Machine Learning Assignment 1

Sayan Sinha
16CS10048

INTRODUCTION

Supervised machine Learning generally boils down to approximating a continuous function. One method of doing so is by making use of Linear Regression. In any method whatsoever, there always remains a set of hyperparameters which need to be tuned intuitively. In this assignment, we experiment with various values of such hyperparameters and try to draw a conclusion out of them.

EXPERIMENTATION

Part 1

A synthetic dataset is generated and is divided into train and test in a 4:1 ratio randomly. This is done by permuting the data in a random fashion and choosing the first 80% of the dataset to be the training set and the rest to be the test set. This part is kept as a separate function to make sure new data is generated only when required, and not all the time before training. The data is saved in the present working directory.

The dataset is then trained using Linear regression. The weights are initialised to random values between zero and one. At the end of each iteration, the loss up to that iteration is sent to STDOUT. Gradient descent is performed as per the following rule:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Here θ is the weight vector, with θ_j marking each element of the vector. α is the learning rate, m is the number of training examples and h_θ is the hypothesis function.
The training function returns the training error, testing error and the RMSE error. For the error, we use squared loss, i.e.:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(W^T \Phi_n(x) - y \right)^2$$

The training takes place separately for degrees 1 to 9. The degrees denote the polynomial degree of the function we are trying to approximate. Since bias is also introduced, there are a total of *degree + 1* parameters to be tuned while training. At the end of a certain number of iterations, these values are saved in the local directory.

The parameter values learned have been provided in Table 1. The first row denotes the degree.

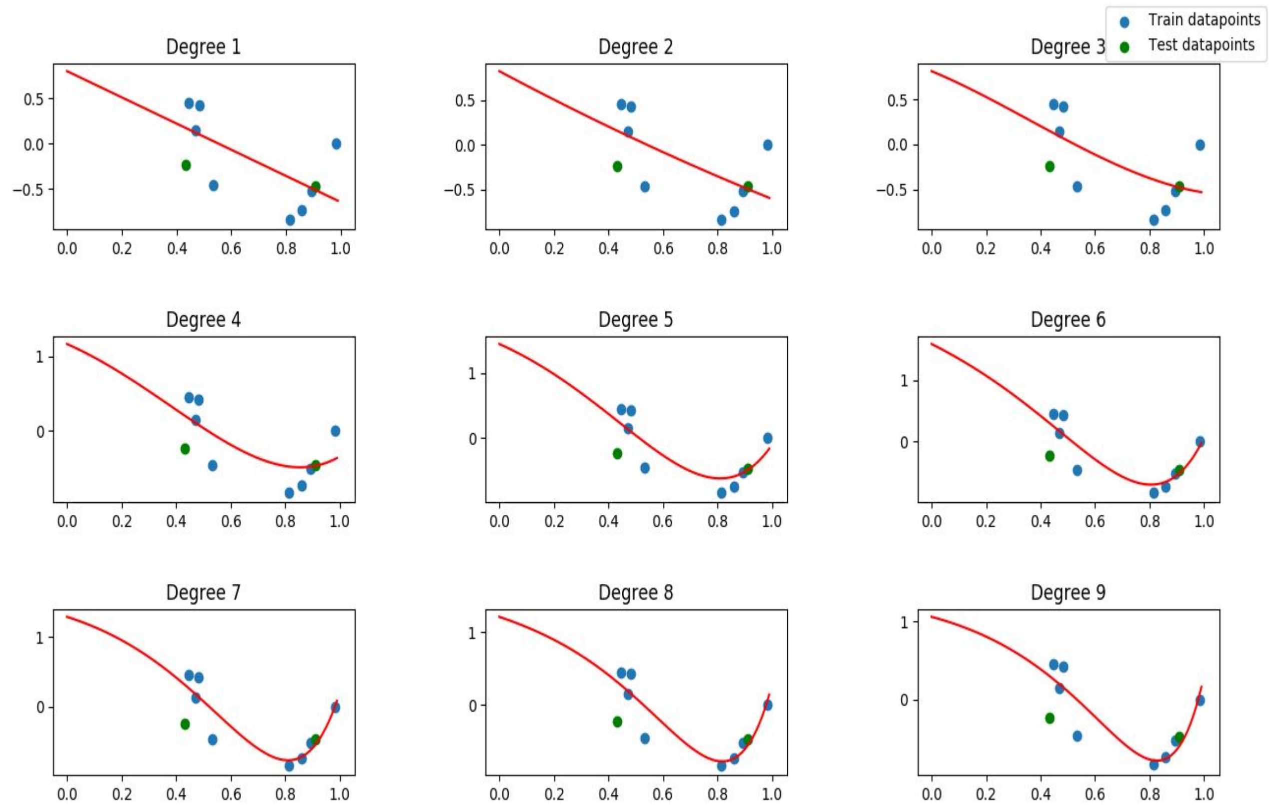
1	2	3	4	5	6	7	8	9
0.7975319619	0.8230280506	0.8183935063	1.1668628	1.442892006	1.594227469	1.293419923	1.216725803	1.056935601
-1.441639135	-1.628034357	-1.265472884	-1.637685999	-1.876832283	-2.252152497	-1.266530609	-1.278813704	-0.94086188
	0.1973332989	-1.059530691	-1.869621587	-2.140028412	-1.5653076	-1.752213111	-1.22210624	-1.410866293
		0.9698013993	0.4543025178	-0.1771347154	-0.6511959638	-1.541795004	-1.481039603	-0.9295300843
			1.535320926	0.4134760332	-0.03912077531	-0.03639705869	-0.4284835354	-0.4103477217
				2.233017764	1.128940674	0.8629344587	0.3571409422	-0.01671729166
					1.847131128	0.8161158519	0.7164855357	0.1033021222
						1.831891609	0.8452412498	0.5232267054
							1.55261011	0.8865247169
								1.451567311

