Tristan Basil
CS460G-001 Machine Learning – Dr. Brent Harrison
3/5/2018

Problem Set 3 - Report

This assignment implements a polynomial regression algorithm with gradient descent that is used to predict y values for three synthetic datasets. There is one included program: Program3.py. It is implemented in Python 2. This report will provide a brief overview of the polynomial regression implementation and usage with visuals.

Part 1: Polynomial Regression with Gradient Descent

Program3.py implements polynomial regression using gradient descent. It is implemented in Python 2 and uses the following packages:

1. **sys** to get command line arguments (the filename),
2. **numpy** to create quick for loops with floats, and
3. **matplotlib** to model data in charts.

For this part, Program3.py is run with the following command:

```
$ python Program3.py [filename] [order]
```

Where [filename] should be replaced with the training file name and [order] should be replaced with the polynomial order of weights that the regression algorithm should use. This will initialize the polynomial regression algorithm, display the variable alpha decreasing to an effective value, then the cost of the algorithm at every 1000 iterations of gradient descent. After running until each weight changes by less than 0.01, or 100,000 iterations have passed, the final mean squared error will be displayed along with the weights the algorithm optimized. Note that using order 9 will always result in the 100,000 iteration limit being met (which takes around 4 minutes), but if more iterations were run, the decreasing cost shows that the algorithm would eventually stop. There were a few implementation decisions worth mentioning:

1. This implementation uses full batch gradient descent.
2. This implementation uses a variable alpha, which is multiplied by 0.5 each time the resulting error from an iteration would have been greater than that of the previous iteration. Because of this, the initial alpha value of 1 was chosen arbitrarily and is decreased to a suitable value for any given order.
3. The 100,000 iteration limit was chosen arbitrarily, after talking to Dr. Harrison in class.
4. If the change in all weights is less than 0.01, regression stops. This is an arbitrary stopping condition that allows for distinguishable differences in the plots that were printed in part 2.
5. The weights were initialized to 0 arbitrarily.

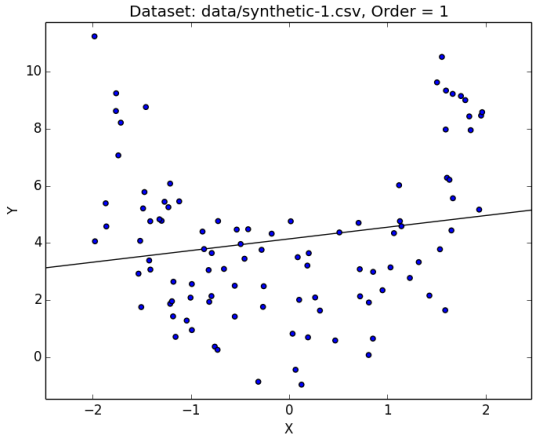
The mean squared error results and weights for each data set and order follow:

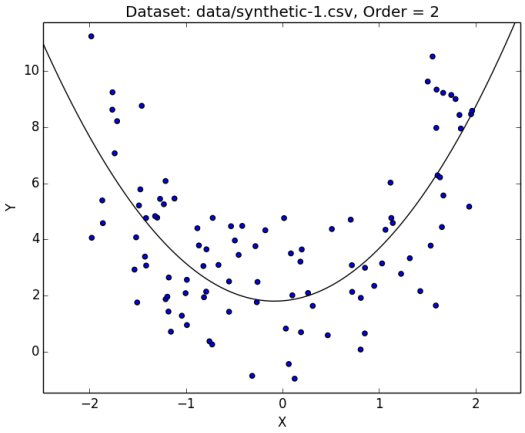
Training set	Order	Mean Squared Error	Weights (θ_0, θ_1, …, θ_{Order})
synthetic-1.csv	1	7.18125545446	[4.13928466161656, 0.4091414768014465]
synthetic-1.csv	2	3.77086293137	[1.8061790045929957, 0.26715352200545045, 1.5993340564223362]
synthetic-1.csv	4	3.74767857892	[2.0293994133917974, 0.20494683830761898, 1.1273756034894127, 0.021121263889241612, 0.13328778401769892]
synthetic-1.csv	9	3.71807083629	[1.8593309374557587, 0.469029295044813, 1.0009040266195355, 0.02823990158445133, 0.3483149945881487, −0.254162867611155, 0.0025154290154203013, 0.1387823738059825, −0.014979951454633354, −0.02102694450334377]
synthetic-2.csv	1	31.0619314115	[−3.5494606477832917, 1.9871322521588999]
synthetic-2.csv	2	30.4056244996	[−4.410741622195432, 1.8889612362192238, 0.674994929005409]
synthetic-2.csv	4	8.28731178219	[−3.0003879721918207, 11.34299457179249, −1.6224742929958418, −3.9451634891945577, 0.7309374483981106]
synthetic-2.csv	9	12.8524562324	[−4.1919341790314135, 3.1626238520441317, 0.48418632994009536, 3.6504176341522476, 0.9890263470101822, 1.597811277741065, −0.6978045698203058, −2.226252006643883, 0.13613283979548946, 0.37993946918413085]
synthetic-3.csv	1	0.626756229317	[0.19665630085069027, −0.005262093836301718]
synthetic-3.csv	2	0.625838558949	[0.1458040852579301, −0.004111528924042779, 0.03307637852127355]

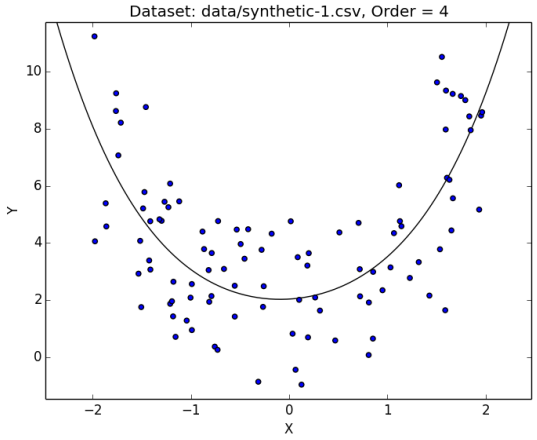
synthetic-3.csv	4	0.550763901359	[0.4118200468362114, 0.16519365217650647, -0.6377080754204318, -0.06188212627387233, 0.1914919169551756]
synthetic-3.csv	9	0.353643751764	[0.28532819186358716, 0.07814955691534739, 0.2712829944326238, 0.1072025390680838, -0.015972541046617345, -0.026648880978534153, -0.2947663830572262, -0.03149825327229785, 0.0791039847402398, 0.007755762483492536]

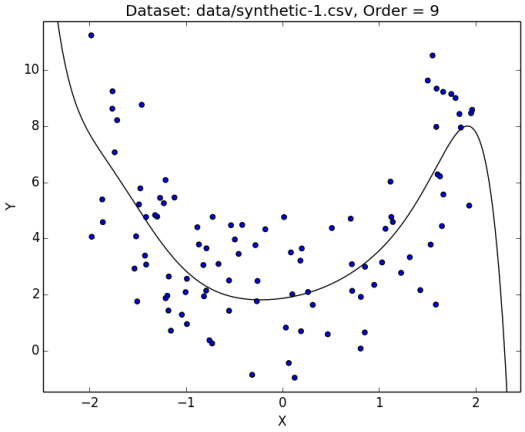
Part 2: Plot your regression lines

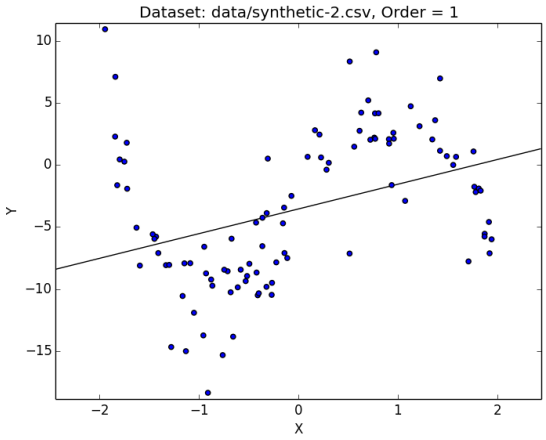
Here are the regression lines for each order and data set:

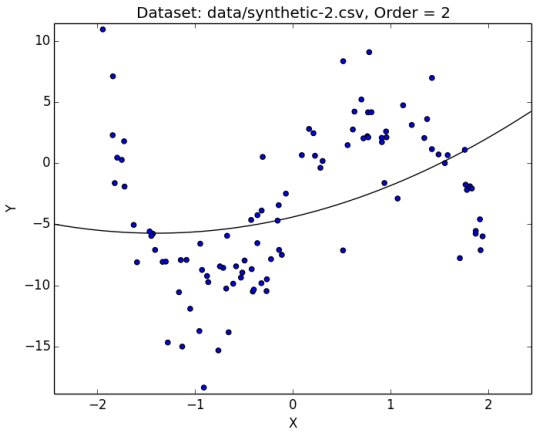


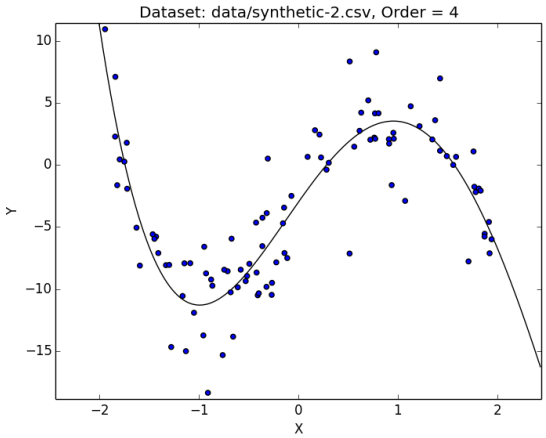


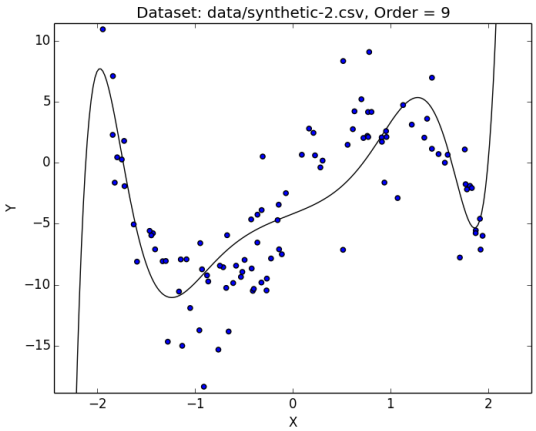


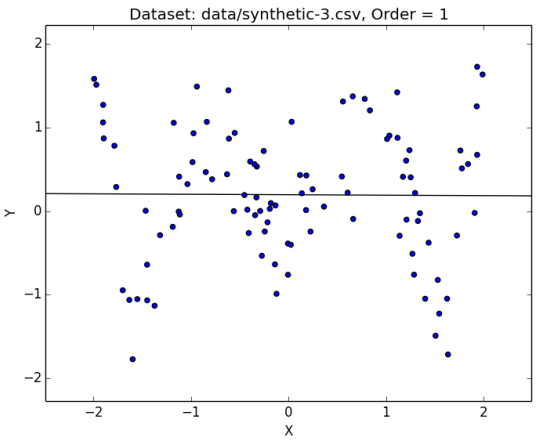


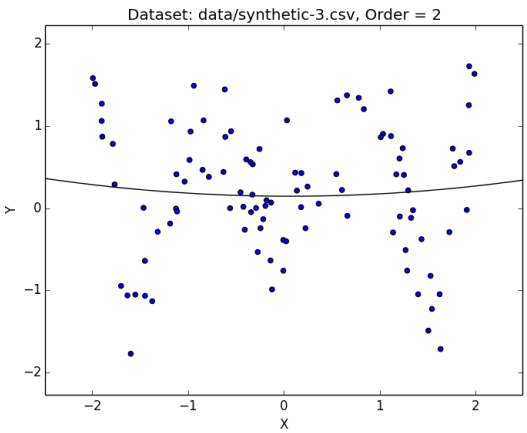

Dataset: data/synthetic-2.csv, Order = 1
X
Y

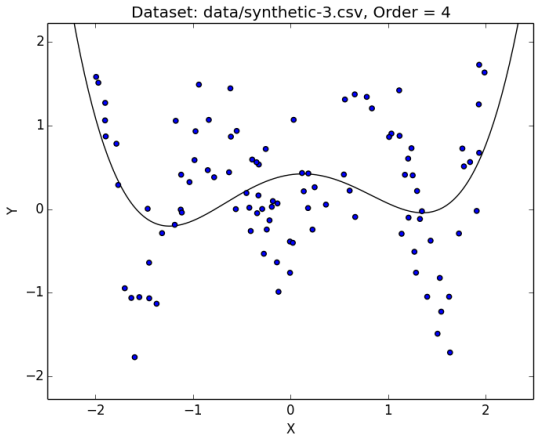

Dataset: data/synthetic-2.csv, Order = 2
X
Y

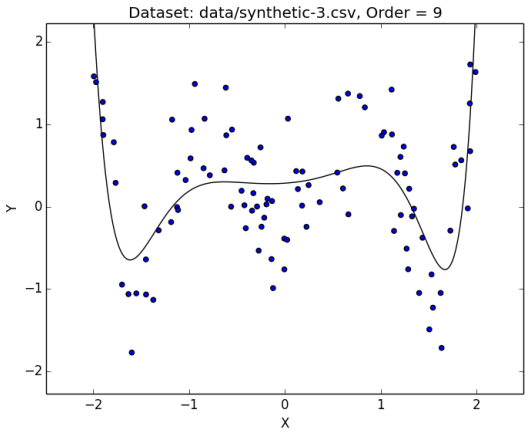

Dataset: data/synthetic-2.csv, Order = 4
X
Y


Dataset: data/synthetic-2.csv, Order = 9
X
Y


Dataset: data/synthetic-3.csv, Order = 1
X
Y


Dataset: data/synthetic-3.csv, Order = 2
X
Y





Bonus: Polynomial Regression with Regularization

Program3Regularization.py extends Program3.py and implements polynomial regression using gradient descent with L2 regularization. It is implemented in Python 2 and uses the same packages as Program3.py, outlined in part 1. For this part, Program3Regularization.py is run with the following command:

```
$ python Program3Regularization.py [filename] [order] [lambda]
```

Where [filename] should be replaced with the training file name, [order] should be replaced with the polynomial order of weights that the regression algorithm should use, and [lambda] should be replaced with the L2 Regularization constant. Because this is an extension of Program3.py, all implementation decisions in part 1 apply to this part as well. That being said, there was an additional implementation decision worth mentioning:

1. Lambda = 1.0 was used arbitrarily as the L2 regularization constant for this part.

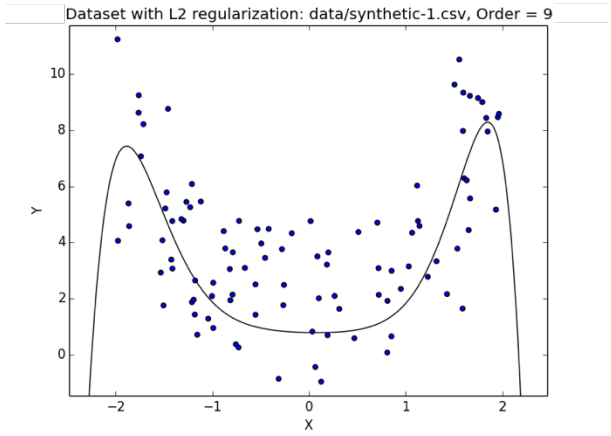
Did regularization make a difference? Why do you think that it did or didn't?

