Supervised Learning Algorithms Applied to Compressive Strength of Concrete

Elizabeth Dworkin

Abstract

This document contains a project applying three supervised learning approaches: Kernel Ridge Regression, K-Neighbors Regression, and a Neural Network to predict the compressive strength of concrete. A comparison of these methods is applied, and Kernel Ridge Regression is found to be the best suited for data of this kind. The code associated with this project is in Python 3, and all results discussed here can be duplicated with the attached files.

1 Introduction

The topic of the analysis is on predicting the compressive strength of concrete. I chose this topic because I wanted to study regression of materials relating city infrastructure and civil engineering. Concrete is arguably the most important substance in the construction of the infrastructure of our cities. Furthermore, the compressive strength of concrete is a well-known non-linear function of its age and ingredients, making this the perfect challenge for non-linear machine learning regression algorithms.

The dataset contains attributes that contribute to the compressive strength of concrete from the UCI machine learning library[1]. The multivariate dataset consists of 1030 datapoints across 9 attributes relating to the age of the concrete and its contents. Since the dataset was donated in 2007, over ten years ago, now is the best time to be studying the compressive strength of concrete in areas of cities that were constructed a few decades ago.

[1] UCI Compressive Concrete Dataset available at http://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength

Figure 1 displays the variables for each attribute in the dataset, in the order in which they are in the dataset .csv file. All attributes are components of the concrete (ingredients) except for age, which is a number indicating the age of the concrete. All data is numerical, either float or int type.

The dataset required little cleaning. A few values were undefined, so I ran a function on all data to replace any undefined values with average values of that category. I randomly sorted 70% into the training set, with the remaining 30% was randomly divided equally among the test set, used for tuning hyperparameters, and the validation set aside and only used to measure performance accuracy.

Name (Number)	Data Type	Unit	In/Out
Cement (1)	Quantitative	Kg in m3 mixture	Input
Blast Furnace Slag (2)	Quantitative	Kg in m3 mixture	Input
Fly ash (3)	Quantitative	Kg in m3 mixture	Input
Water (4)	Quantitative	Kg in m3 mixture	Input
Superplasticizer (5)	Quantitative	Kg in m3 mixture	Input
Coarse Aggregate (6)	Quantitative	Kg in m3 mixture	Input
Fine Aggregate (7)	Quantitative	Kg in m3 mixture	Input
Age (8)	Quantitative	Day (1-365)	Input
Compressive Strength (9)	Quantitative	MPa	Output

Figure 1: Dataset Details.

Evaluation Metric: A commonly used metric of evaluating supervised learning regression problems is mean squared error (MSE). This is the metric that is used to compare the accuracy and performance of all algorithms implemented in this paper. As a non-negative indicator of the quality of regression, smaller MSE values indicate a higher quality algorithm, with zero representing perfectly accurate predictions.

The formal definition of MSE for prediction is:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(h(x^i) - y^i)^2 \qquad (1)$$

Where n is the number of data points, h(x) is the hypothesis class for the x features, and y is the target labels.

2 Description of the Algorithms Used

This first algorithm is Kernel Ridge Regression (KRR), which utilizes a linear function on data. In this case, we apply a non-linear kernel with KRR effectively learn a non-linear regression function in the original space. KRR is a combination of using a kernel with linear least squares and 12-norm regularization [2]. KRR hyperparameters include the following: [3]

- **Alpha** – a positive float to reduce variance
- **Kernel** – can be many types, but for this implementation it is a string that will be passed into the pairwise kernel functions (PAIRWISE_KERNEL_FUNCTIONS) method to generate the kernel. The valid non-linear values are [‘additive_chi2’, ‘chi2’, ‘poly’, ‘rbf’, ‘laplacian’, ‘sigmoid’, ‘cosine’].
- **Gamma** – a float parameter for the kernels (RBF, laplacian, exponential chi2, and sigmoid only)
- **Coefficient** – for the sigmoid kernel only this applies. The default value is 1, which for simplicity I do not alter unless testing indicates sigmoid is the best kernel.

Secondly, K-nearest-neighbors (KNN) is a simple but effective regression method. We select an integer k which is between 1 and the size of the dataset. For every k closest datapoints to a given datapoint x, we sum the labels of these k nearest neighbors and assign the mode of these labels as the prediction for x. Essentially, we assume that a label can be predicted based on the most common of the k nearest datapoint’s labels.

In this implementation of KNN, I chose to use Euclidean distance for simplicity. The k parameter needs tuning, so to determine the best k value we need to run a test plotting the loss of KNN graphing both and test and training data against increasing values of K. This cross validation is graphically displayed in the following section. A k equal to 1 would greatly overfit the data and hinder prediction for the test set, while a k equal to the size of the dataset would always return the most commonly occurring label. So, our experiment to find a good k value locates a good balance between this tradeoff.

Thirdly, I use a neural network (NN). Neural nets learn patterns in data by mimicking neural structures in the brain, where each node is responsible for only a small piece of the data. This is a deep neural network utilizing the Pandas[4], sklearn[5], and Keras[6] libraries. This neural network uses the RELU activation function and contains three hidden RELU layers. The general structure and code of the neural net is based on this[7] page from Towards Data Science. You can see this visually in figure 2.