Table 1: Weights for 10 datapoints

Part 2

Part 2 takes up the job of looking into how good the training went about using visual plots. First, it asks the user if there is a desire to generate a new dataset, or if at all the training needs to be performed again. After such confirmations, it proceeds to plot the data. First, it reads the saved data and weights after which it recomputes the test and train errors. This is done as the test and train errors, though computed, were not saved in Part 1. For plotting the approximated function using the given weights, the output of the function is obtained for inputs between 0 and 1, separated by 0.01. Thus, when a line plot is drawn on this, it gives the illusion of a curve.

Fig 1: Curve fitting against 10 datapoints



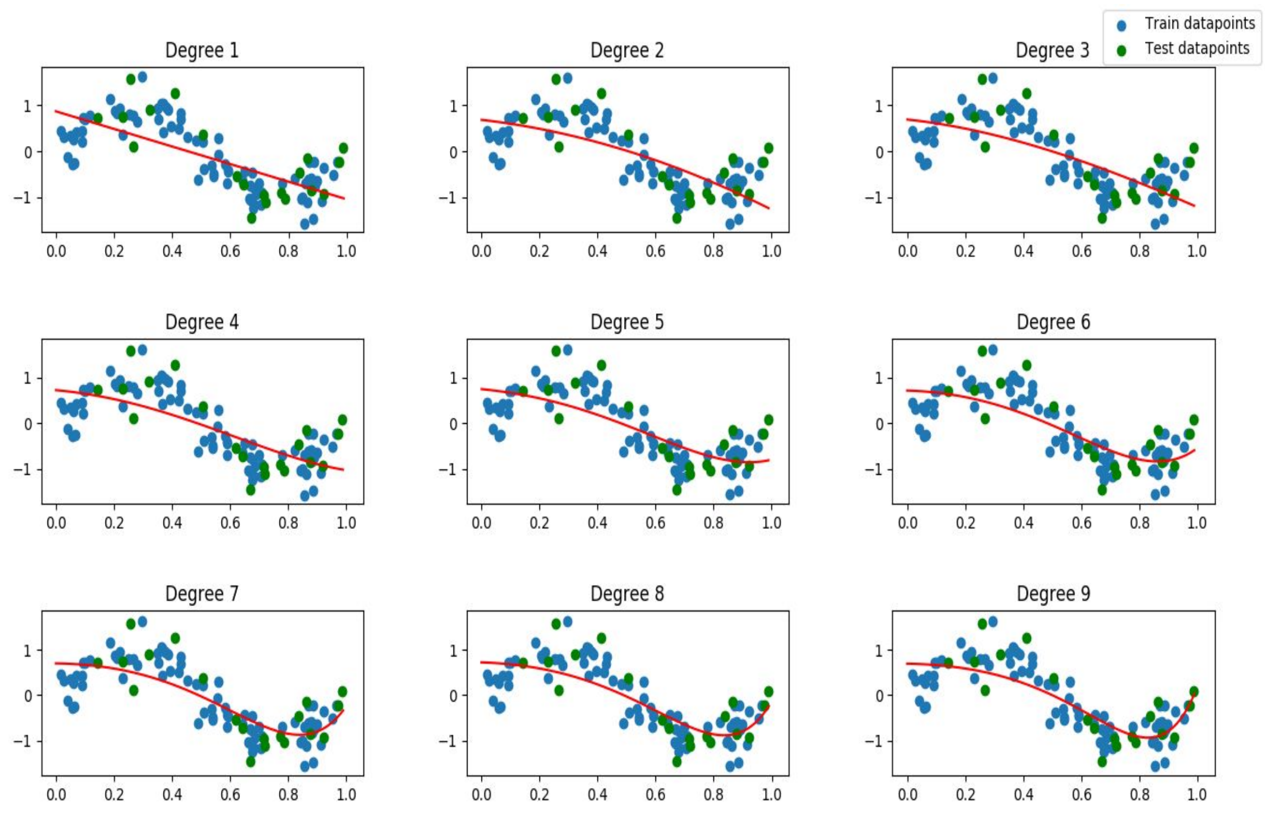


Fig 2: Curve fitting against 100 datapoints

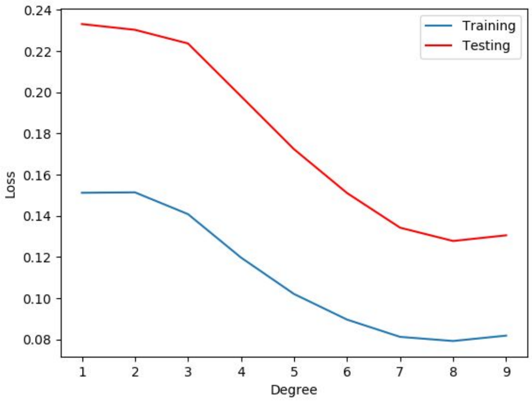


Fig 3: Loss v/s degree for 100 datapoints

