BITS F464: Machine Learning

20 Linear Model for Regression

Dr. Kamlesh Tiwari
Assistant Professor, Department of CSIS,
BITS Pilani, Pilani Campus, Rajasthan-333031 INDIA

March 10, 2021 ONLINE (Campus @ BITS-Pilani Jan-May 2021)

http://ktiwari.in/ml

Regression

Consider following data

x_1	x_2	x_3	y
10	50	20	1
11	31	22	0
11	12	15	0
20	55	20	1
23	41	27	1
31	12	35	0
13	18	12	0
21	55	16	0
32	56	27	1
12	22	11	?

x_1	x_2	x_3	y
10	50	20	10
11	31	22	12
11	12	15	4
20	55	20	22
23	41	27	1
31	12	35	9
13	18	12	23
21	55	16	16
32	56	27	22
12	22	11	?

What should be ?

Regression

Regression: predicts the value of continuous target variable

- A simplest model for regression can be a linear combination of the input variables

$$y(x,w) = w_0 + w_1x_1 + ... + w_nx_n$$

where input x is represented by a n dimensional feature vector $(x_1, x_2, ..., x_n)$

- It can be extended by considering linear combination of fixed nonlinear functions ϕ (called basis functions)

$$y(x,w) = w_0 + \sum_{i=1}^{n} w_i\phi_i(x)$$

- In short $y(x,w) = w^T\phi(x)$
- Objective is to choose w, such that it makes $y(x^{(i)}, w)$ as close to $y^{(i)}$ as possible

Example

Assume for some chosen w we compute $y(x,w)$

x_1	x_2	x_3	y	$\hat{y} = y(x,w)$	$(\hat{y} - y)^2$
10	50	20	10	8	4
11	31	22	12	9	9
11	12	15	4	3	1
20	55	20	22	26	16
23	41	27	1	1	0
31	12	35	9	4	25
13	18	12	23	30	49
21	55	16	16	13	9
32	56	27	22	21	1

Then

$$J(w) = \frac{1}{2 \times 9} \times 114 = 6.33$$

Regression

- Determining w, is similar to solving a minimization problem. Let us define a **squared error cost function** as

$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(y(x^{(i)}, w) - y^{(i)})^2$$

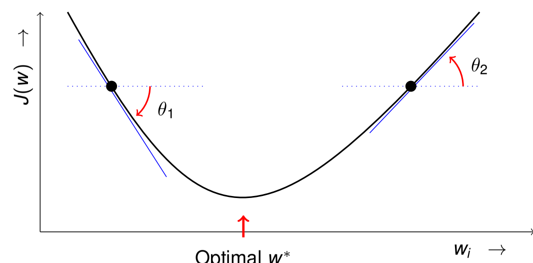
where m is number of training examples

- Then one have to minimize the value of $J(w)$

$$\operatorname*{argmin}_{w} J(w)$$

- Basic idea: Push w_i a bit against the direction of its gradient

Consider $w_i = w_i - \alpha\frac{\partial}{\partial w_i}J(w)$



- Slope $\tan\theta_1$, representing $\frac{\partial}{\partial w_i}J(w)$ is $-ve$ so the equation $w_i = w_i - \alpha\frac{\partial}{\partial w_i}J(w)$ moves w_i towards w^*
- $\tan\theta_2$, being $+ve$ the equation still moves w_i towards w^*

Gradient Descent

Algorithm 1: Gradient Descent

1. Initialize w randomly
2. **repeat**
3. Simultaneously update all w_j with $w_j - \alpha \frac{\partial}{\partial w_j} J(w)$
4. **until** *converge*;
5. **return** w

- Here α is a learning rate. If α is small enough then $J(w)$ would decrease in every iteration
  (large α can overshoot the minimum and may fail to converge)
- Susceptible to local minimum
- As it moves closer to local minimum, it automatically takes smaller steps as gradient decreases. Time complexity $O(m)$

