Programming Assignment 1

JIANJUN WEI

February 24, 2022

1 Introduction

1.1 Description of the problem

This project is going to find the factors that contributes to the hardest concrete. In this project, I design and implement gradient descent algorithms for uni-variate linear regression model, multi-variate linear regression model and polynomial regression model(quadratic regression model in this project) to analyze each feature and draw the conclusion.

For uni-variate linear regression model, we will follow the function

$$y = f(x) = mx + b$$

where m and b are parameters and x is the only variable. We will calculate mean squared error in our algorithm. Also, it will be applied on our gradient descent algorithm to optimize the uni-variate linear regression model.

For multi-variate linear regression model, we will expand our input as a vector of features with length p. Combined with b parameter, there would be $p + 1$ parameters in our model. Our model could be described as

$$y = f(x) = (m \cdot x)$$

, which means dot product between m and x. In this function, $m = (m_0, m_1, \ldots, m_p)^T$ and $x = (x_0, x_1, \ldots, x_p)^T$. Mean squared error objective function would also be used for optimization of our linear regression model.

For polynomial regression model, we may follow the function

$$y = X \cdot m$$

where X is a Vandermonde matrix while $m = (m_0, m_1, \ldots, m_p)^T$. In this project, we would design and implement a multi-variate quadratic regression model, which means the highest order of the Vandermonde matrix X is 2. Besides the mean squared error objective function, R^2 function, which represents the proportion of the variance for a dependent variable that's explained by an independent variable or variables in a regression model, would also be applied to indicate how much variation of a dependent variable is explained by the independent variable(s) in a regression model.

We will use the Concrete Compressive Strength dataset in the UCI repository[1]. The first 900 instances would be used for training while the rest are for testing.

1.2 Details of Algorithm

Our algorithm is based on matrix computation. Although our gradient descent algorithm is not stochastic, we still get much greater training and testing results.

For uni-variate linear regression model, we set the initial learning rate 0.00001. Since one independent feature may have much larger impact on the observed output, we set a smaller learning rate so that it could be slowly descended without missing the local minimum. However, when training multi-variate and polynomial regression model, the initial learning rate we choose is 0.0001. The reason why it is a little larger than the uni-variate linear regression model is that these two models take all features into consideration, which means that they will have less impact on the output observed data.

[1]https://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength

Thus, we could learning a little bit fast than the uni-variate regression model. The learning rate is adaptive in our algorithm when we train the data. We adjust it according the difference between the value of two mean squared error(MSE). When the new MSE is less than the previous one, we would accelerate our training process by setting it into 120 percent of its original value. Otherwise, we set it into 75 percent since the difference between the new MSE and the previous one now is a signal that we are moving too fast.

For all three models, we set the max steps as 100000. When the difference of the two MSE is less than 1e-18, or we have already moved 100000 steps, we will stop training since the loss function is approaching convergence in this situation and would not have too much change later.

Since our algorithm is based on matrix computation, the derivative equation might be slightly different from we have learnt in class. They actually have the same meaning, but different in representation. Our derivative equation will be:

$$\frac{\partial J}{\partial M} = (M^T X^T X - y^T X)$$

where $M = (m_0, m_1, \ldots, m_p)$. And thus, we will update the parameters according to the following function:

$$M_{k+1} = M_k - \alpha \frac{\partial J}{\partial M}$$

1.3 Pseudo-code of algorithm

Here is the pseudo-code of descend gradient algorithm:

Algorithm 1 Pseudo-code of our descend gradient algorithm

```
Require: $traininginput, parameter, trainingoutput, learningrate, maxstep$
  size ← count of training data
  previous loss ← $Inf$
  diff ← $Inf$
  $step$ ← 0
  while diff > 1e-18 and max step < 100000 do
      calculate derivative matrix
      update parameter matrix $M$
      calculate loss function MSE
      calculate difference between previous MSE and the new one
      adjust learning rate according to the previous MSE and the new one
      $step \leftarrow step + 1$
      set previous loss
  end while
```

The following part will be the whole process of optimized regression model algorithm:

Algorithm 2 Pseudo-code of optimized regression model algorithm

```
Require: dataset
  Splitting data into training set and testing set
  Pre-processing training set
  if uni-variate linear regression then
      training model for each features
      testing model according to training results
  else if multi-variate linear regression then
      training model for multi-variate linear regression
      testing model according to training results
  else if polynomial regression then
      training model for polynomial regression
      testing model according to training results
  end if
```

1.4 Normalize the data

We would like to normalize our input data in our all three regression models. Since the output would only be influenced by single variable in uni-variate linear regression, there would be just slightly difference after normalization. However, when dealing with multiple variables, the features have different units and they all contribute to the output data. Thus, we need to normalize all the features to avoid the situation that the output will be over dependent on one or some features.

