CS460G MACHINE LEARNING
PROJECT 2 – REGRESSIONS WRITEUP

Part 1: Linear Regression with Gradient Descent

Implementation Choices:

- Continuous red wine feature values are normalized by $x_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}$ with:
  + x_i is the i-th value of a feature (there are 11 features),
  + $\max(x)$ is the maximum value of that feature,
  + $\min(x)$ is the minimum value of that feature.
  Thus, all normalized feature values are within the range of 0 and 1.
- Then, I add the bias feature column to the beginning of the feature set as x_0 feature with all of it values equal 1.
- The weights/theta values are randomly initialized in the range of 0 and 1.
- The weights are updated using full batch gradient descent method (batch size = number of examples).
- The stopping criteria for the regression model is time, or the number of epochs, which in this part is 3000.

Mean Squared Error:

I ran the code file ***linearRegression.py*** and got these following results:

Dataset {DATA_FILE}	Learning rate {ALPHA}	Iterations {NUM_EPOCHS}	Mean Squared Error	Side-by-side Prediction vs Key Comparison File
winequality-red	0.01	3000	0.5345894595911682	classified_winequality-red.csv

The linear regression model is: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_{10} x_{10} + \theta_{11} x_{11}$

The weights (theta values) of the linear regression model are:

[[2.99012564], [1.02063512], [0.03487777], [0.56491746], [0.14429516], [0.52750981], [0.58006872], [0.2043399], [0.52768081], [1.60225567], [1.30083497], [2.12935034]]

Part 2: Polynomial Regression Using Basis Expansion

Implementation Choices:

- The implementation for this dataset is similar to the winequality-red dataset, except for the data preparation. For these two synthetic datasets, I do not normalize the feature values and I expand the dataset using by basis expansion (adding “new” features by raising the original feature to the k-order). So:

+ For the 2nd-order polynomial, I added 1 new column of feature X values squared. The model then will be: $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$

+ For the 3rd-order polynomial, I added 2 new column of feature X values raised to the power of 2 and 3. The model then will be: $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$

+ For the 3rd-order polynomial, I added 4 new column of feature X values raised to the power of 2, 3, 4, and 5. The model then will be: $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 + \theta_5 x^5$

Then, I added the bias feature column (all values equal 1) to the beginning of the expanded dataset and proceeded the regression algorithm with full batch gradient descent as the previous part.

- The weights/theta values are randomly initialized in the range of -3.0 and 3.0.
- The stopping criteria for the regression model is time, or the number of epochs, which in this part is 2000.

Mean Squared Error:

I ran the code file ***polynomialRegression.py*** for each of the synthetic data files. For each synthetic dataset**, I changed the initial variables *DATA_FILE* and *ALPHA* at the beginning of the code file before running**.

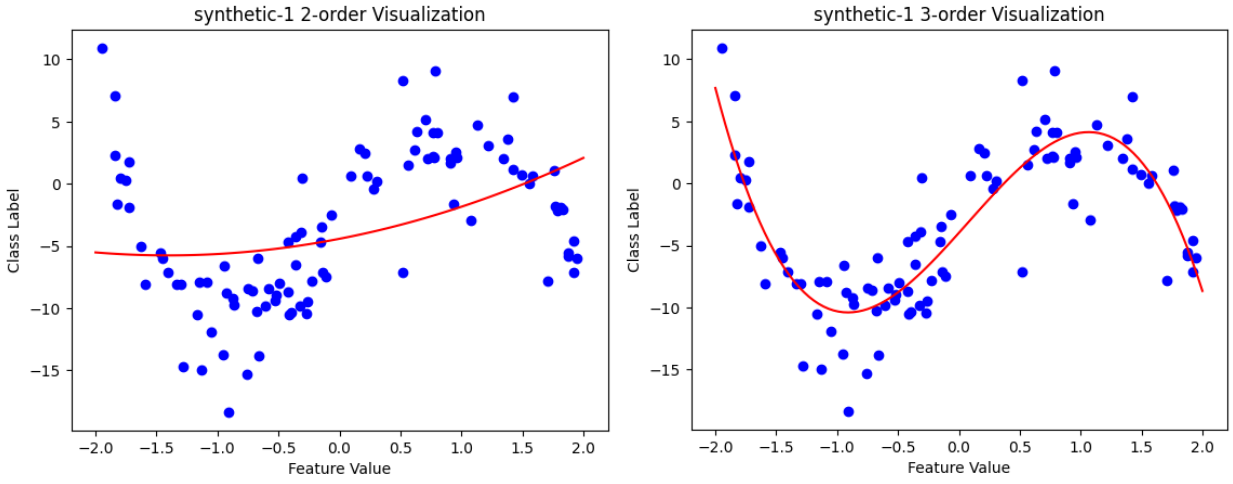
For each run, the program would call the Regression class 3 times consecutively and **stop between each order model** due to the pop-up graph **then continue creating the next model after the pop-up graph is closed manually**. After changing the parameter values, I got these following results:

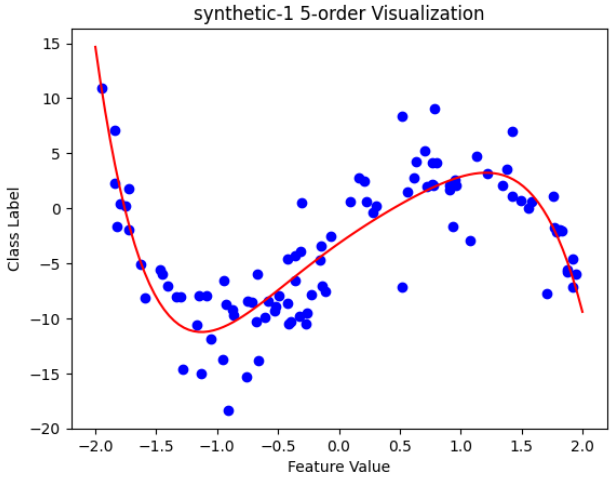
Dataset {DATA_FILE}	Learning rate {ALPHA}	Iterations {NUM_ EPOCHS}	Order	Weights & Mean Squared Error (MSE)	Side-by-side Prediction vs Key Comparison File
synthetic-1	0.01	2000	2	Weights: [[-4.41814572], [1.90014917], [0.67588914]] MSE = 30.712509552923958	classified_synthetic-1_2-order.csv
			3	Weights: [[-3.98071887], [10.95617737], [0.87372379], [-3.76078947]] MSE = 9.025855345207804	classified_synthetic-1_3-order.csv
			5	Weights: [[-3.18691497], [7.83417605], [-1.65791317], [-0.04956681], [0.77800501], [-0.85336794]] MSE = 9.056736109147046	classified_synthetic-1_5-order.csv
synthetic-2	0.007	2000	2	Weights: [[0.36303167], [-0.04931537], [-0.17333306]]	classified_synthetic-2_2-order.csv

				MSE = 0.3307816002746618	
			3	Weights: [[0.33868555], [-0.30064793], [-0.16018777], [0.0976336]] MSE = 0.3425315877890235	classified_synthetic-2_3-order.csv
			5	Weights: [[0.49694341], [-0.97541396], [-0.54265167], [0.9225592], [0.10948869], [-0.18999385]] MSE = 0.3315644390489069	classified_synthetic-2_5-order.csv

