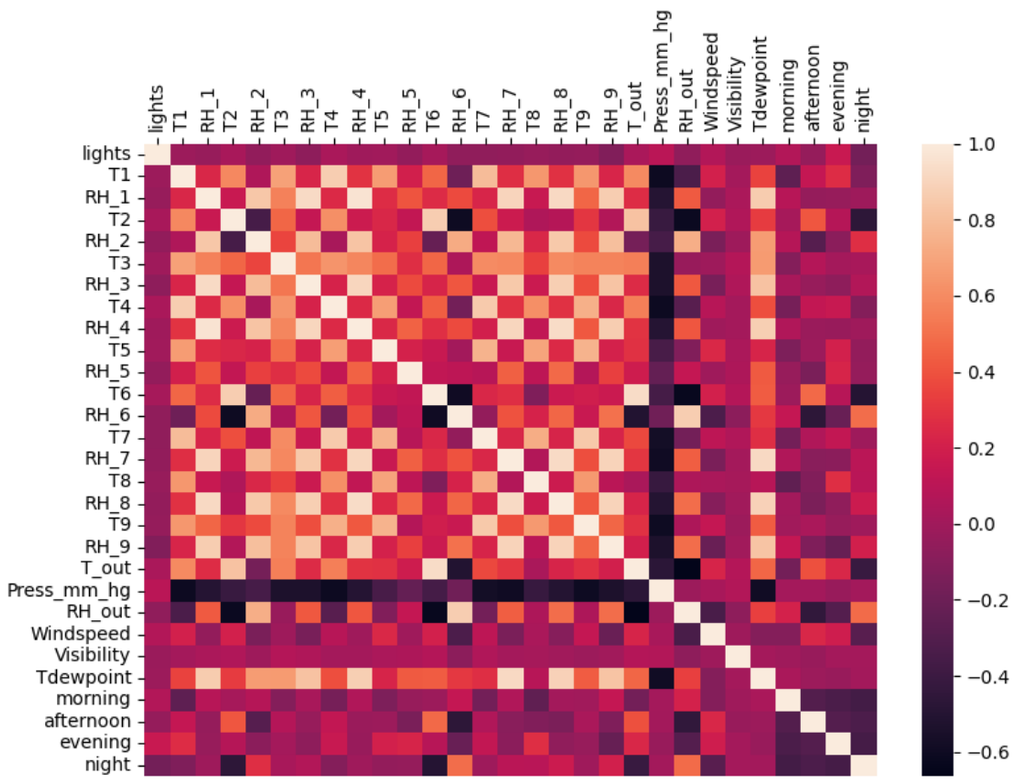
a) See selFeat.py

b) The training and testing heatmaps are as follows:

Training:



Testing:



c) Given the columns found through doing the Pearson Correlation, I went through and excluded any column that contained any values that were greater than 0.5, or less than -0.5. The remaining columns were then used to define a smaller dataframe (excluding the highly correlated columns.)

We do not want columns that are highly correlated with each other and we know that a column is highly correlated when the Pearson Correlation returns a value greater than 0.5 or less than -0.5. In selFeat.py, you can see that I do my feature selection programmatically, so potentially any dataset could be used and the features can by pruned dynamically.

d) The preprocessing steps I chose to do started with making sure the columns that I kept from the previous step were matching between the training and test datasets. I found that they did not, so I wrote a short algorithm that deleted extra columns and therefore made the features match between the training and test datasets.

My last preprocessing step I chose to do was to normalize the data. I used the standard scalar from sklearn to create a scaler for the training dataset and I then applied it to both the training and test dataset. With that, the preprocessing of the data was done and it was output as new_xTrain.csv and new_xTest.csv.

Problem 2: Linear Regression: Single Unique Solution

a) See standardLR.py

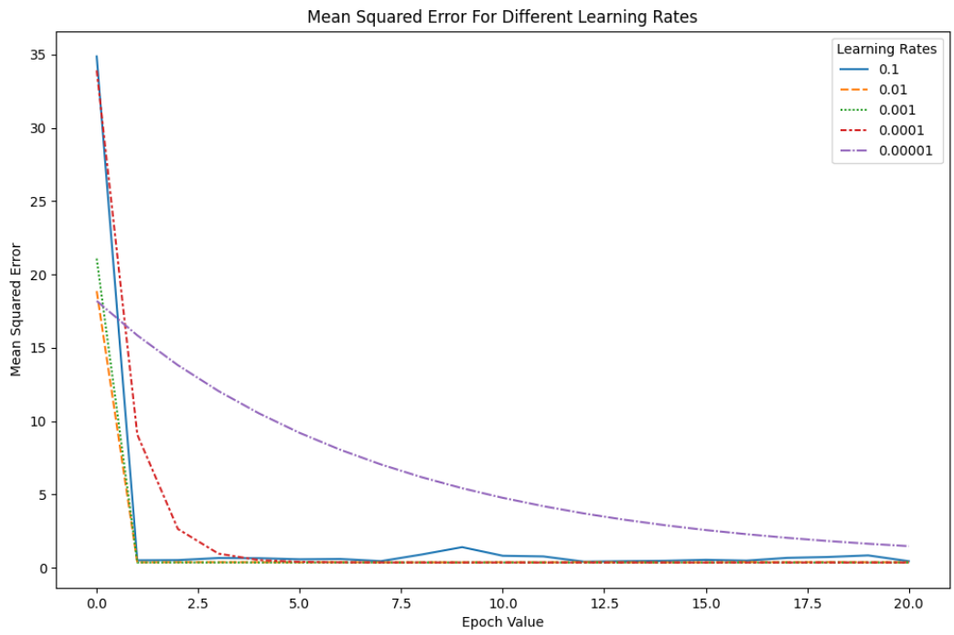
b) After implementing the closed form solution in standardLR.py, I found running the solution to have the following output:

```
{0: {'time': 0.003002643585205078, 'train-mse': 0.37595469803815945, 'test-mse': 0.29006172232577737}}
```

