Introduction

Description

Concrete is a composite material that commonly used in civil engineering. It composed of blast furnace slag, cement, fly ash, coarse aggregate, water, superplasticizer, and fine aggregate. Different proportion of its ingredients and its age would affect concrete compression strength. A model could be trained through concrete ingredients and its age to predict the concrete compression strength.

Algorithm

The learning method for the model is Linear Regression. The object function is Mean Squared Error.

$$L(m,b) = \frac{1}{n}\sum_{i-1}^{n}\left(y_1 - (mx_i + b)\right)$$

To minimize the object function, gradient decent is used; but it's not a stochastic gradient decent. It's the basic gradient decent that add up all independent variables. The step size is 0.00001, because if the step size is too large, such as 0.01, the result of loss function would be too large and the algorithm might not stop, and if the step size is less than 0.00001, the result would be similar with 0.00001. Also, the algorithm would dynamically increase the step size by 1% to boost the process when the loss is getting smaller; and decrease by 50% when the loss is getting larger. For multivariate, the algorithm would stop when the norm of gradients is smaller than 0.01 or the total steps reaches 100000. For univariate, the algorithm would stop when the sum of gradients' absolute values is smaller than 0.01 or the total steps reaches 100000. The reason why to set a maximum number of steps is that sometime the algorithm takes too many times to stop, and the result is similar with smaller total steps.

pseudo-code

univariate:

```
1.  Y = all response variables
2.  For each column (feature) in the training set (except the column for response values):
3.      Set initial values to 10 for m, n, partial derivatives of m and b.
4.      Set initial step size to 0.00001
5.      Set initial total steps to 0
6.      Set previous loss to infinite
7.      X = all values in the column
8.      While the sum of absolute values of gradients of m and b is greater than 0.01 and the
        total steps is less than 100000
9.          Calculate the loss
10.         Calculate gradient for m and b
11.         Update m and b
12.         If loss is getting smaller:
```

```
13.                 Increase the step size by 1%
14.             Else:
15.                 Decrease the step size by 50%
16.             Update previous loss
17.             Increment total steps by 1
```

Multivariate:

```
1.  Set initial step size to 0.00001
2.  Y = Take out response variables from the training set and convert it to a k by 1 matrix
3.  X = all features that converted to a 900 by 9 matrix, where values in the last column are
    ones
4.  Initialize m and b with a 9 by 1 matrix, where all values are zeros. The last one is b, and the
    first eight values belongs to m
5.  Initialize norm to infinite
6.  Initialize previous loss to infinite
7.  Initialize total step size to 0
8.  While norm is greater than 0.01 and total step size is less than 100000:
9.       Calculate loss through matrix
10.      Calculate gradient through matrix derivatives
             $\frac{\partial L}{\partial w} = 2x^T xw - 2x^T y$
11.      If loss is getting smaller:
12.          Increment step size by 1%
13.      Else:
14.          Decrease step size by 50%
15.      Update previous loss
16.      Calculate norm
17.      Increment total step size by one
```

Quadratic regression

For the quadratic model, there are total 45 parameters, the way to calculate gradient for quadratic model is the same as the multivariate model, which uses matrix derivatives (see above multivariate pseudo code), except that the initial m and b matrix is 45 by 1 matrix, and the x is a 900 by 45 matrix. Also, the initial step size, stopping criterion is the same as the multivariate model.

