Joshua Gabella
CSE 514A
Project Implementation I
2/23/2022

Introduction

The Concrete Compressive Strength dataset, made public in the UCI repository, labels measured compressive strengths for different mixes of concrete. It offers 1030 data points and eight features: Cement (kg/m^3), Blast Furnace Slag (kg/m^3), Fly Ash (kg/m^3), Water (kg/m^3), Superplasticizer (kg/m^3), Coarse Aggregate (kg/m^3), Fine Aggregate (kg/m^3), and Age (days). With a linear regression model, we could uncover weights that determine relative importance values of these "ingredients" in amplifying the compressive strength of concrete. In this report, I will build nine linear regression models – eight univariate models (one for each feature) and one multivariate using all features – using gradient descent. I will repeat this process after zero-means normalizing the data for a combined total of 18 linear models. I will compare each model's time and iteration taken to converge and their performance on both test and training sets as measured by Mean-Squared Error (MSE) and Variance Explained.

As mentioned above, I used gradient descent to optimize the weights of my regression model. While I toyed with stochastic gradient descent, I ended up not using singular data points or randomly selected batches to calculate error because I didn't need to. Stochastic gradient descent in expectation performs as well as gradient descent, however in reality may come with performance hits with the benefit of added time. Given the size of the dataset and the hardware accessible to me, I determined I would not risk the performance. My implementation of gradient descent works as follows:

```
For t in range of maximum iterations allowed:
      w_{t+1} = w_t - stepsize*∇_w(MSE(w_t))
      Calculate MSE(w_{t+1})
      if MSE(w_{t+1}) > MSE(w_t):
            w_{t+1} = w_t                    #Revert w back to a vector with less worse performance
            stepsize = 0.5*stepsize          #Reduce stepsize by half
            if stepsize < 1e-14:
                  stepsize = 1e-14           #Induce minimum stepsize for gradient descent
      else:
            Calculate ∇_w(MSE(w_{t+1})
            if ||∇_w(MSE(w_{t+1})||_2 < Allowed Tolerance:
                  return w_{t+1}, t          #If gradient gets too small, return weight vector
            stepsize = 1.01*stepsize         #Increase stepsize by 1%
```

The weight vector is initialized to all zeros. Each data point is joined with a leading bias term of 1 so that there are a total of nine dimensions in both the weight vector and any given data point. It is important to note that I do not normalize the gradient and incorporate its l2-norm into my stepsize. Because mean-squared error is a convex function with respect to **w**, a large gradient may indicate distance from the optimum, and so a large stepsize is appropriate. If the gradient becomes very small, we may be approaching the optimizing solution, and so a smaller stepsize is appropriate. I also implement a variable stepsize that is halved when an updated weight vector returns a larger loss than the previous weight vector iteration. If error continues to decrease, the stepsize coefficient will increase by 1% for each successful iteration. This is a common trick that I learned from CSE517A – Machine Learning. I used the following stopping criteria: a maximum number of descent iterations equal to 10000 and a minimum l2-norm of the gradient equal to 1e-03. To zero-means normalize the dataset, I used SciKitLearn's Standard Scaler object from their preprocessing package. Normalizing in this fashion preserves the relative distribution but centers the data around zero and reduces the scope of values, as shown by the figures below.

