Assignment 2 – Report

**Implementation Decisions**

Choosing alpha value – When determining what alpha value to use for this part of the assignment I decided to set a static alpha value instead of adjusting the alpha value throughout the linear regression process. This is because it was very simple to find an alpha value that would hit the benchmarks after experimenting with a few different alpha values. The alpha value that I decided to use was .001 for the wine dataset, .0025 for synthetic1, and .025 for synthetic2, which allows me to always hit my benchmarks.

Updating the weights (theta values) – An implementation decision that I had to make regarding updating my theta values was to decide what was the best way to stop updating the theta values. The way I decided to stop my theta values from updating was to update the theta values a certain amount of time steps and once that number of time steps has been reached the theta values will no longer update. I chose this method to stop my theta values from updating because it allows me to control how long my linear regression will run.

Full batch gradient descent – In this assignment I chose to use full batch gradient descent. This is because full batch gradient descent allows me to calculate only two dot products, one dot product for the examples and theta values and another dot product for the singular examples’ values and the feature values for that example, for every update that occurs. This method allows my program to be simpler and increase its runtime.

Normalizing the data – Since the different features of the wine dataset may be on various scales, I had to find a way to normalize the data for every feature. The way I chose to normalize this data is to shrink every feature and its values down to a scale of 0 to 1, this way every feature is on the same scale. To implement this method, I took every example which respect to every feature and found the maximum value for each feature and the minimum value for each feature. Then I took each example for each feature and subtracted that value by the minimum value of the feature and divided that calculation by the maximum value of the feature minus the minimum value of the feature.

Initializing theta values (weights) – When implementing how I would initialize the theta values I decided to use the np.uniform function. This is because the np.uniform function allows me to set a range of where I want my random theta values to be in and a set number of values that I need initialized. When using the np.uniform function, I used the range of -1 to 1 so my theta values could have both positive and negative values.

Differences in part2.py from part1.py – There are a few differences between part2.py and part1.py such as in part 2 I did not need to normalize the data so that implementation is not included in part 2. In part 2 we must compute various orders of the feature values, so I created a function (newOrder) that takes the original feature column and creates new columns of data based on what order of the feature we want to go up to. Part 2 also goes through both synthetic data files and computes the 2nd, 3rd, and 5th orders for each file. Another addition to part 2 is the implementation of the createGraph and createFunction functions which allow me to plot my regression lines.

**Results**

**Part 1:**

**winequality-red dataset**

Mean Squared Error: 0.7570802928840573

Weight Values: [ 2.86846018 1.09859878 1.21889659 0.48093228 0.05573919 0.76352919

0.60286839 0.70262319 0.67423457 2.19083806 -0.18720429 1.01424054]

**Part 2:**

**Synthetic1 datase**t

Order: 2

Mean Squared Error: 30.405389929432854

Weight Values: [-4.42036393 1.90027643 0.67674165]

Order: 3

Mean Squared Error: 8.940334637588576

Weight Values: [-3.96780218 11.0510664 0.87886638 -3.79830857]

Order: 5

Mean Squared Error: 9.026579906974872

Weight Values: [-3.29279847 7.88556248 -1.37096554 -0.19914355 0.70181721 -0.81391345]

**Synthetic 2 dataset**

Order: 2

Mean Squared Error: 0.3276429479455292

Weight Values: [ 0.37024944 -0.04780947 -0.17884706]

Order: 3

Mean Squared Error: 0.32761960364296255

Weight Values: [ 0.3694769 -0.05770507 -0.17811546 0.00455446]

Order: 5

Mean Squared Error: 0.3019094330287211

Weight Values: [ 0.48473775 -0.37175242 -0.52949056 0.32298261 0.11045078 -0.06525566]

**Visualizations**

Chart, scatter chart

Description automatically generatedChart, scatter chart

Description automatically generated

Chart, scatter chart

Description automatically generatedChart, scatter chart

Description automatically generated

Chart, scatter chart

Description automatically generatedChart, scatter chart

Description automatically generated

**BONUS**

**Results:**

Filename: synthetic-1.csv

Order: 5

Lambda Value: 0.1

Mean Squared Error: 8.818631913280385

Weight Values: [-3.29845802 8.35426125 -1.3011571 -0.72113847 0.67704419 -0.69873715]

Filename: synthetic-2.csv

Order: 5

Lambda Value: 0.1

Mean Squared Error: 0.3019205708069234

Weight Values: [ 0.48168706 -0.36230563 -0.52342652 0.31437095 0.10882413 -0.06357885]

**Visualizations:**

Chart, scatter chart

Description automatically generatedChart, scatter chart

Description automatically generated

When experimenting with the L2 Norm Regularization method I used various values for lambda, but they all produced similar results, so I decided to use the lambda value of 0.1 for the report. Comparing the results from the L2 norm regularized 5th order polynomial models vs the results of the unregularized 5th order polynomial models I determined that the using the L2 norm regularization produced little to no benefit. This can be seen by looking at the mean squared error and plots for each regularized model as they are almost the exact same as the unregularized models. This may be because the weight values are already relatively small so the L2 norm regularization may have little effect in shrinking these values. When I was experimenting with different lambda values if I used a really high lambda value it would make my MSE worse.