Problem 3: Linear Regression using SGD

a) See sgdLR.py

b) For this plot, I used the learning rates of 0.1, 0.01, 0.001, 0.0001, and 0.00001. I chose to not use a learning rate of 1 as beta goes to infinity quite quickly and would make the graph unreadable.

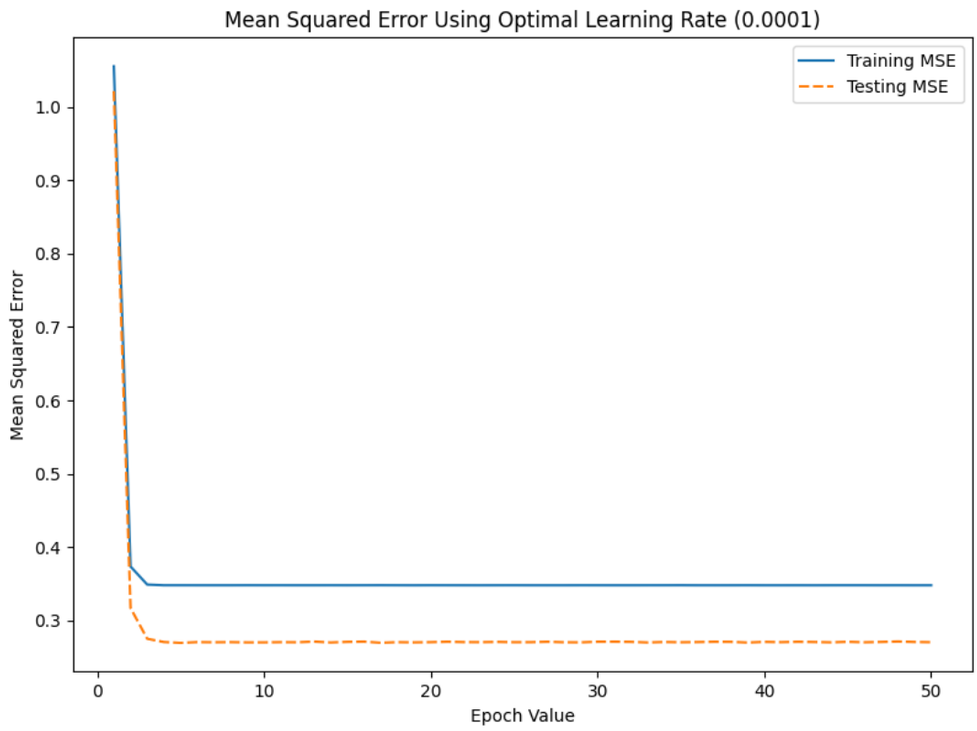


As we can see from the plot, the learning rates of 0.1 through 0.001 are extremely aggressive, going to their final mse after just one epoch. 0.1 is especially problematic as it tends to diverge at an epoch of around 8.

The learning rates of 0.0001 and 0.00001 however take some more time, curving gradually towards the converging mse value.