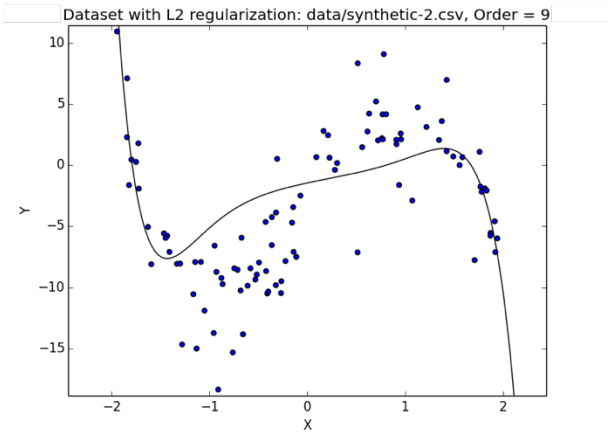
Regularization made a difference in the mean squared errors in that they were larger with L2 regularization, which was expected. Additionally, the models for synthetic-1.csv and synthetic-3.csv actually reached the <0.01 stopping condition for weight changes I outlined in part 1, preventing iteration counts much greater than the hard limit I set at 100,000 iterations and resulting in a much faster runtime. I think that the regularization penalized overly complex modeling by limiting the magnitude of the overall weight vector in the modified cost function, penalizing the extreme curves associated with large weight values and resulting in a more general model. Overall, the use of L2 normalization with lambda = 1.0 greatly generalized the models and resulted in reaching the stopping condition in gradient descent much faster.

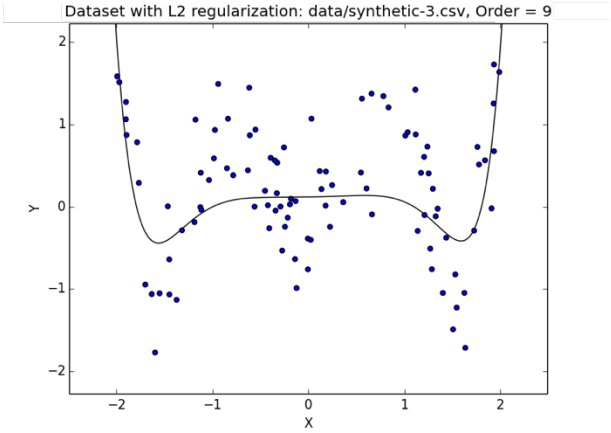
Here are the results of running the L2 Program3Regularlization.py with each dataset as an order 9 polynomial with Lambda = 1.0:

Training Set	Order	Mean Squared Error	Weights (θ_0, θ_1, …, θ_9)	Lambda
synthetic-1.csv	9	4.58551780587	[0.7779996231599843, −0.06500072262704903, 0.41443610347628196, −0.0010403668289569643, 0.4056229561718044, 0.013062958093027868, 0.3386107518790766, 0.04054403546709581, −0.0924452449741846, −0.01070085716081117]	1
synthetic-2.csv	9	21.9269666499	[−1.4635651122217426, 1.7107245747992184, −0.5666002451408378, 0.9001914901968202, −0.29972152557591225, 0.4523947265655499, −0.0510800298409283, −0.22014061297106308, 0.06648727273325665, −0.024209644515050182]	1
synthetic-3.csv	9	0.411858656927	[0.11801513413947413, 0.02075637597190514, 0.04162732500719578, 0.0107510484168716, −0.04800932659453256, 0.006734994647959479, −0.12384352908438873, −0.008627194534793096, 0.04056379695199585, 0.0013414256979435045]	1

Here are the regression lines for each data set using L2 Regularization with lambda = 1.0:







Resources Used:

- https://stackoverflow.com/questions/3777861/setting-y-axis-limit-in-matplotlib (used to set axes on each plot so line would run through the edges)