# CS 334 - Homework 2

Alex Welsh

October 4th 2020

## 1 Question 1 - Feature Extraction + Feature Selection

**a.** From the original date column (e.g., 3/20/16 5:30), I separated the date and the time into two separate features. However, 3/20/16 and 5:30 are not numbers that can be used by the model so I converted the date into a day of the year. Ex. Feb 1st is the 32nd day of the year. I then converted 5:30 into minutes of the day. So 5:30AM would be 330 minutes. Now, both features are a number that can be processed. Code for this can be found in **selFeat.py**

**b.** The heat map is shown below in Figure 1. This heat map was generated with all of the features, including the two features I added in question 1a, but before I removed the features that were not very correlated. This training data was stored in **new_xTrain-pre-select-features.csv** and the code that generated the heat map can be found in **q1b.py**.

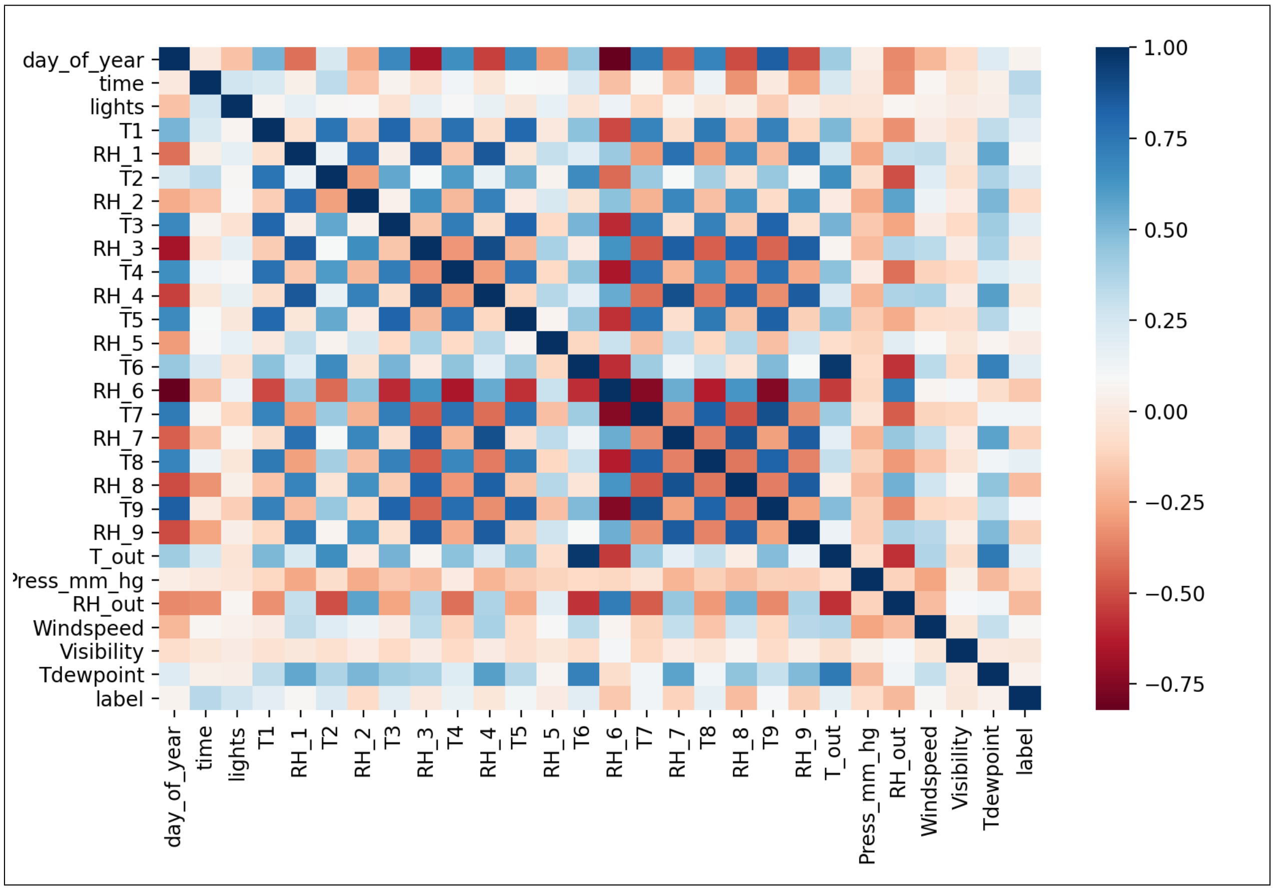

**Figure 1. Heatmap of Pearson correlation between all the features in the training data.**

**c.** I selected 'time', 'lights', 'T2', and 'RH_out' as my features that I wanted to keep. This was because these features had Pearson correlation values that were either greater than .2 or less than -.2. I wanted to pick the features that were most highly correlated with the label and these 4 features had relatively high correlations with the label compared to the other feature's correlations with the label. The code that indicates these 4 features pass this threshold can be found in **q1b.py**. The code that actually extracts the features is in **selFeat.py** and they were manually selected for.

