HW 1: Programming Portion

github.com/Buggy-Virus/SI-ML-Homework/tree/master/assignment-1
October 7, 2021

Problem 2

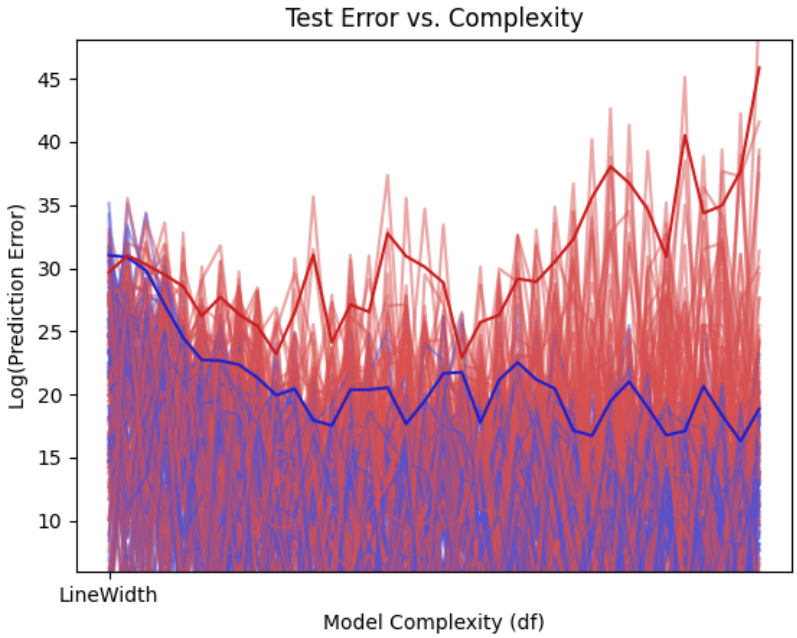
My program for problem 2 is generally structured into four different functions and a main function, I'll summarize them quickly here.

- generate_curve_function
  This returns a callable function which is some polynomial with noise. The number of variables and degrees are passed, as well on parameters to control the random variables that are generated for coefficients and parameters to use as noise parameters every time the created function is called.

- create_data
  Create data simply takes in the number of observations, predictions, and a callable function and creates a number of predictor vectors with the length of the number of observations, each vector is generated with a Gaussian function with different parameters for mean and noise for each predictor vector. These are stitched together to form x, and then the function is called on x to get y.

- get_curves
  Get curves does most of the heavy lifting. iterating through the number of instances we want to run (in this case 100). And each instance creating data for both training and testing (50 obs for each here), and then iterating from 0 to the max complexity (35 in this case), each time training a polynomial Lasso model using SKlearn with a degree equal to the current complexity.
  This model then has its mean squared error for the predictor having been fed the training data. This represents the training error. Which we then take the mean of all the MSQE at each complexity to create a curve representing the expected test error.
  Additionally at each of these complexities for each of these instances, an equally another x as large as the training set, which was reserved from training is used for prediction, and compared against the real y that was generated along with the data. This represents the conditional test error, and similarly it takes the mean across the complexities of these curves to get an estimate of the conditional test error.

- plot_curves
  Plot curves does what it says, it plots the curves. I did have very high values for my error, so I took the log before plotting, which is why the mean curves do not look like they sit exactly in the middle of the curves they were generated from.



Above you can see the result of the program. Which illustrates the point made in the original image that as complexity increases and fit of the training set increases, eventually it causes wild overfit and causes the conditional test error to begin exploding. This was generated with using a 9 degree polynomial as the underlying function.

Regrettably the way I setup my program made it difficult for me to extract a graph where the individual curves and the resulting mean curves weren't overwhelming noisy. I experimented with various values passed to my curve function generate and my create data generator, but wasn't able to find more pleasant values in time.