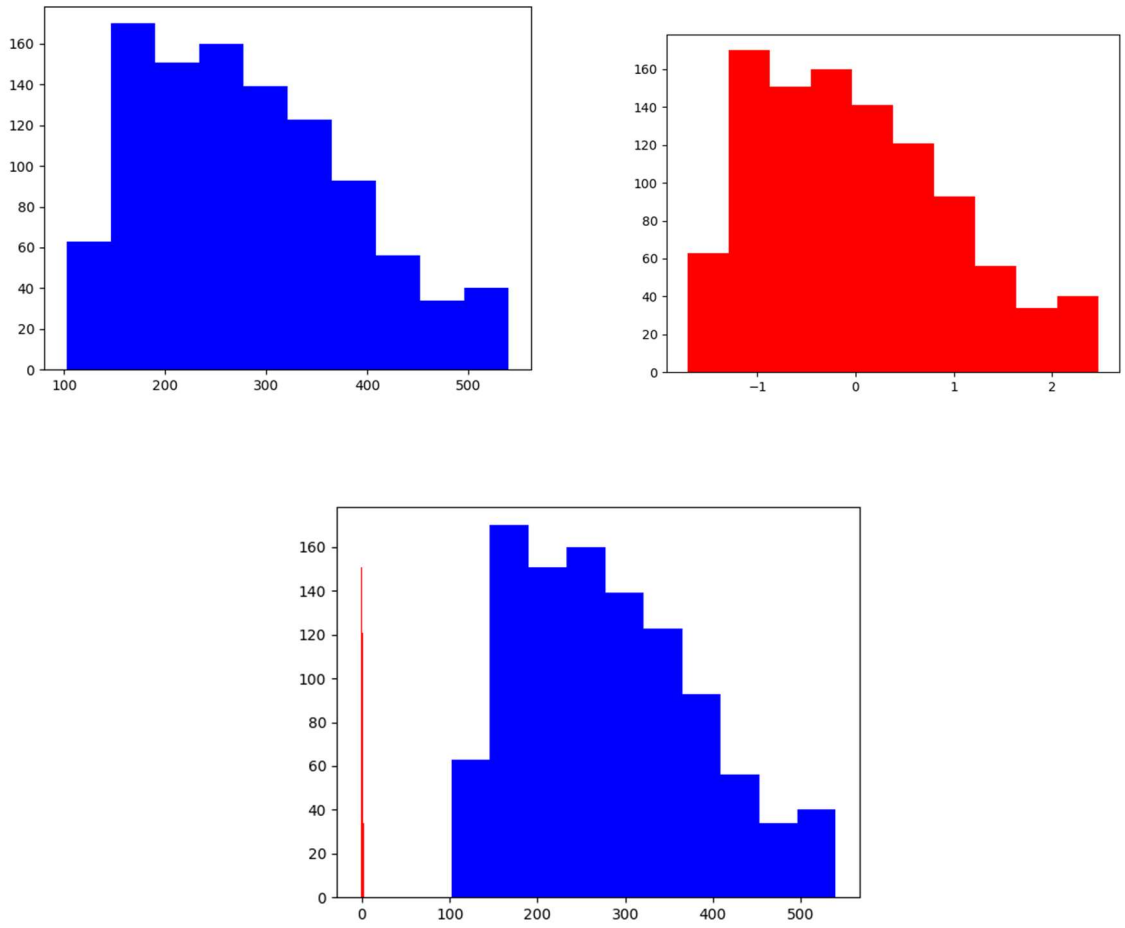

Figure 1. Distribution of Cement density used in mixtures, normalized (Red) and non-normalized (blue).

Results

Below are the MSE and Variance Explained (highlighted in blue) values for all eight univariate models and the multivariate model. I've also included the time it took for these models to converge and the number of iterations. Most notably, the Superplasticizer model is the only model here to not use 10,000 iterations.:

	Cement	B.F. Slag	Fly Ash	Water	S.plastic.	C. Agg.	F. Agg.	Age	Multivar
MSE on Training Set	248.44786	394.2151	313.1691	356.79103	244.3012	322.33761	333.4061	279.5809	115.2863
MSE on Test Set	107.57058	136.3437	178.4746	187.53744	236.8687	167.21631	172.5236	146.3067	59.22922
Variance Explained on Training Set	0.1602655	-0.33242	-0.05849	-0.205926	0.174281	-0.089476	-0.12689	0.055038	0.610341
Variance Explained on Test Set	0.2560805	0.057096	-0.23427	-0.296942	-0.6381	-0.156408	-0.19311	-0.0118	0.590392
Number of Iterations	9999	9999	9999	9999	1116	9999	9999	9999	9999
Time (s) to Convergence	66.535339	66.03295	65.9793	66.029317	7.391164	65.773664	65.74447	72.77606	65.95726

Table 1. Performance Metrics of Univariate and Multivariate Models on Non-normalized Data

The time taken, both in iterations and in seconds, for a model to converge becomes an important metric when looking at models trained on normalized vs. non-normalized data. When training on zero-means normalized data, none of the univariate models took more than 600 iterations to converge, and the multivariate model took fewer than 1000. Each model converged roughly 15 times faster than when dealing with non-normalized data. On top of time efficiency, some models gained performance as well. Models that did worse, were not too far off of where they were.