Data pre-processing

For data pre-processing, mean normalization is used:

$$x' = \frac{x - average(x)}{\max(x) - \min(x)}$$

Results

Univariate – training dataset:
0.2251403916933733,
0.017208834458507183,
0.002035219755547102,
-0.10484359729916815,
0.17428066872317483,
-0.0593944897901626,
-0.08157346778484631,
0.11138226935028206

Univariate – testing dataset
0.428177689265937,
-0.11080362461782944,
-0.04016788505103275,
-0.19359745989442056,
-0.6373943771089985,
-0.12678408959247878,
-0.155141328462151,
-0.04991272677257608

Multivariate – training dataset:
0.6125262962907925

Multivariate – testing dataset:
0.5824270875813841

Normalized univariate - training dataset:
0.22825730980431724,
0.01720756289273273,
0.0020345680969763302,
0.08707171573913874,
0.17427893038475706,
0.03846511592413737,
0.032284714847082396,
0.1113803445732493

Normalized univariate – testing dataset:
0.43524948533304597,
-0.11120743278509915,
-0.041651072923905685,

-0.07804978983640254,
-0.6351833048587594,
-0.3219981190222716,
-0.17524492597165242,
-0.050414632080122646

Normalized multivariate – training dataset:
0.612662096066432

Normalized multivariate – testing dataset:
0.5743772318582852

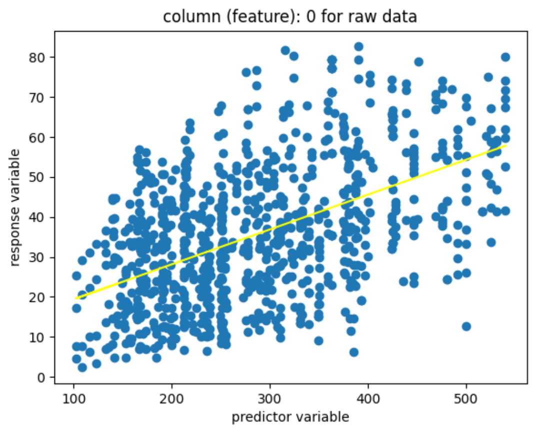
Quadratic – training dataset:
-1.1538788050289885e+177

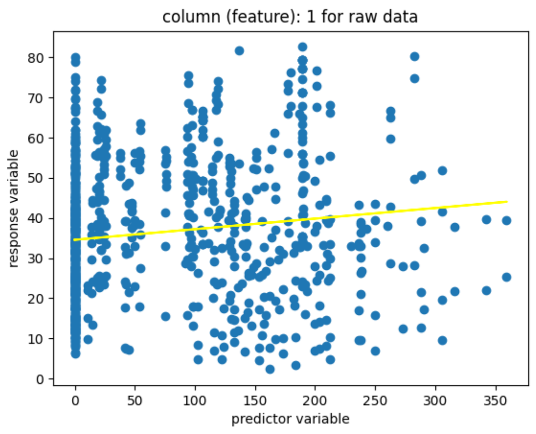
Quadratic – testing dataset:
-4.634861486073459e+177

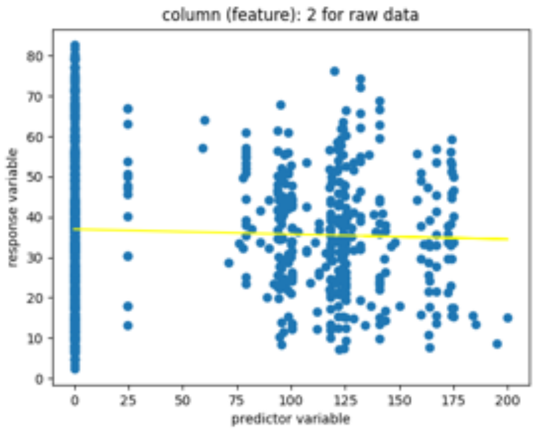
Normalized quadratic – training dataset:
0.8064320402935595

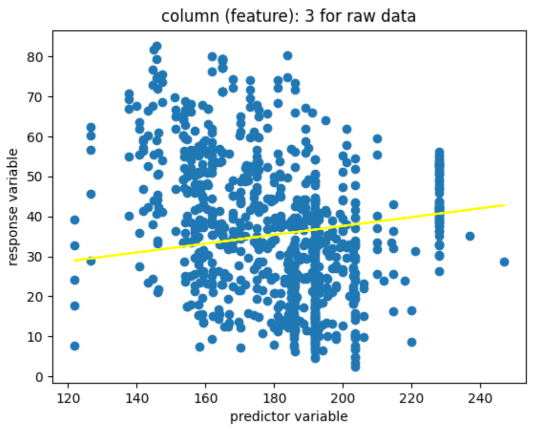
Normalized quadratic – testing dataset:
0.6822971978437784

