CS 460G Assignment 2 Report

Tyler Smith

Part 1: Linear Regression with Gradient Descent

For this assignment I created a linear regression class that had three variables along with three methods. The three variables are the model’s alpha value, number of steps the model will take before stopping, and the model’s theta values or weight values. The three methods are relatively simple and are an initialization method, fit method, and a predict method. In my fit method, I first randomly initialize my theta values using the np.random.uniform() method and then enter into a loop computing gradient descent and updating my theta values until either the number of steps have been reached or my loss is sufficiently small. I decided to use a full batch update for my theta values, mainly because I thought it was easier and less time consuming to implement. To compute gradient descent, I took advantage of the numpy matrix dot product along with other matrix operations to not have to enter any other loops. My predict method calculates my hypothesis (dot product of the data and theta values) minus the actual class label and is used in my fit method along with calculating mean squared error. For this part I used a static alpha value, set when initializing the model, and was found by just trying different values. To construct the linear regression model for the wine dataset, the data was preprocessed by making each feature between zero and one. After the data is preprocessed, the model is initialized with a step amount of 10,000, and the fit method is used on the wine data.

My final model had these values:

* Alpha = 0.001
* Mean Squared Error = 1.4991633652583343
* Weights = 2.6952334948408128, 0.3820633916602288, 1.3902601715551048, 1.5225999506089658, 0.7796109559668474, 1.1864675230493869, 0.5906602491211191, -0.5196601050406684, 1.028351583244202, 0.5049676701171434, 1.2407310695224367, -0.1307487243450276

Part 2: Polynomial Regression Using Basis Expansion

For this part I used the exact same linear regression class as the previous part. To create the higher order polynomials, I first normalized my one feature by making each value between zero and one. I then raised those values to the required power, creating a new feature column for each value between one and the required order. Each alpha value for the six different models were chosen by running tests until a good alpha value was found and remained static. Since I used the same class as the previous part, I will not go over it again. Below is the list for my six different polynomial regression models. All of these models had a step amount of 100,000 but always ended before then.

Synthetic-1 2nd Order Polynomial:

* Alpha = 0.01
* Mean Squared Error = 34.99445951131108
* Weights = -3.371935302806368, -0.405715073478246, 1.3245497611334005

Synthetic-1 3rd Order Polynomial:

* Alpha = 1
* Mean Squared Error = 9.999802553078649
* Weights = 2.994860349332081, -107.12166040051156, 274.66630047733247, -175.9715467401156

Synthetic-1 5th Order Polynomial:

* Alpha = 1
* Mean Squared Error = 9.99997249486212
* Weights = 3.646981430900197, -101.02762279795057, 202.04049386248644, -19.90321725559499, -89.02325169059579, -3.2207280654613526

Synthetic-2 2nd Order Polynomial:

* Alpha = 0.01
* Mean Squared Error = 0.4262049302197737
* Weights = 0.5575337587821269, -0.9070902776337175, 0.8290953597849535

Synthetic-2 3rd Order Polynomial:

* Alpha = 0.01
* Mean Squared Error = 0.4965338928388875
* Weights = 0.5515920450457329, 0.6403871226127464, -0.48856400396820077, -0.659602712916918

Synthetic-2 5th Order Polynomial:

* Alpha = 0.01
* Mean Squared Error = 0.49939470524386737
* Weights = -0.2543602605451638, 0.49958382115319366, -0.563918145610953, -0.32852105060236636, 0.32957028036739794, 0.9408338269778844

Part 3: Plot Your Regression Lines

In order to plot my polynomial regression models, I used the matplotlib library to create a scatter of the normalized data points and regression line. The x-axis of my plot is my one feature, and the y-axis is my class label. I created a function that will take in my theta values of my model and create an equation based on them. The x-values I used to model my line were created by using np.linespace(0,1,10000), which creates 10,000 equally spaced values between zero and one. The six plots of my models are below.

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: Polynomial Regression with Regularization

In order to do L2 Norm Regularization I used a lambda value of .1. I commented this out in my code, so it won’t affect any other models whenever it is turned in and ran. Looking at the graphs for the 5th order polynomials and their regularized counterparts, there seems to be a little bit of difference. There is especially a difference between the synthetic-1 models, with the regularized version having a much smaller curve overall. The synthetic-2 models also have differences, with the regularized version being a higher line with less of a curve. I think that the regularization made a difference in my models because I normalized my data before creating them. Since I did this, I think the small .1 lambda value had a greater affect than if I didn’t normalize my data. My MSE for the regularized synthetic-1 model is higher than what it is for the unregularized version and seems to have hurt my model.

Synthetic-1 5 Order Polynomial Regularized MSE: 17.870840119646537

Synthetic-2 5 Order Polynomial Regularized MSE: 0.48302225878379673

Chart, scatter chart

Description automatically generatedChart, scatter chart

Description automatically generated