	Cement	B.F. Slag	Fly Ash	Water	S.plastic.	C. Agg.	F. Agg.	Age	Multivar
MSE on Training Set	228.33	290.8	295.2626	270.1026022	244.3012	284.4838728	286.3124	262.9107	114.5825
MSE on Test Set	80.633	160.8	150.6187	156.1887045	236.8647	191.5448553	170.2476	152.0336	60.8647
Variance Explained on Training Set	0.2283	0.017	0.002035	0.087074162	0.174281	0.038466583	0.032286	0.111382	0.61272
Variance Explained on Test Set	0.4424	-0.11	-0.041625	-0.08014507	-0.63807	-0.32465553	-0.17737	-0.05141	0.579082
Number of Iterations	572	573	575	570	573	579	570	569	948
Time (s) to Convergence	3.8605	3.846	3.855614	3.864525795	3.823482	3.879553795	3.775427	3.766661	6.291375

Table 2. Performance Metrics of Univariate and Multivariate Models on Preprocessed Data

The next step in comparing models was to plot models over the scatter plots. This helps visualize performance of a model fitting its data. There are eight subfigures in the figure below. They read left to right, top to bottom. The order of models shown on the data is as follows (from left to right, top to bottom): Cement, Blast Furnace Slag, Water, Superplasticizer, Coarse Aggregate, Fine Aggregate, and Age. Horizontal values are in kg/m^3 for the first seven plots, and in days for the last plot. The vertical output values are the concrete's compressive strength in megapascals (MPa).