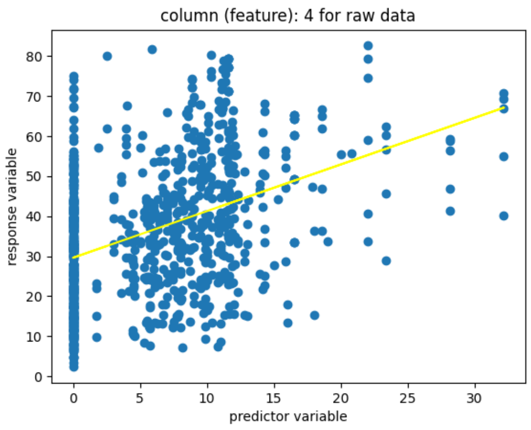
Raw data plots:

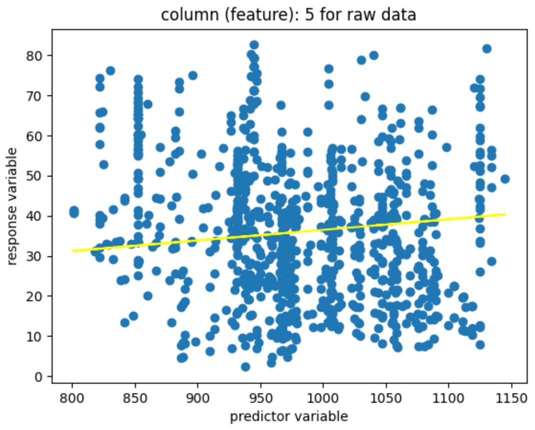


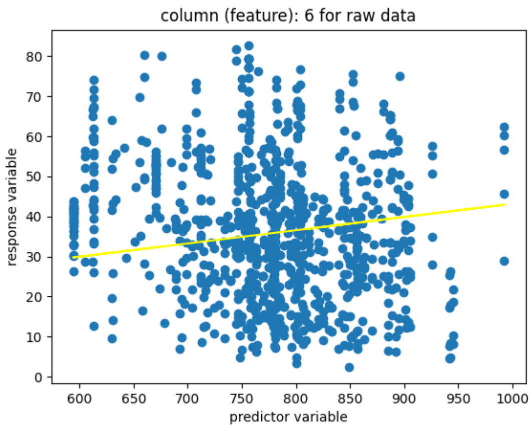


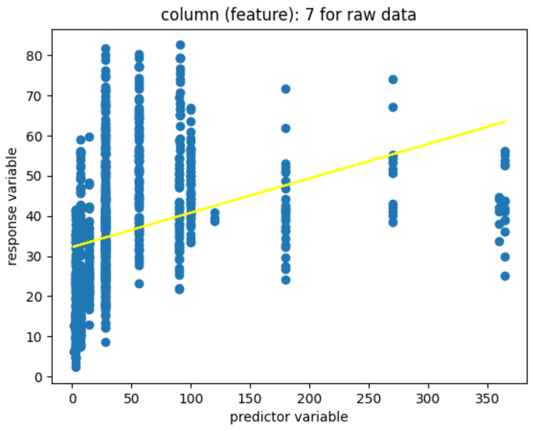

column (feature): 2 for raw data
response variable
predictor variable


column (feature): 3 for raw data
response variable
predictor variable

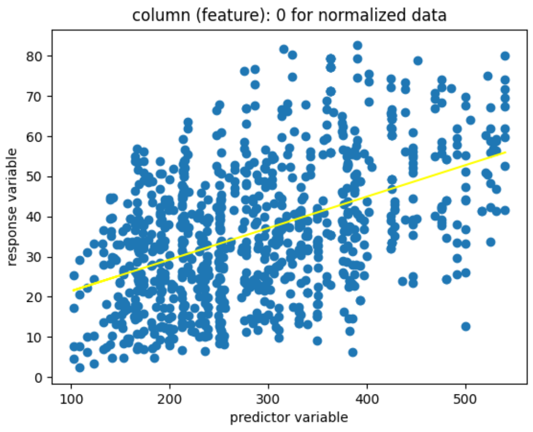

column (feature): 4 for raw data
response variable