In this project, we would like to normalize each feature as the following function:

$$x\_norm = \frac{x - \mu}{s}$$

where μ is the mean of the feature and s is the standard deviation.

Figure 1 and Figure 2 are the histograms that illustrate how the distribution of a feature variable's values changed. In Figure 1, we do not pre-processing the data and we could see the range of the value would be [0, 1200]. However, after we normalize the input data, we could see in Figure 2 that the range comes to [-3, 5], which would be much more helpful to grasp the real features and train the correct and useful model.

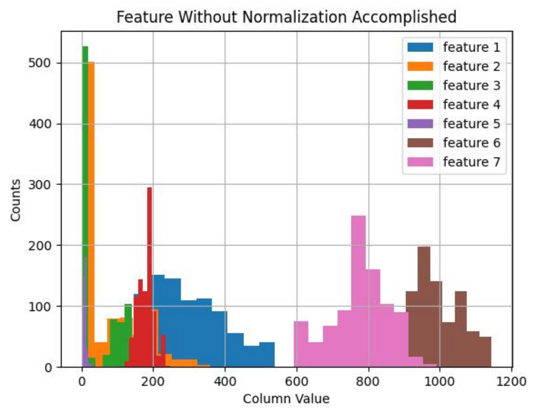

Figure 1: Data without normalization

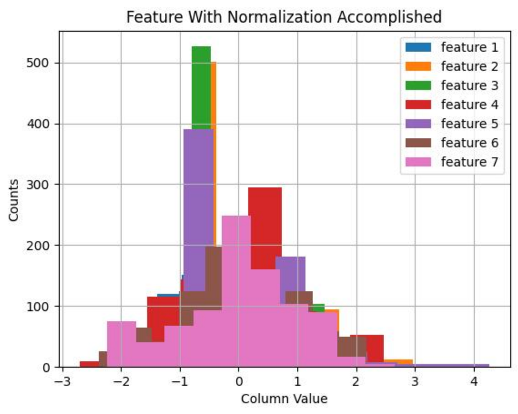

Figure 2: Data with normalization

Feature	MSE(norm)	R_square(norm)	MSE(without norm)	R_square(without norm)
Cement	241.214	0.185	241.216	0.185
Blast Furnace Slag	290.773	0.0172	290.773	0.0172
Fly Ash	295.262	0.002	295.262	0.002
Water	343.997	-0.162	344.001	-0.162
Superplasticizer	244.301	0.174	244.301	0.174
Coarse Aggregate	322.252	-0.0891	322.252	-0.089
Fine Aggregate	333.095	-0.125	333.095	-0.125
Age	262.910	0.111	262.910	0.111

Table 1: Results for uni-variate linear regression model on training data set.

2 Results

In this section, we put much more emphasis on the results. Here we may focus on the "R_square", which could be defined as:

$$1 - \frac{MSE}{Variance(observed)}$$

2.1 Variance explained of models on the training data set

For uni-variate linear regression model, the result has shown in Table 1.

For multi-variate linear regression model, the result has shown in Table 2. We have already normalized the training data set.

	MSE	R_square
with norm	115.051	0.611
without norm	114.629	0.612

Table 2: Results for multi-variate linear regression model on training data set.

For polynomial regression model, the result has shown in Table 3. Also, we have already normalized the training data set and more precisely, in this project we mainly focus on the quadratic regression model.

	MSE	R_square
with norm	56.584	0.808
without norm	55860.771	-187.805

Table 3: Results for multi-variate quadratic regression model on training data set.

2.2 Variance explained of models on the testing data set

For uni-variate linear regression model, the result has shown in Table 4.

Feature	MSE(with norm)	R_square(with norm)	MSE(without norm)	R_square(without norm)
Cement	98.996	0.311	99.313	0.308
Blast Furnace Slag	159.685	-0.111	159.687	-0.111
Fly Ash	149.467	-0.039	149.480	-0.040
Water	179.496	-0.248	179.599	-0.249
Superplasticizer	235.338	-0.637	235.338	-0.637
Coarse Aggregate	165.900	-0.154	165.900	-0.154
Fine Aggregate	171.355	-0.192	171.346	-0.192
Age	150.919	-0.050	150.924	-0.050

Table 4: Results for uni-variate linear regression model on training data set.

For multi-variate linear regression model, the result has shown in Table 5. In addition, the testing data set has already been normalized.