**d.** I applied a MinMaxScaler to the remaining features as my pre-processing step. This code can be found in **selFeat.py**. The result of **selFeat.py** is two files, **new_xTrain.csv** and **new_xTest.csv** which have the pre-processed, extracted and selected features and are used in the remaining work.

## 2 Question 2 - Linear Regression: Single Unique Solution

**a & b.** The code for the predict function can be found in **lr.py** and the code for train_predict can be found in **standardLR.py** The train stats are returned for the closed form solution and for a time of 0.002901, the train-mse is 0.370803 and test-mse is 0.3023047.

## 3 Question 3 - Linear Regression using SGD

**a.** The code the for linear regression using SGD can be found in **sgdLR.py**. For each iteration, the iteration number, time, train-mse and test-mse are saved and then returned at the end.

**b.** I chose .001 as my optimal learning rate because it converged smoothly to one of the lowest MSE's of all the learning rates. It was also more stable than learning rates of .1 and .001 which, while also low, had jagged spikes which I felt were not ideal (Figure 2.). The code that generates this plot can be found at the bottom of **sgdLR.py**. There is a comment that says "# Code for Question 3B".

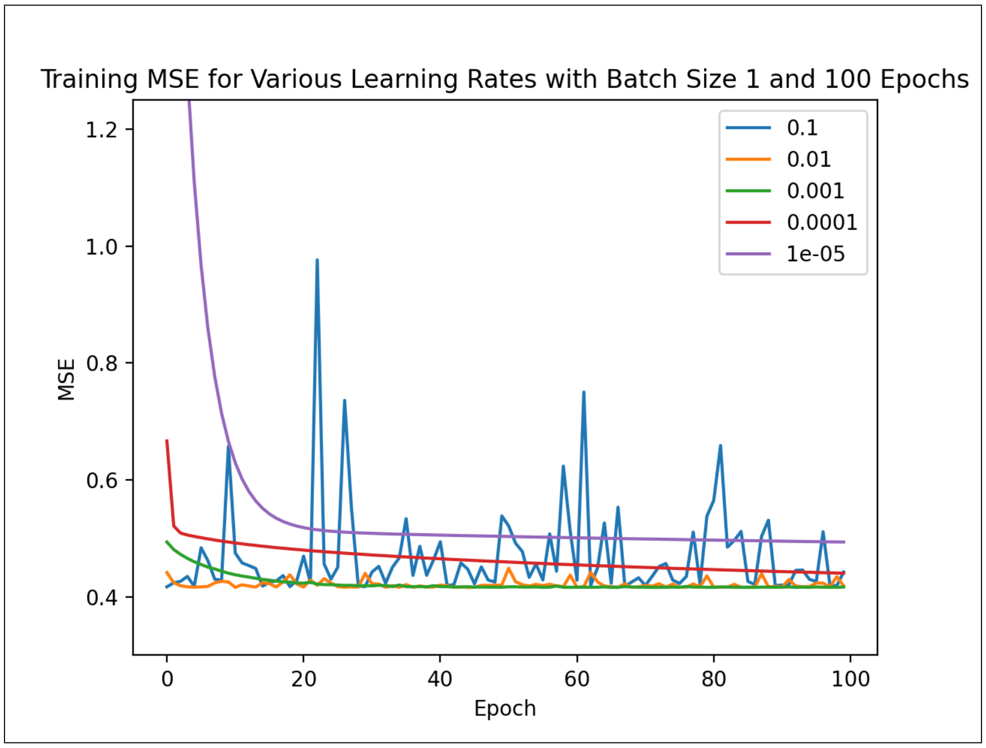

**Figure 2. Training MSE for different learning rates over 100 epochs.**

**c.** See Figure 3. The code that generates this plot can be found at the bottom of **sgdLR.py**. There is a comment that says ”# Code for Question 3C”.