The Partial Derivative term

$$\begin{aligned}
\frac{\partial}{\partial w_j} J(w) &= \frac{\partial}{\partial w_j} \frac{1}{2m} \sum_{i=1}^{m} (y(x^{(i)}, w) - y^{(i)})^2 \\
&= \frac{1}{2m} \sum_{i=1}^{m} \frac{\partial}{\partial w_j} (y(x^{(i)}, w) - y^{(i)})^2 \\
&= \frac{1}{2m} \sum_{i=1}^{m} 2(y(x^{(i)}, w) - y^{(i)}) \frac{\partial}{\partial w_j} (y(x^{(i)}, w) - y^{(i)}) \\
&= \frac{1}{m} \sum_{i=1}^{m} (y(x^{(i)}, w) - y^{(i)}) \frac{\partial}{\partial w_j} y(x^{(i)}, w)
\end{aligned}$$

For $y(x^{(i)}, w) = w_0 + w_1 x_1^{(i)} + \ldots + w_n x_n^{(i)}$ we have $\boxed{\frac{\partial}{\partial w_j} y(x^{(i)}, w) = x_j^{(i)}}$

$$\frac{\partial}{\partial w_j} J(w) = \frac{1}{m} \sum_{i=1}^{m} (y(x^{(i)}, w) - y^{(i)}) x_j^{(i)}$$

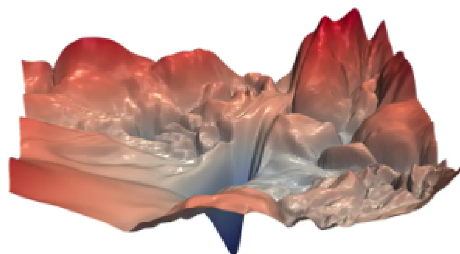
Batch-Gradient Descent

Algorithm 2: Batch-Gradient Descent

1. Initialize w randomly
2. **repeat**
3. Simultaneously update all w_j with
   $w_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (y(x^{(i)}, w) - y^{(i)}) x_j^{(i)}$
4. **until** *converge*;
5. **return** w

- At every step it evaluate all training examples
- Sometime it is also called multi-variate linear regression

Real Loss Landscape



- Loss Landscape is rude [1]

[1] Visualizing the Loss Landscape of Neural Nets, Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, Tom Goldstein, 2018

Example: Gradient Descent (learning rate α)

Consider following data

	x_1	x_2	x_3	y
1	10	50	20	10
2	11	31	22	12
3	11	12	15	4
4	20	55	20	22
5	23	41	27	1
6	31	12	35	9
7	13	18	12	23
8	21	55	16	16
9	32	56	27	22
10	8	22	35	11

Learning rate $\alpha = 0.1$

J=396.662506
w=(0.500 0.500 0.500 0.500)

J=19454472.000000
w=(-2.055 -51.070 -100.970 -62.640)

J=1036526813184.000000
w=(590.236 11518.771 23902.906 13778.349)

J=55230041021218816.000000
w=(-135891.922 -2653678.250 -5525792.000 -3170425.000)

J=2942865354556228763648.000000
w=(31365378.000 612476928.000 1275658624.000 731686912.000)

J=156806972273681738831495168.000000
w=(-7240111104.000 -141378551808.000 -294465732608.000 -168895037440.000)

J=8355266546526971027269827428352.000000
w=(1671254376448.000 32634791002112.000 67972370530304.000 38986479304704.000)

J=445200079222591879770706068887306240.000000

w=(-385780270759936.000 -7533178826784768.000

-15690251045437440.000 -8999357718200320.000)

Example: Gradient Descent (learning rate α)

Consider following data

	x_1	x_2	x_3	y
1	10	50	20	10
2	11	31	22	12
3	11	12	15	4
4	20	55	20	22
5	23	41	27	1
6	31	12	35	9
7	13	18	12	23
8	21	55	16	16
9	32	56	27	22
10	8	22	35	11