	MSE	R_square
with norm	64.393	0.551
without norm	60.07	0.582

Table 5: Results for multi-variate linear regression model on testing data set.

For polynomial regression model, the result has shown in Table 6. Also, the testing data set is normalized. And, more precisely, in this project we mainly focus on the quadratic regression model.

	MSE	R_square
with norm	46.565	0.675
without norm	105424.342	-732.543

Table 6: Results for multi-variate quadratic regression model on testing data set.

2.3 Plots of uni-variate models

Figure 3 to Figure 10 show the images of uni-variate models. The trained uni-variate models are printed on top of scatter plots of the training data used.

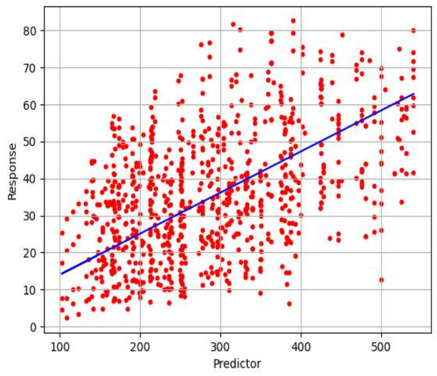

Figure 3: Feature: Cement, uni-variate linear regression

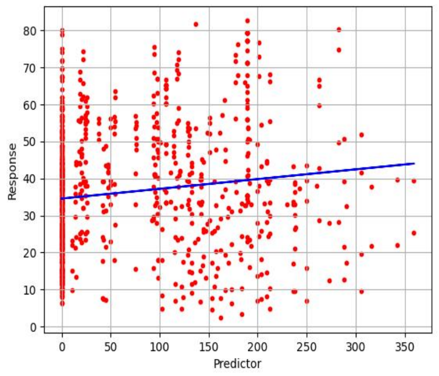

Figure 4: Feature: Blast Furnace Slag, uni-variate linear regression

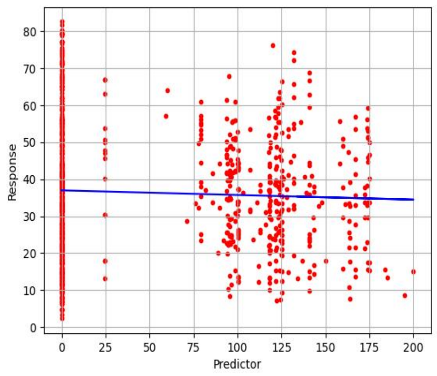

Figure 5: Feature: Fly Ash, uni-variate linear regression

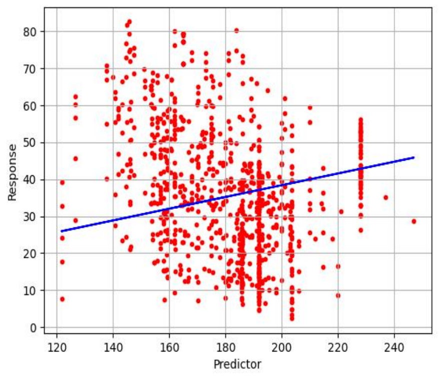

Figure 6: Feature: Water, uni-variate linear regression

3 Discussion

3.1 Describe models compared in performance on the training data

We have trained the data set from three models, the uni-variate linear regression model, the multi-variate linear regression model and quadratic regression model. They all have different performance.

From the result, we could see that quadratic model gets much lower loss function and much higher R_square value compared to the linear regression. The reason may lie in the fact that quadratic model has much more complex function which allows it to learn or match the features we provide. However, although the multi-variate linear regression model takes all features into consideration, the model did not take care the high dimensional data, which means the model could not learn features with high dimension well. For uni-variate model, since it only consider one variable a time, it performs not as good as multi-variate regression model and quadratic model.

For all three models we choose in this project, the coefficients trained from the training data points performs well on testing data, but the result of testing data is a little bit different from the result of training data. One of the reason is that we have 900 data for training but 130 data for testing. When