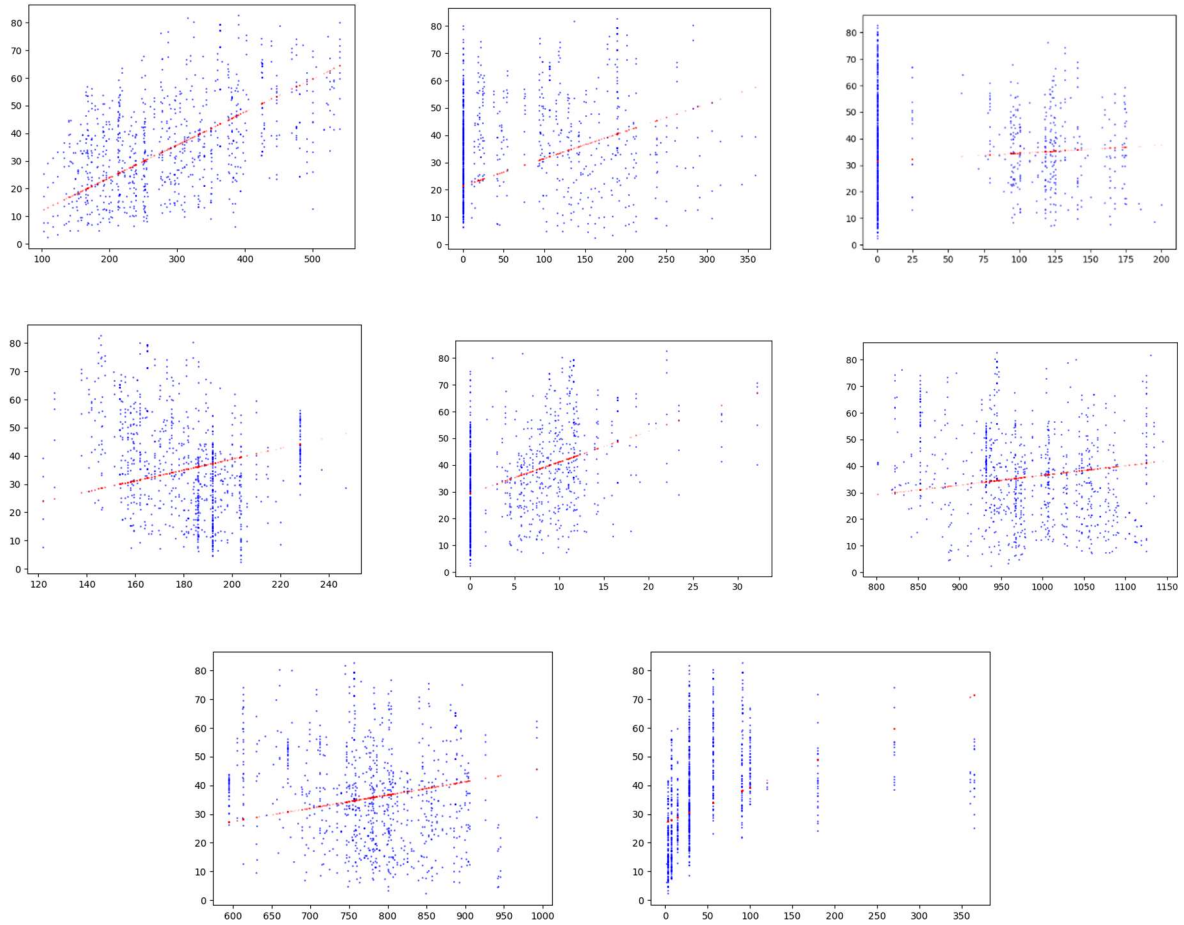

Figure 1. Plots of Eight Trained Univariate Models (Red) on Scatterplots of Non-normalized Data (Blue)

These plots can be compared to the following, which shows the same models built on normalized data. The model plots appear in the same order and feature the same data scheme, except for the

horizontal axis which shows relative distribution by mean and variance as opposed to raw values of concrete components in different mixes.