Learning rate $\alpha = 0.001$

J	w
396.663	(0.500 0.500 0.500 0.500)
664.137	(0.474 -0.016 -0.515 -0.131)
1131.021	(0.508 0.631 0.881 0.628)
1943.882	(0.464 -0.249 -0.910 -0.435)
3357.625	(0.523 0.888 1.492 0.914)
5815.401	(0.446 -0.630 -1.641 -0.908)
10087.491	(0.549 1.356 2.518 1.456)
17512.684	(0.415 -1.274 -2.941 -1.693)
30417.834	(0.592 2.183 4.276 2.432)
52847.020	(0.359 -2.383 -5.221 -3.028)
91828.805	(0.668 3.630 7.314 4.151)
159578.781	(0.263 -4.302 -9.200 -5.330)
277327.562	(0.799 6.152 12.580 7.155)
481973.594	(0.093 -7.633 -16.125 -9.316)
837646.250	(1.025 10.537 21.725 12.387)
1455801.375	(-0.201 -13.418 -28.168 -16.234)
2530147.500	(1.417 18.162 37.611 21.491)
4397349.000	(-0.715 -23.472 -49.103 -28.249)
7642525.500	(2.097 31.415 65.218 37.319)
13282603.000	(-1.608 -40.944 -85.492 -49.126)
23084998.000	(3.278 54.449 113.196 64.832)
40121436.000	(-3.162 -71.310 -148.738 -85.405)
69730584.000	(5.329 94.483 196.578 112.653)
121190936.000	(-5.863 -124.085 -258.660 -148.456)
210628448.000	(8.894 164.060 341.494 195.769)
366069856.000	(-10.559 -215.809 -449.705 -258.035)
636225152.000	(15.088 284.983 593.355 340.226)
1105751936.000	(-18.721 -375.224 -781.739 -448.479)
1921783808.000	(25.852 495.147 1031.086 591.291)
3340036608.000	(-32.908 -652.287 -1358.811 -779.468)

Example: Gradient Descent (learning rate α)

Consider following data

	x_1	x_2	x_3	y
1	10	50	20	10
2	11	31	22	12
3	11	12	15	4
4	20	55	20	22
5	23	41	27	1
6	31	12	35	9
7	13	18	12	23
8	21	55	16	16
9	32	56	27	22
10	8	22	35	11

Learning rate $\alpha = 0.0001$

J	w	
396.663	(0.500 0.500 0.500 0.500)	
246.798	(0.497 0.448 0.399 0.437)	
158.286	(0.495 0.408 0.321 0.388)	
105.980	(0.494 0.377 0.262 0.349)	
75.041	(0.493 0.353 0.218 0.319)	
56.711	(0.492 0.334 0.184 0.295)	
45.826	(0.491 0.320 0.159 0.276)	
39.335	(0.491 0.308 0.140 0.260)	
35.439	(0.490 0.299 0.126 0.248)	
33.077	(0.490 0.291 0.115 0.238)	
31.621	(0.490 0.285 0.108 0.229)	
30.703	(0.490 0.280 0.103 0.222)	
30.104	(0.490 0.276 0.099 0.216)	
29.694	(0.489 0.273 0.097 0.210)	
29.399	(0.489 0.270 0.096 0.206)	
29.172	(0.489 0.268 0.095 0.202)	
28.987	(0.489 0.266 0.096 0.198)	
28.830	(0.489 0.264 0.096 0.194)	
28.689	(0.489 0.262 0.097 0.191)	
28.560	(0.489 0.260 0.098 0.188)	
28.439	(0.489 0.259 0.099 0.185)	
28.325	(0.489 0.258 0.101 0.182)	
28.216	(0.489 0.256 0.102 0.179)	
28.111	(0.489 0.255 0.104 0.177)	
28.011	(0.489 0.254 0.105 0.174)	
27.913	(0.489 0.253 0.107 0.172)	
27.819	(0.489 0.252 0.109 0.170)	
27.728	(0.489 0.251 0.110 0.167)	
27.555	(0.490 0.249 0.114 0.163)	
24.926	(0.507 0.207 0.215 0.020)	**Iteration 300**
24.768	(0.710 0.219 0.213 0.005)	**Iteration 3000**

Some Tricks

- Apply feature scaling (like min-max normalization)
- Sometime mean normalization is also good idea
- Declare convergence when $J(w)$ decreases less then a small constant ϵ (say $\epsilon = 0.001$)
- Choose α as 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, ...
- Having with some insight of the problem, you may define new features as $f(x_i, x_j, ..., x_k)$
- In case one wish to use $y(x, w) = w_0 + w_1x_1 + w_2x_2^2 + w_3x_3^4 + ...$ Replace the features x_2 with $x_2' = x_2^2$ and x_3 with $x_3' = x_3^4$ and so on. This would enable the use of gradient descent possible once again