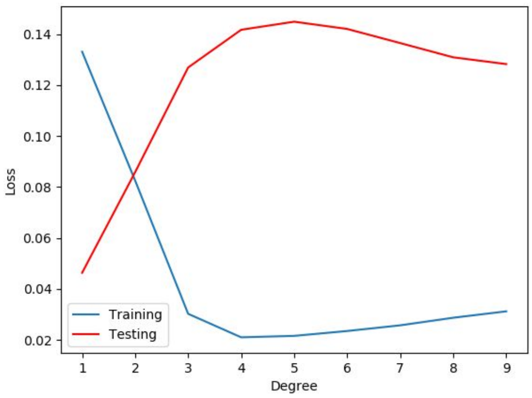


Fig 4: Loss v/s degree for 10 datapoints

Next, the error values obtained in the previous part is plotted against the degree of the polynomial function which is being approximated. Lastly, the degree for which the minimum value of test error was obtained, is returned.

In the experiment, it was found that if the number of datapoints is restricted to 10, the plots obtained were a bit arbitrary. When increased to 100, they looked like they had been anticipated.

Part 3

Part 3 makes a call to Part 2 to obtain the best value of n. Accordingly, it then instructs Part 1 to generate data with various sizes, namely 10, 100, 1,000, and 10,000. The weights obtained have been presented in Table 2, 3 and 4. Upon testing a number of times, the value of the best n is found according to the following distribution (Table 5). The plot against no of datapoints is given in Fig 5.