predictor variable

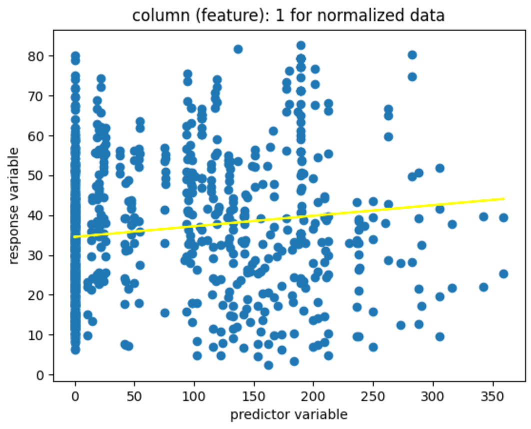

column (feature): 5 for raw data
response variable
predictor variable

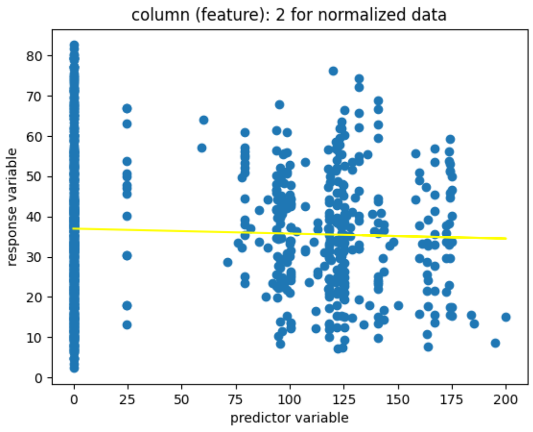

column (feature): 6 for raw data
response variable
predictor variable

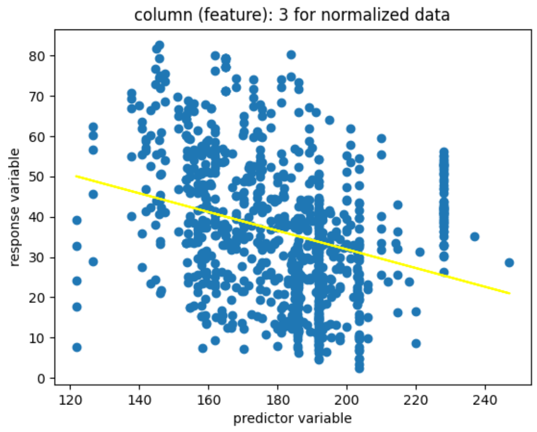

column (feature): 7 for raw data
response variable
predictor variable

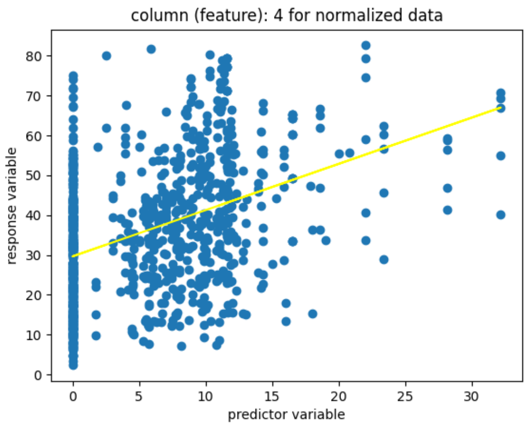
Normalized data plots:

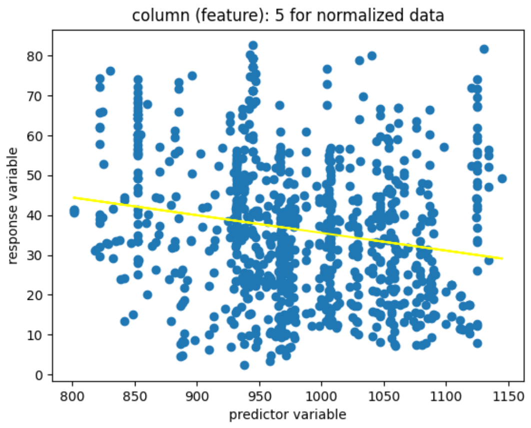


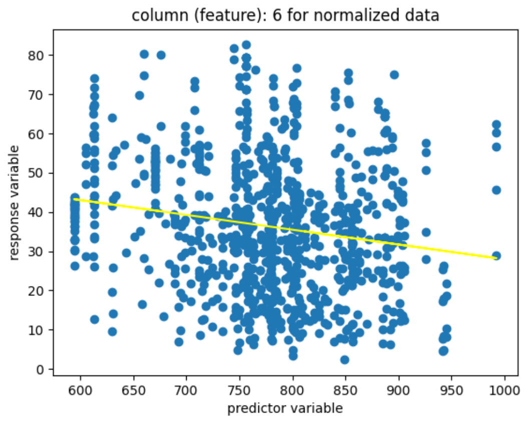


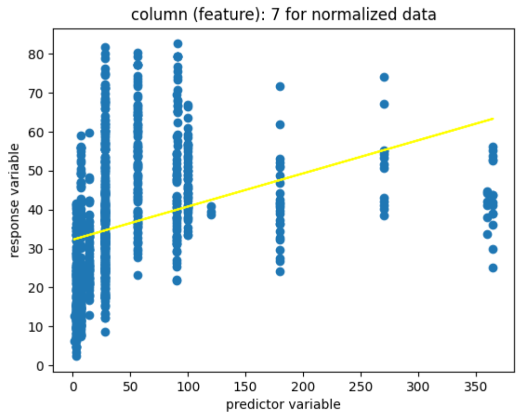












Discussion

For univariate, the trained model performs bad on the training data, since the variance explained is too small, and even includes negative values, the same model performs even worse on testing data, because most variance explained are negative. For normalized data set, the trained model performs better than raw data set, but this model still performs bad on testing data.

For multivariate, the trained model performs good on the training data; the variance is above 0.6. The variance explained for testing data is close to 0.6. So, the model performs also good on testing data, but not as good as on training data. For normalized data set, the result is almost the same as the raw data set.

For quadratic, the trained model performs bad on both training data and testing data, their variance explained are both negative, but for normalized data set, the trained model works very good on training data; the variance explained is above 0.8; and it also performs good on testing data, the variance explained is close to 0.7.

It seems that coefficients of the univariate model and multi-variate model are positively related.
Univariate coefficients:
0.08721521, 0.02635448, -0.01248033, 0.11060268, 1.16459584, 0.02639053, 0.03296995, 0.08552572
Normalized univariate coefficients:
34.33596132, 9.52444421, -2.46323379, -29.07299848, 37.35297789, -15.28652172, -15.03660799, 30.97964983
Multivariate coefficients:
1.12057212e-01, 9.59885865e-02, 8.86235566e-02, -1.88278264e-01, 2.42481346e-01, 1.19712462e-02, 1.05500854e-02, 1.15693361e-01, 3.58388304e-05

Normalized multivariate coefficients:
47.22625974, 32.76984422, 16.41139382, -23.81250255, 9.77073276, 3.40560017, 2.52173953, 41.94447562, 35.84429216

To make the hardest possible concrete, the key feature might be the cement, because in univariate models, this feature is the only one that trained models perform better on testing data than training data.