Example: Gradient Descent (Feature scaling)

Feature scaling

	x_1	x_2	x_3	y
1	0.08	0.86	0.35	10
2	0.12	0.43	0.43	12
3	0.12	0.00	0.13	4
4	0.50	0.98	0.35	22
5	0.62	0.66	0.65	1
6	0.96	0.00	1.00	9
7	0.21	0.14	0.00	23
8	0.54	0.98	0.17	16
9	1.00	1.00	0.65	22
10	0.00	0.23	1.00	11

Learning rate $\alpha = 0.1$

J	w	
95.472	(0.500 0.500 0.500 0.500)	
73.399	(1.679 1.025 1.220 0.983)	
58.326	(2.658 1.455 1.822 1.364)	
48.020	(3.470 1.808 2.326 1.663)	
40.961	(4.147 2.096 2.749 1.893)	
36.116	(4.710 2.331 3.106 2.066)	
32.778	(5.180 2.522 3.407 2.193)	
30.468	(5.574 2.677 3.662 2.283)	
28.859	(5.903 2.803 3.880 2.341)	
27.729	(6.181 2.904 4.066 2.373)	
26.925	(6.415 2.985 4.226 2.385)	
26.344	(6.613 3.049 4.364 2.379)	
25.916	(6.782 3.100 4.485 2.360)	
25.593	(6.926 3.140 4.590 2.329)	
25.342	(7.050 3.170 4.683 2.289)	
25.141	(7.158 3.193 4.766 2.241)	
24.974	(7.252 3.210 4.839 2.188)	
24.833	(7.334 3.222 4.906 2.129)	
24.708	(7.407 3.230 4.966 2.067)	**Iteration 18**
24.596	(7.472 3.234 5.021 2.003)	
24.493	(7.530 3.236 5.071 1.935)	
24.397	(7.583 3.235 5.118 1.866)	
24.306	(7.632 3.233 5.161 1.796)	
24.219	(7.677 3.229 5.202 1.725)	
24.136	(7.718 3.225 5.241 1.653)	
24.056	(7.757 3.219 5.277 1.581)	
23.979	(7.794 3.213 5.311 1.509)	
23.903	(7.830 3.206 5.344 1.436)	
23.830	(7.863 3.198 5.375 1.364)	
23.759	(7.896 3.191 5.405 1.292)	**Iteration 30**
20.174	(12.021 4.618 4.794 -7.329)	**Iteration 3000**

Stochastic Gradient Descent

Gradient descent is computationally expensive for large training set

- Don't wait! **consider single training example during iteration.** [2]

Algorithm 3: Stochastic-Gradient Descent

1. Initialize w randomly
2. Randomly shuffle the dataset
3. **repeat**
4. **for** $i = 1$ *to* m **do**
5. Simultaneously update all w_j with $w_j - \alpha(y(x^{(i)}, w) - y^{(i)})x_j^{(i)}$
6. **until** *converge*;
7. **return** w

- The idea can be applied to other learning method as well

[2] Takes a random looking path and wonder around minimum (may not converge)

Example: Stochastic Gradient Descent

Data

	x_1	x_2	x_3	y
1	10	50	20	10
2	11	31	22	12
3	11	12	15	4
4	20	55	20	22
5	23	41	27	1
6	31	12	35	9
7	13	18	12	23
8	21	55	16	16
9	32	56	27	22
10	8	22	35	11