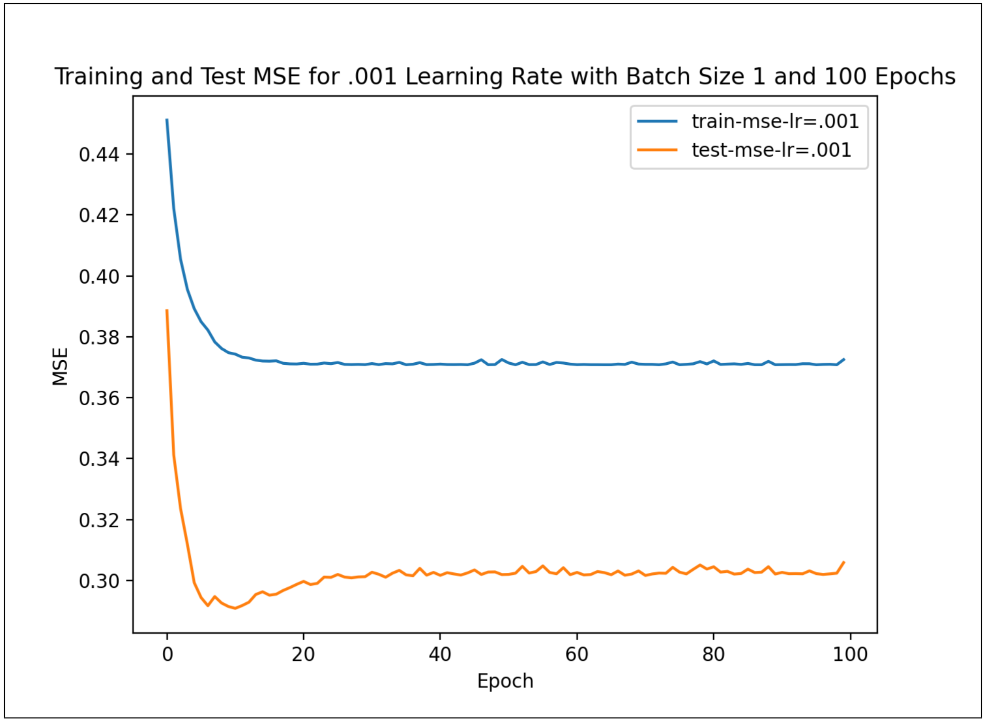

**Figure 3. Training and test MSE with a learning rate of 0.001 for 100 epochs.**

## 4 Question 4 - Comparison of Linear Regression Algorithms using SGD and Closed Form Solutions

**a.** First, I wrote some code that generates plots for each of the batch sizes I tested and plots curves of the train-mse for different learning rates. Since my batches were {1, 5, 10, 15, 26, 39, 65, 86, 129, 195, 258, len(xTrain) = 16770}, I generated 12 plots. I didn’t include these in the write up because it would have take up a lot of space. It essentially looked like Figure 2, but then for each batch size. The code to generate these plots is in **sgdLR.py**. Then, I manually inspected each plot to find the best learning rates. The array of learning rates is {.001, .01, .01, .01, .01, .01, .1, .1, .1, .1, .1, .1} and corresponds with the batch values listed above. From these values, I then plotted the total time against the train-mse and the test-mse and included the time and MSE’s for the closed form solution. This code is also in **sgdLR.py** and can be see in Figure 4.

**b.** From the plots in Figure 4, the point in the top and the far right are N and 1 sized epochs, respectively. It is clear from those points that they are not optimal compared to the close form solution and other batch sizes because their MSE is too high or they take to long when other batch sizes will give comparable MSEs. It is also evident that mini-batches are much better to get the same MSE values compared to the closed form, although they do take about 4-6 seconds, whereas the closed form is very fast. The only limitation of the closed form is that some of the mini-batches (ex, bs=39) will give a test MSE that is slightly lower than the closed form.

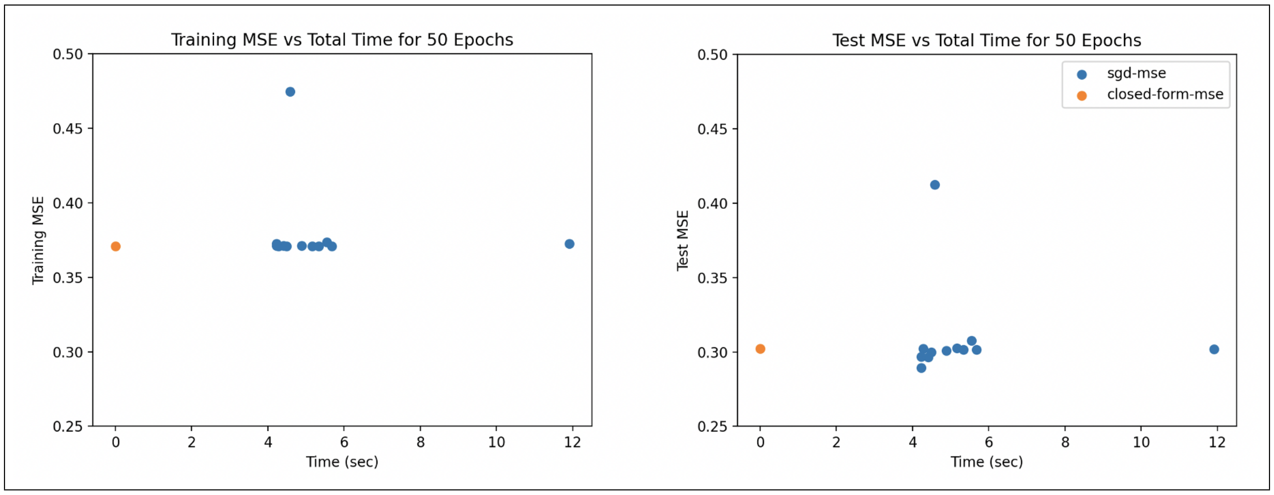

**Figure 4. Training and test MSE vs time for different batch sizes and optimal learning rate compared with closed form solution.**