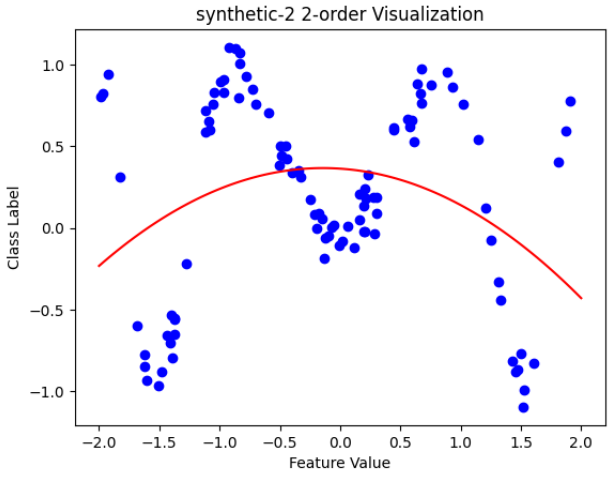
Part 3: Plot Your Regression Lines

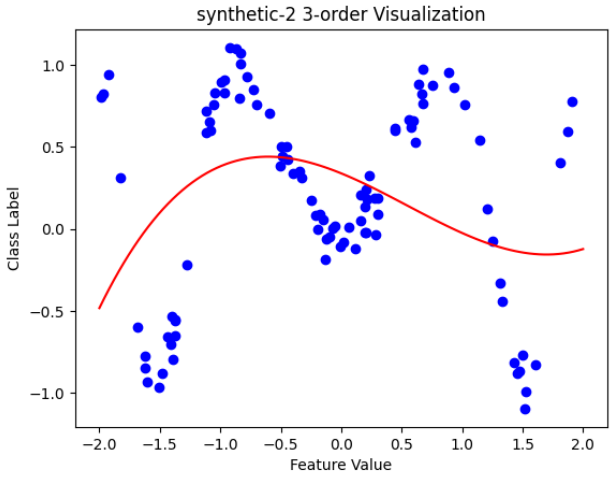
- Synthetic data visualizations as scatter plots using *matplotlib.pyplot* with the data points in blue and regression lines in red.
- I chose to graph the regression lines using the model equations based on the polynomial order (2,3, or 5), so the visualize function first validate the order of the model then create the graph accordingly.
    1. synthetic-1.csv models of 2, 3, and 5 order respectively

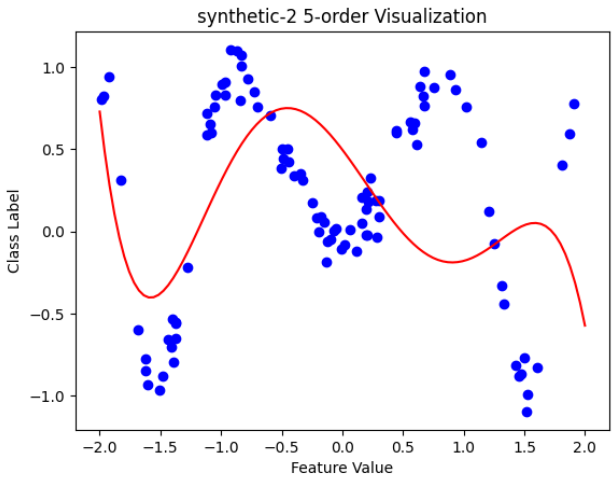




2. synthetic-2.csv models of 2, 3, and 5 order respectively







Bonus: Polynomial Regression with L2 Regularization

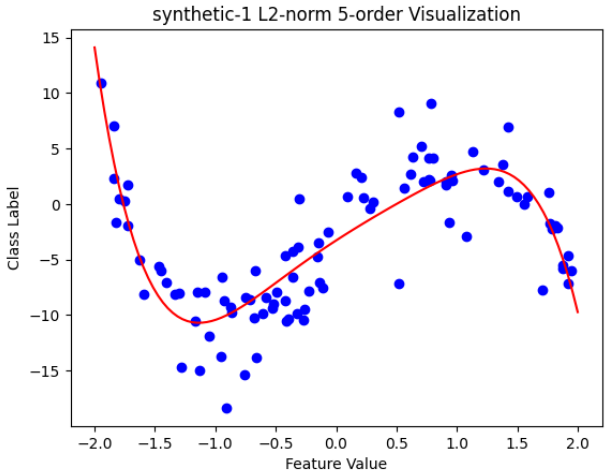
- Each synthetic dataset is expanded using basis expansion to 5th order polynomial using the same method with part 2. The program is also run with the same learning rate and epoch as part 2. The only difference is a penalty of λ is added while updating the weights.
- The gradient of j-th weight is calculated by the formula:

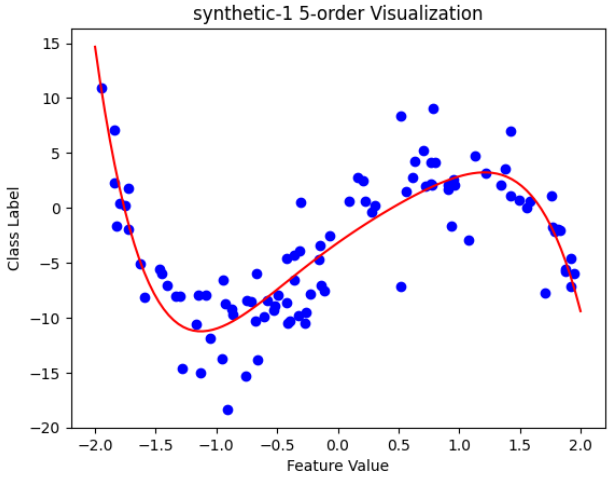
$$Gradient = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) * x_j^{(i)} + \lambda * \theta_j$$

  with λ is the penalty added to simplify the model to prevent overfitting.
  Then the weight is updated as the previous part: $\theta_j = \theta_j - Gradient$
- I ran the code file ***L2Regularization.py*** for each of the synthetic data files. For each synthetic dataset**, I changed the initial variables *DATA_FILE*, *ALPHA* and *LAMBDA* at the beginning of the code file before running**.

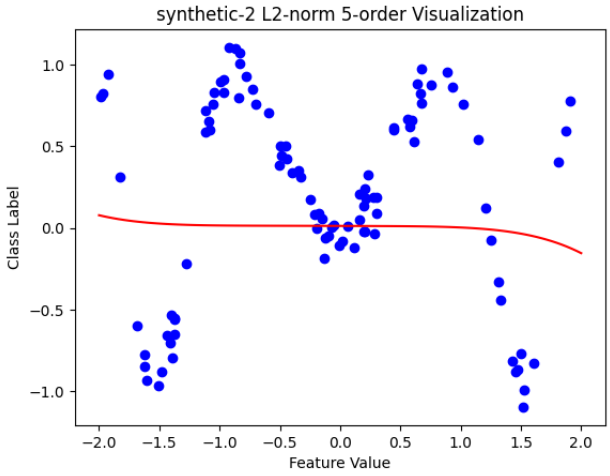
Dataset {DATA_FILE}	Learning rate {ALPHA}	Iterations {NUM_ EPOCHS}	Penalty {LAMBDA}	Weights & Mean Squared Error (MSE)	Side-by-side Prediction vs Key Comparison File
synthetic-1	0.01	2000	0.0003	Weights: [[-3.24603034], [7.27206485], [-1.28246309], [0.18531592], [0.66052224], [-0.86771505]] MSE = 9.302716313031832	classified_L2-norm_synthetic-1_5-order.csv
synthetic-2	0.007	2000	0.1	Weights: [[0.01273866], [-0.00200289], [-0.00034697], [-0.000869], [-0.00307182], [-0.0032799]] MSE = 0.3977021694522707	classified_L2-norm_synthetic-2_5-order.csv

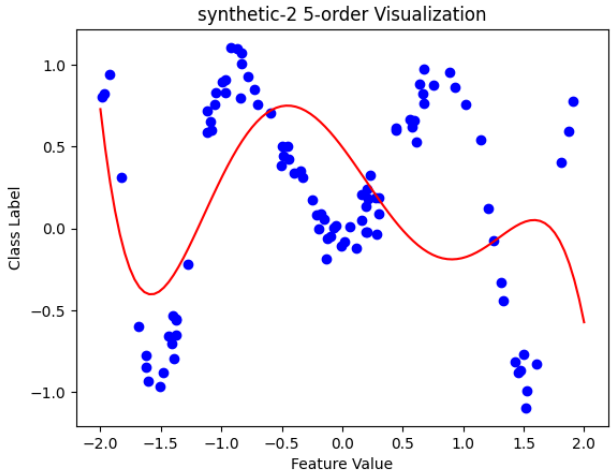
- Visualizations comparing the polynomial model with and without L2 regularization:
  1. synthetic-1.csv





2. synthetic-2.csv





- Observing these graphs, we see that the penalties (λ) added to the updates of the weights make the regression model less complex. It is not a big change in the first synthetic dataset as the λ value is relatively small. However, in the second model, we can clearly see how the regression line is a lot simpler with L2 Regularization than the one without it. Additionally, I noticed that after adding the L2 Regularization, the second model weights are stable/resulting in the same values every single run compared to the same model without it in part 2.