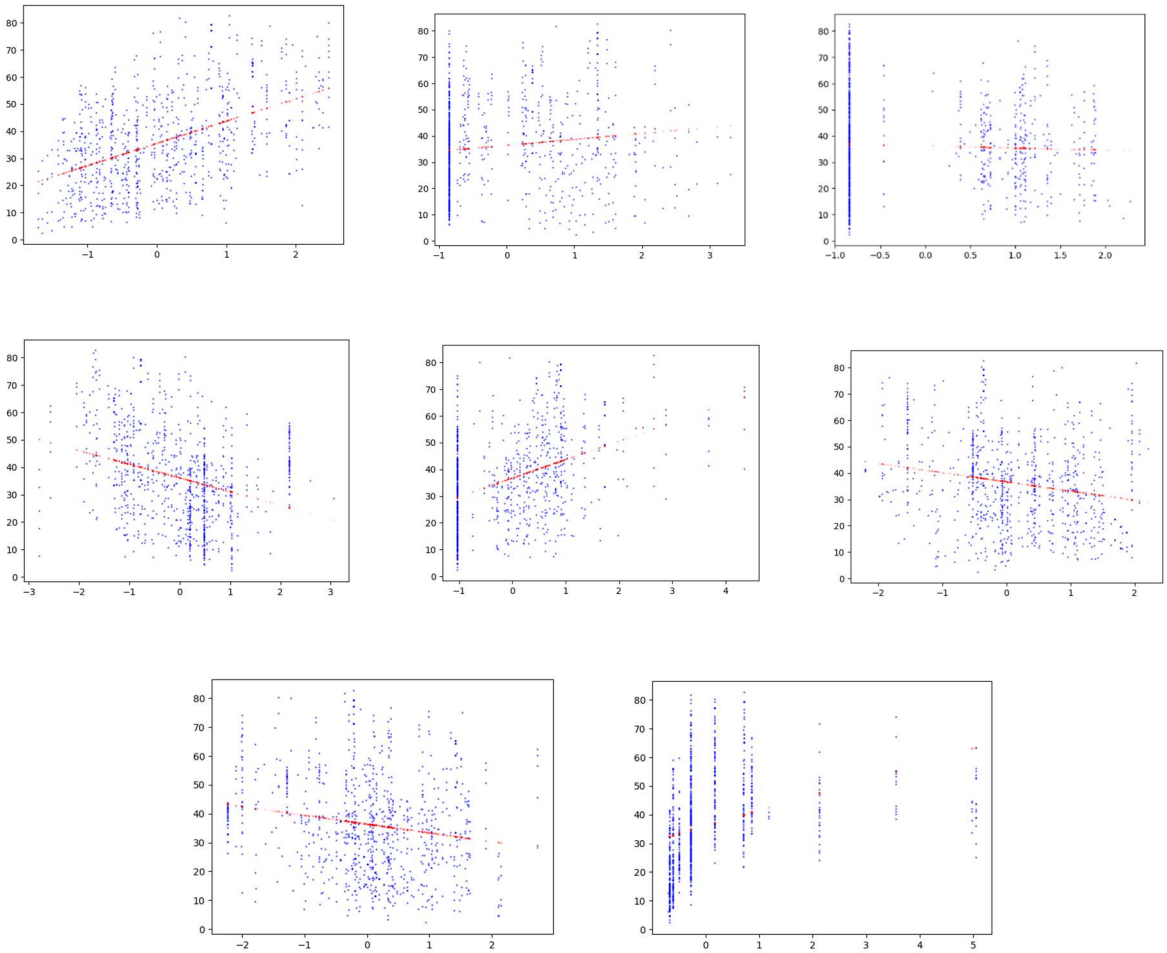

Figure 2. Plot of Eight Trained Univariate Models (Red) on Scatterplots of Normalized Data (Blue)

Discussion

There is no clear trend of the models performing better on the training data than the test data. Normally we expect this to happen as it is difficult to prevent overfitting a model to its training data. But both the Cement and Blast Furnace Slag models performed better on the test data than the training data. In the normalized data set, only the Cement model performed better on the test data than the training data.

Interestingly enough, performance on normalized data was consistently better than performance on non-normalized data. Time to convergence was a massive benefit, but on normalized data, the course aggregate model has a variance explained of -0.32. In the non-normalized data, it has a variance explained of -.16. The multivariate, fine aggregate, and superplasticizer perform roughly similar on both normalized and non-normalized data. The cement, fly ash, and water models take a performance increase while the slag, age, and course aggregate models decline significantly in performance.

Looking at Table 3, we can see that only certain univariate models were good predictors for the coefficients in the multivariate model. For the non-normalized data, all model coefficients were reasonably close to the multivariate counterparts, with the exceptions of water and superplasticizer.

Looking at the table of coefficients for normalized data, there appear to be more differences between the univariate and multivariate models. Cement, Water, and Age seem to be reasonably consistent across both model types. The univariate models that are poor predictors of multivariate coefficients display clear dependence of mix ingredients interacting with each other.

	Univariate Coefficients	**Multivariate**	**Univariate Coefficients (Normalized)**	**Multivariate (Normalized)**
Cement	0.119	0.117	8.21	11.6
B.F. Slag	0.100	0.102	2.27	8.17
Fly Ash	2.99e-02	9.80e-02	-0.802	5.51
Water	0.192	-0.208	-4.98	-3.75

Superplasticizer	1.16	7.43e-02	6.95	1.86
Coarse Aggregate	0.0366	1.14e-02	-3.47	1.03
Fine Aggregate	0.0459	1.41e-02	-3.04	0.83
Age	0.121	0.117	05.40	7.28

Table 3. Linear Coefficients by Feature for Normalized and Non-Normalized Data

Looking at the table above, cement has the largest positive correlation with compressive strength of a concrete mix, and water has the most negative correlation with compressive strength. This is also clear looking at the slopes of the linear models in their univariate plots. While I have not manipulated the data enough to give you relative amounts to optimize compressive strength of cement, more cement and superplasticizer, not too much water, and increased aging would be my best guesses for optimizing compressive strength.