The below table shows the values for Test and Conditional Test Error mean and standard deviation at 0 degrees of freedom and 35. And you see the expected effect of it reducing for both in the test case, while exploding at 35 for the conditional test error. If more data was presented we would also see the effect the graph shows of the red curve dropping initially as complexity increases to match that of the funtion.

df	TE u	CTE u	TE std	CTE std
0	3.001889e+13	7.702568e+12	1.964344e+14	3.181605e+13
35	1.546179e+08	8.351572e+19	1.240098e+09	8.308365e+20

MATH-GA 2046 - HW 1: Programming Portion **Oliver Hare(oph206)**

Problem 3

dev	nulldev	df	lambdau	x1	x2
0.0	0.018750833389951583	0.0	3.757032166102741	0.0	0.0
0.3333628697205153	2.5972422825575787	2.0	2.9667490417319367	0.0	-3.060
0.5648701263711019	3.014092221492414	2.0	2.342700165313589	0.0	-6.043
0.7092266462113938	19.51686626105804	2.0	1.8499185429436844	0.0	-8.398
0.799240260133883	0.45444417944477034	2.0	1.4607924079216497	0.0	-10.25
0.8553683151767106	36.80409085395796	2.0	1.153518065528409	0.0	-11.72
0.8903670051038085	101.25822432643723	2.0	0.9108781783672647	0.0	-12.88
0.9123418764686804	0.29351492676403557	3.0	0.7192770365894479	0.013093777651933345	-13.80
0.9326606227201717	12.519578325793749	3.0	0.567978756821529	0.6630033847620588	-14.49
0.9453303922498775	13.842430232367589	3.0	0.4485057242063249	1.1762060310403917	-15.03
0.9532306363413504	42.87762915852784	3.0	0.35416357078482763	1.5814576742915425	-15.46
0.9581568393443739	60.1551745219009	3.0	0.2796660735891913	1.9014655611345228	-15.80
0.9612285768433698	48.45908110882338	3.0	0.22083895456405767	2.1541605207901093	-16.07
0.9631439610159154	12.820100759241194	3.0	0.17438598549706533	2.353701673557605	-16.28
0.9643383002228422	71.16933685862693	3.0	0.13770429224234382	2.5112698011611374	-16.45
0.9650830313121184	11.664299914199276	3.0	0.10873850927822375	2.6356938334579354	-16.58
0.9655474044048146	69.40823339681762	3.0	0.08586561252020636	2.7339420664146474	-16.68
0.9658369691123048	5.0315258479294656	3.0	0.0678039772883546	2.811527496818006	-16.77
0.9660175273983568	8.190496147958779	3.0	0.053541565723272816	2.8727930094776815	-16.83
0.9661301146501774	86.44381623332688	3.0	0.042279219814910486	2.921171461654433	-16.88
0.9662003185137036	56.3486270195805	3.0	0.03338588261307674	2.9593736170500673	-16.92
0.9662440941843439	29.025695901352933	3.0	0.026363238553920792	2.9895400364208276	-16.95
0.9662713905359086	4.1803414389139375	3.0	0.02081779161287492	3.0133610184630246	-16.98
0.9662884111959659	0.1269186644849361	3.0	0.01643881675427286	3.0321713114076574	-17.00
0.9662990244410398	8.431002072933468	3.0	0.012980949243119209	3.0470249022477036	-17.02
0.9663056423377971	6.589841036432505	3.0	0.010250436255312505	3.0587540742224806	-17.03

I'm up against the wire, and am rushing to submit (not that it's an excuse) so I have to apologize about not formatting the data. correctly where you could see theirs against mine side by side.

I was not able to converge to the same beta as glmnet generally, and my overall deviance was not as low. That being said looking into their code, I am unsure they were calculating deviance in the exact same way so it may have had an impact on the different between the two.

That being said, the directions of my steps, and the lambdas when the degrees of freedom increased were generally the same, showing that it was following a similar path, even if it were not the case that the beta it converged to was the same.

I'll try to resubmit nice data. (Just making sure I get something in.)