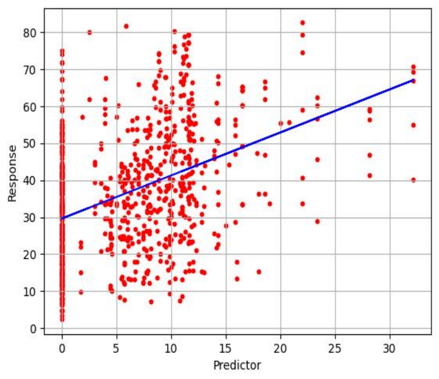

Figure 7: Feature: Superplasticizer, uni-variate linear regression

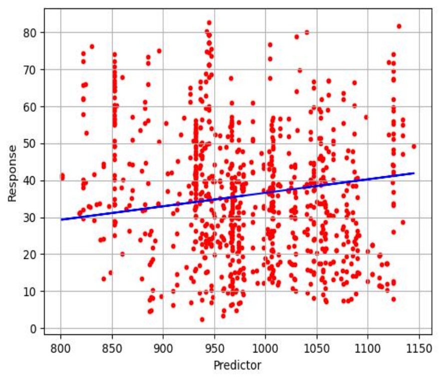

Figure 8: Feature: Coarse Aggregate, uni-variate linear regression

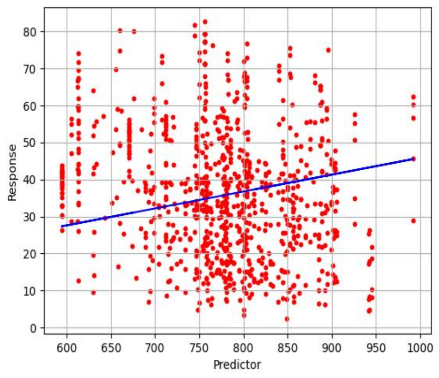

Figure 9: Feature: Fine Aggregate, uni-variate linear regression

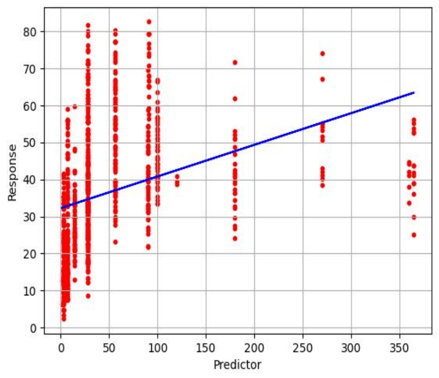

Figure 10: Feature: Age, uni-variate linear regression

we have more data, regardless of training data or testing data, our model could have more chance to learn feature. As a result, we could get much more precise coefficient and performance would be improved.

3.2 Describe the coefficients of the uni-variate models

We could know that the coefficients of the uni-variate models fail to predict the coefficients in the multi-variate model.

In uni-variate regression model, we separate all features into individual one. It is true that the model learns the influence from each feature to the output data, but they are independent and we did not take the combination of these features into consideration. This means that we ignore some features that we should have learned. As a result, the coefficients of the uni-variate models did not perform well when predict the coefficients in the multi-variate model.

3.3 Draw some conclusions about what factors predict concrete compressive strength

We could know from the project the the more correct data we have, the higher possibility that could be to help us precisely predict the concrete compressive strength. In addition, a powerful model also contributes to the prediction of the result. A powerful model means that the model has a much more powerful learning ability that could be adaptive to more complex data input and various features.

Besides, chosing the suitable learning rate and normalization function would be speed up the training process and improve the training results.

From my point of view, it is the cement, water, blast furnace slag and fine aggregate that contributes to the hardest possible concrete. So when we hope to make the concrete, we may put more emphasis on these ingredients so that we could make a much harder concrete.

3.4 Comparisons to the results from normalized

Usually, we normalize the training data points when we need to take several features into consideration. We could see from the histogram in the previous section and find that the range of the data is much larger and the value of every feature vary from each other. When we start regression without data normalization, the value of each feature will be larger especially when we combine several feature together. And the result is that we get a huge MSE when we start training and need to take a much longer path to descend the gradient to get the parameters we need, which will take much more time to grasp the feature that really contributes to the hardest concrete.

We could see that although the results with normalization and the those without normalization in our uni-variate and multi-variate linear regression model are similar, there is a huge difference in our quadratic model. The reason is that we have already do enough steps for data without normalization in our first 2 models and they reach out the condition that could hardly descend in gradient, however, since the quadratic model is much more complex, the max steps for first two model is not satisfied and we need a much smaller learning rate and more steps to train the data so that the loss function could become much more flat.

In addition, when we normalize the data, we limit the range of all features in a much smaller value. This may help us quickly find the correct and proper parameter that satisfy the training data and features since data now is more concentrated and we have a much shorter descend path when we running descend gradient algorithm.

4 Code

This project is written in Python. The input data is the CSV file which contains all the data we need. The output will be the images we may need to be used to analyze and some metrics that we calculated.

By setting the correct path of the input CSV file, the program could be run just by a simple click of mouse.

4.1 Source Code

For the source code, I have already updated on the github. You could get the code as well as the data set from `https://github.com/weijj27/514-Programming-Assignment`.