Table 5: Distribution of degrees

Degree	7	8	9
%	6.67	13.33	80.00

Table 2: Weights for 100 datapoints

1	2	3	4	5	6	7	8	9
0.8710262544	0.829211183	0.875575157	0.908950669	0.9066962384	0.884664807	0.8442195304	0.8218452823	0.7833830117
-1.894498614	-1.656683609	-1.625247068	-1.363524754	-1.085309647	-0.879548715	-0.6232059524	-0.5293453787	-0.3901481505
	-0.2260270969	-1.296194565	-1.916144381	-2.230179573	-2.199814806	-2.001010521	-1.938706417	-1.637137163
		1.160183327	-0.4602454562	-0.8824852599	-0.8762674572	-1.675845952	-1.206534577	-1.623533107
			2.240392562	1.066217635	-0.03871792457	0.05764072756	-0.9174305237	-0.8751853222
				1.875091938	0.908160266	0.6051387781	0.4390457082	0.381673158
					2.073271886	1.295730354	1.074630988	0.3045033555
						1.563315841	1.146619257	0.7089731967
							1.346124277	1.513295952
								1.27308484

Table 3: Weights for 1000 datapoints

1	2	3	4	5	6	7	8	9
0.7952949234	0.7956614451	0.6710086979	0.6747722721	0.6762151944	0.7410858763	0.6804140972	0.7977588119	0.7010402116
-1.606634293	-1.160043219	-0.6352979473	-0.7712088956	-0.5971610846	-1.063300577	-0.659153834	-1.048135427	-0.6758975782
	-0.6432510116	-0.5930785415	-0.3775994579	-1.008405992	-0.5432735428	-1.003218382	-0.6164176367	-0.7705241842
		-0.6035425272	-0.7093812278	-0.5299901328	-0.1561004362	-0.3253323324	-1.015222288	-0.7028251569
			0.08222034768	0.2349528062	-0.1114817494	-0.3636626983	-0.03881281459	-0.7782746306
				0.2671951549	-0.1732857535	0.4754291399	0.4174995428	0.2780407732
					0.4212442658	0.04348657395	0.4134331526	0.08779355429
						0.3870090428	-0.1779750626	0.3389685656

							0.7677204145	0.9017862445
								0.197253533

Table 4: Weights for 10,000 datapoints

1	2	3	4	5	6	7	8	9
0.7817143592	0.7099143363	0.6829120663	0.6907175004	0.6828111931	0.752605908	0.731046685	0.7099359958	0.8534327123
-1.572001909	-0.7281331692	-0.5973554095	-0.7419433226	-0.9360888669	-0.9072858963	-0.6476066363	-0.8274934965	-1.121219198
	-1.033013008	-1.085452251	-0.7087318272	-0.3199051655	-1.038105144	-1.134167163	-0.6964087892	-0.9780148136
		-0.07708984932	-0.5570433458	-0.2741011187	-0.07861873709	-0.8304785904	-0.6281422084	-0.1718268267
			0.2927682211	-0.3073883862	-0.2964376787	0.03263071575	-0.1969830868	-0.7225220256
				0.1351452717	0.4535990327	-0.06545867159	0.0781221145	0.03853214125
					0.3550223538	0.5607274715	0.1711708567	0.5505616662
						0.8175464423	0.1183689984	-0.161832919
							0.6918120453	0.6537416599
								0.8281820676