Learning rate $\alpha = 0.0008$

J	w
396.663	(0.500 0.500 0.500 0.500)
38.074	(0.484 0.320 -0.008 0.139)
108.403	(0.479 0.167 -0.068 -0.033)
42.720	(0.483 0.299 -0.016 0.116)
95.886	(0.479 0.185 -0.061 -0.013)
47.188	(0.482 0.283 -0.022 0.098)
87.268	(0.480 0.198 -0.055 0.002)
51.086	(0.482 0.272 -0.027 0.085)
81.238	(0.480 0.208 -0.052 0.013)
54.312	(0.482 0.263 -0.030 0.076)
76.962	(0.480 0.215 -0.049 0.022)
56.898	(0.482 0.257 -0.033 0.068)
73.893	(0.480 0.221 -0.047 0.028)
58.930	(0.481 0.252 -0.035 0.063)
71.671	(0.481 0.225 -0.045 0.033)
60.503	(0.481 0.248 -0.036 0.059)
70.050	(0.481 0.228 -0.044 0.036)
61.711	(0.481 0.246 -0.037 0.056)
68.860	(0.481 0.231 -0.043 0.039)
62.631	(0.481 0.244 -0.038 0.053)
67.983	(0.481 0.232 -0.042 0.041)
63.328	(0.481 0.242 -0.038 0.052)
67.334	(0.481 0.234 -0.042 0.042)
63.855	(0.481 0.241 -0.039 0.050)
66.853	(0.481 0.235 -0.041 0.043)
64.252	(0.481 0.240 -0.039 0.050)
66.495	(0.481 0.235 -0.041 0.044)
64.551	(0.481 0.240 -0.039 0.049)
66.229	(0.481 0.236 -0.041 0.045)
64.775	(0.481 0.239 -0.040 0.048)

Mini-Batch Gradient Descent

- Doing somewhat in between the batch-Gradient Descent (all m training sample) and Stochastic-Gradient Descent (only one training sample)
- Let b be the size of mini batch

Algorithm 4: Mini-Batch Gradient Descent

1. Initialize w randomly
2. Randomly shuffle the dataset
3. **repeat**
4. **for** $i = 1$ *to* $m - b + 1$ *in steps of* b **do**
5. Simultaneously update all w_j with $w_j - \alpha\frac{1}{b}\sum_{i=1}^{b}(y(x^{(i)}, w) - y^{(i)})x_j^{(i)}$
6. **until** *converge*;
7. **return** w

Direct method of MSE optimization

If we have two data points (x_1, y_1) and (x_2, y_2) having two attributes $m = 2, n = 1 + 1$

- Finding parameters w_0 and w_1 such that $y = w_0 + w_1 x$ is easy
- Let

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \end{bmatrix} \qquad y = [y_1, y_2] \qquad w = [w_0, w_1]$$

- Write $y = w.X^T$
- That leads to $w = y.(X^T)^{-1}$

But most of the time we have lot many data points as compared to number of attributes $(m >> n)$.
In such over constrained systems, inverse $(X^T)^{-1}$ does not exists

Direct method of MSE optimization

Gradient is zero at the point where loss is minimum.

Loss function being $J_w = (y - w.X^T)^2$ gradient is given by $\nabla J_w = -(y - w.X^T)X$. Let us find where it is zero

$$\begin{aligned} -(y - w.X^T)X &= 0 \\ y.X - w.X^T.X &= 0 \\ w.X^T.X &= y.X \\ w &= y.X.(X^T.X)^{-1} \end{aligned}$$

- here $X.(X^T.X)^{-1}$ is called as pseudo-inverse

With large data size m, it takes $O(m^3)$ time to compute. However, gradient descent would be much faster, taking only $O(m)$ time.

Normal Equation

- Consider X as *design matrix*, constructed using training data of size $m \times (n + 1)$
- And Y be the target values (matrix of size $m \times 1$)
- Weights could be computed directly using the calculus
$$w = y.X.(X^T X)^{-1}$$
- Computation of matrix inverse could need pseudo-inverse computation if $X^T X$ is non invertible (singular)
- This happens if features are linear combination of others, or the matrix is not square. Delete some features or apply regularization.
- No need to chose α
- Matrix computation is hard takes roughly $O(m^3)$ time. One can avoid normal equation if $m > 1000$

MSE

$$J_\theta = \sum (y - \hat{y})^2$$

- Use of MSE is computationally convenient.
- It measures the variance of residue
- Corresponds to the likelihood under Gaussian model of noise
- It is very sensitive to outliers

Issues with gradient descent

- Very general
- Gets stuck at local minima
- Difficult to find appropriate step size (learning rate α)

Thank You!

Thank you very much for your attention! (Reference[3])

Queries ?

[3] [1] Book - *Pattern Recognition And Machine Learning*, Bishop, Springer-2006 (CH-3), [2] Book - *Machine Learning*, ch-6, Tom M. Mitchell.