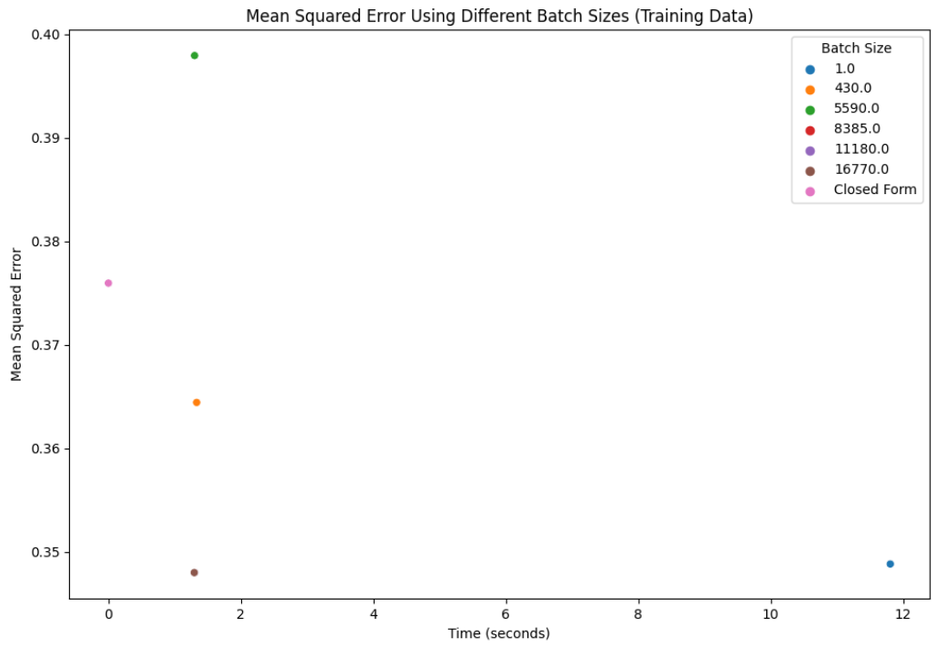
The best learning rate for a batch size of 1 is then 0.0001. 0.00001 may also work, but as shown on the graph it doesn't converge even after 20 iterations. 0.0001 has the best balance of speed and accuracy, which is why I will be using it for comparing the training and test mse in the next sub-problem.

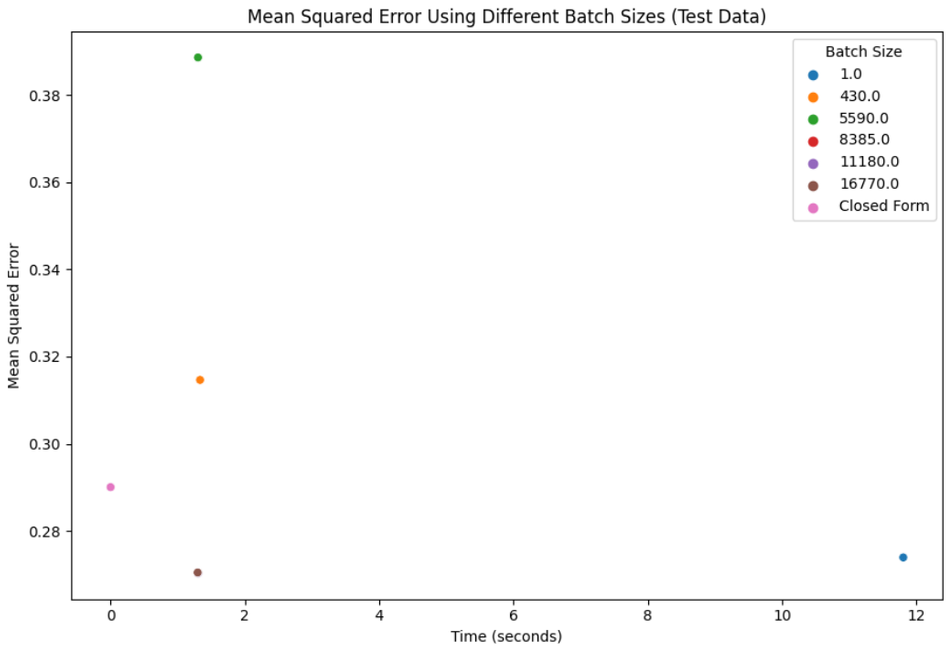
c) Armed with the knowledge that a learning rate of 0.0001 is the best when using a batch size of 1 with this dataset, I was able to create the following plot:



Problem 4: Comparison of Linear Regression Algorithms using SGD and Closed Form solutions

a) The plots comparing batch sizes, times, and accuracies are as follows:





b) When looking at these plots, I was surprised at both the incredible speed and relative lack of accuracy when using the closed form solution. Using numpy's implementation of matrix inversion, transposition, and multiplication is clearly really quick, with a time that is so fast that it basically looks like it's 0 seconds flat on the graph.

The mean squared error doesn't look quite as good, however. It is clearly better compared to some of the batch sizes, but for both training and test datasets, a batch size of 1, 2/3 N, and N (where N is the entire dataset) clearly yield better results. These results are at the expense of of time.

Using a max epoch of 101 (for a total of 100 iterations), the differing batch sizes all took more time than the closed form solution and some significantly more time. A batch size of 1 took the most amount of time, while the others all took just under 2 seconds to complete.

What sets these batch sizes apart is not the time then, but the mean squared error. The relationship of batch size to mean squared error seems to be on a curve regarding both the training and test datasets. A batch size of 1 and N both gave very low mean squared errors as stated before, but going up from 1 yielded mixed results. A batch size that was half of the dataset had the worst mean squared error, while 430 lies inbetween. Once the batch size reaches 2/3 N, the mean squared error seems to converge which is why you cannot see that datapoint: it perfectly overlaps with the N batch size datapoint.

Overall, the biggest tradeoff between using the closed form solution is the potential for a better mean squared error value at the expense of time. To minimize this, it does seem that pretty large batch sizes (at least for this dataset), seem to strike a good balance between speed and accuracy.