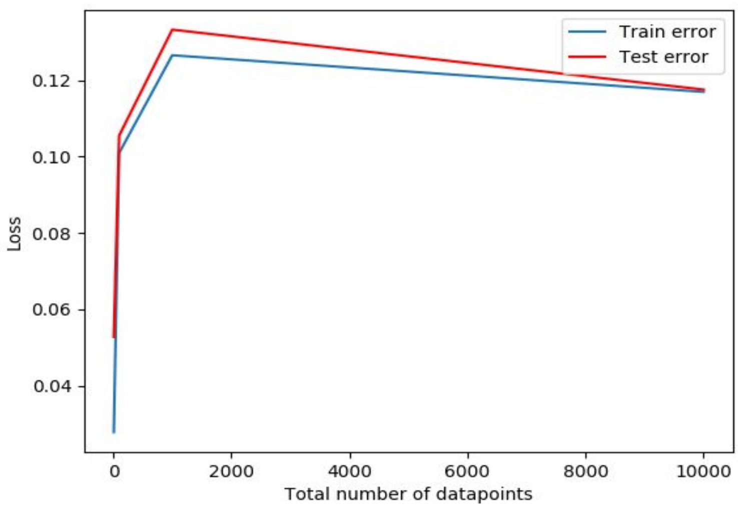


Fig 5: Loss v/s no of datapoints

Part 4

Two other cost functions were experimented with, namely the mean absolute error and the fourth power error. For the mean absolute error, the following formula was used for computing cost

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left| W^T \Phi_n(x) - y \right|$$

and the following for gradient.

$$\frac{d\mathrm{MAE}}{dy_{\mathrm{pred}}} = \begin{cases} +1, & y_{\mathrm{pred}} > y_{\mathrm{true}} \\ -1, & y_{\mathrm{pred}} < y_{\mathrm{true}} \end{cases}$$

For the fourth power error, the cost function was

$$\frac{1}{2m} \sum_{i=1}^{m} \left(W^T \Phi_n(x) - y \right)^4$$

Accordingly, the gradient comes out to be

$$\frac{2}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^3 x_j^{(i)}$$

Here, the weights have been initialised randomly between 0 and 0.001 to prevent exploding of values.

The RMSE error is trained and computed for each degree. For each instance of the learning rate (0.025, 0.05, 0.1, 0.2 and 0.5) and for every cost function, the degree having the minimum value of RMSE is chosen and plotted across the learning rate values. The weights have been given in Table 6, 7 and 8. The provided weights are for that degree for which the minimum value of RMSE was obtained. The first row of the table mentions the learning rate.

Table 6: Weights for Squared error loss

0.025	**0.05**	**0.1**	**0.2**	**0.5**
0.7293215757	0.7465717494	0.9291357642	1.042771469	0.9614016857
-0.7186918233	-0.8293871128	-0.9888155837	-1.183347055	-0.5719301969
-0.845257154	-0.9279066871	-1.329807527	-1.330827547	-1.885688898
-0.8664141269	-0.2307502415	-0.8151101417	-0.9837553255	-1.699878445
0.07024936875	-0.1309270971	-0.3318826047	-0.6089554393	-0.173358147
0.1790967975	-0.1041283308	0.246028049	-0.1855961717	-0.2156919053
-0.1095506467	0.1267188376	0.4721587248	0.9466496538	0.669988285
0.3165117195	-0.1687833015	0.7737945748	0.4701290394	0.8501394341
0.2331473181	0.7872001551	0.8259655405	1.190339921	1.096247705
0.5834135234	0.242288971		0.6975961542	1.155297139

Table 7: Weights for Mean Absolute Error

0.025	0.05	0.1	0.2	0.5
0.5897437497	0.7458329931	0.990000795	1.225462428	1.269331133
-0.5366108515	-0.7737905044	-1.352076637	-1.661870657	-1.43829365
-0.5703084878	-0.5708972985	-0.8887757569	-1.341224788	-1.558887571
-0.5431326869	-1.033364742	-0.3741909892	-0.4470348273	-1.569393699
-0.3934620506	0.06160323211	-0.5582285952	-0.1297491422	-0.2512687934
0.263836122	0.266539504	0.1825713155	-0.3301895356	0.3245040276
0.2240216534	0.06892601893	0.4764327226	0.2615325467	0.9356641203
0.02726837111	0.2371244755	0.3778189537	0.3538152723	1.070499122
-0.1784790498	0.5112638983	0.3292100977	1.07589597	1.193997085
0.5924522313		0.5246380648	0.9210709979	

Table 8: Weights for Fourth Power Error

0.025	0.05	0.1	0.2	0.5
0.6245150067	0.8007088396	0.9432650773	0.989307045	0.9290307589
-0.670324757	-0.9542326425	-1.168411553	-1.129097379	-0.6344867589
-0.6319956396	-0.883328324	-1.152236085	-1.400483947	-1.812908766
-0.3972964747	-0.5198589783	-0.6711720606	-0.870859511	-1.294542826
-0.1873818257	-0.1955922497	-0.2225846011	-0.3046772061	-0.5224040152
-0.03427099297	0.04018260693	0.1083178591	0.1296636942	0.1047213058
0.0724038637	0.201124684	0.3334643396	0.4263289758	0.5362645949
0.1431212527	0.3063118265	0.4794926944	0.618876233	0.8057831371
0.1888810998	0.3731522693	0.5704175346	0.7354502344	0.9603547974
0.2180880341	0.4133952611	0.6234962827	0.8010226496	1.034530432

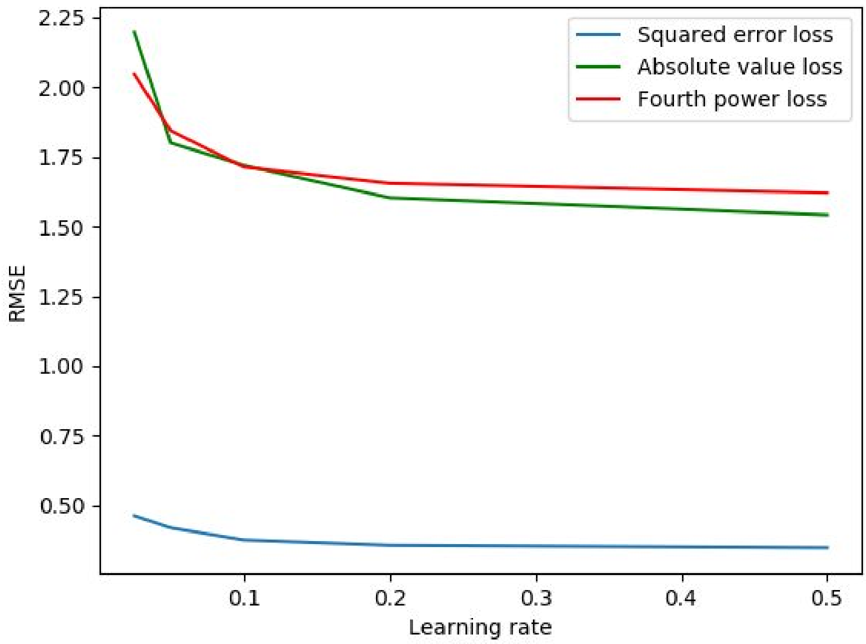


Fig 6: Test RMSE v/s Learning rate

CONCLUSION

We conclude that the best degree for fitting the curve has been 9 (as evident from Table 5), as seen in Part 4. The best cost function is Squared Error Loss (as evident from Fig 6). In reality, higher powers are prone to noises, and since noises were manually added to the synthetic dataset generated, perhaps that's why it did not perform well. The mean absolute error is not differentiable and hence, not used in general. The best learning rate has been experimentally found to be 0.5. Along with that, we can infer that we should have sufficient number of datapoints to draw valid conclusions about our model. For instance, in Part 2, having 10 datapoints gave us a wrong notion about our model (ref to Fig 4), whereas having a 100 datapoints could fit the data better (Fig 2). Having an even higher no of datapoints decreases the loss, as evident in Part 3 (Fig 5).

REFERENCES

- Seber, George AF, and Alan J. Lee. Linear regression analysis. Vol. 329. John Wiley & Sons, 2012.
- Needell, Deanna, Rachel Ward, and Nati Srebro. "Stochastic gradient descent, weighted sampling, and the randomized Kaczmarz algorithm." Advances in Neural Information Processing Systems. 2014.
- Robert, Christian. "Machine learning, a probabilistic perspective." (2014): 62-63.