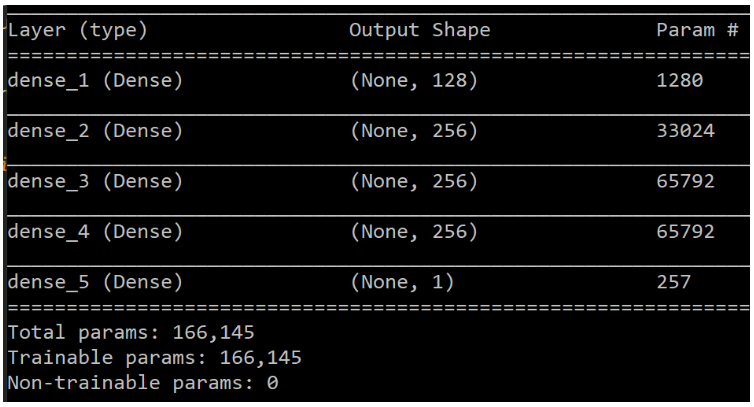
```
Layer (type)                 Output Shape              Param #
================================================================
dense_1 (Dense)              (None, 128)               1280
________________________________________________________________
dense_2 (Dense)              (None, 256)               33024
________________________________________________________________
dense_3 (Dense)              (None, 256)               65792
________________________________________________________________
dense_4 (Dense)              (None, 256)               65792
________________________________________________________________
dense_5 (Dense)              (None, 1)                 257
================================================================
Total params: 166,145
Trainable params: 166,145
Non-trainable params: 0
```

Figure 2: Neural Net Structure.

[2] “Machine Learning: A Probabilistic Perspective” Murphy, K. P. - chapter 14.4.3, pp. 492-493, The MIT Press, 2012

[3] This information detailed at https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/kernel_ridge.py

[4] https://pandas.pydata.org/docs/

[5] https://scikit-learn.org/stable/

[6] https://keras.io/

[7] Towards Data Science Neural Net Regression: https://towardsdatascience.com/deep-neural-networks-for-regression-problems-81321897ca33

3 Non-Linearity of the Data

Before diving too deeply into hyperparameter tuning experiments, here is a quick demonstration of the non-linearity of the data. Figure 3 displays the data in a scatter plot with each input parameter as independent variables and the corresponding output as the dependent variable. The number of the input parameters correspond to the order in the .csv file as well as the numbers listed in Figure 1.

The data is shown to be non-linear by the following experiment. Linear regression of the data gives a variance score of 0.57, where a 1 represents perfect prediction. Additionally, the plot in figure 4 displays residual error, and the errors of train and test data are not easily separated out linearly using visual intuition. No linear boundary can be drawn that would provide any predictive value to this data as it is. Thus, the following methods for non-linear data are justified.

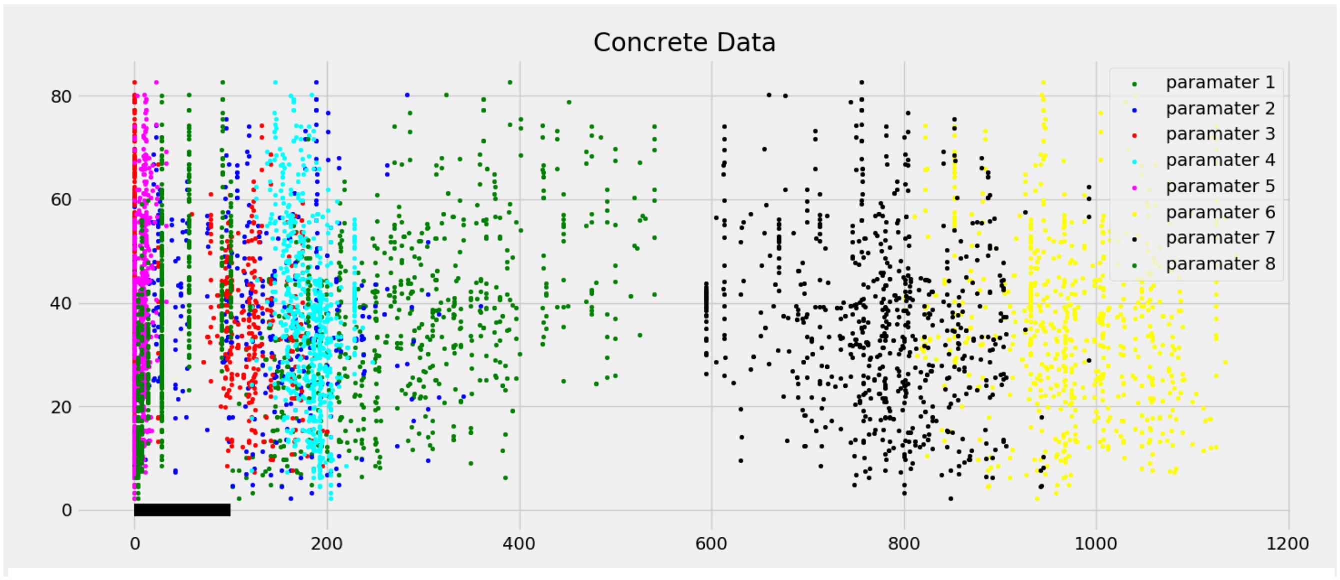

Figure 3: Distribution of Data.

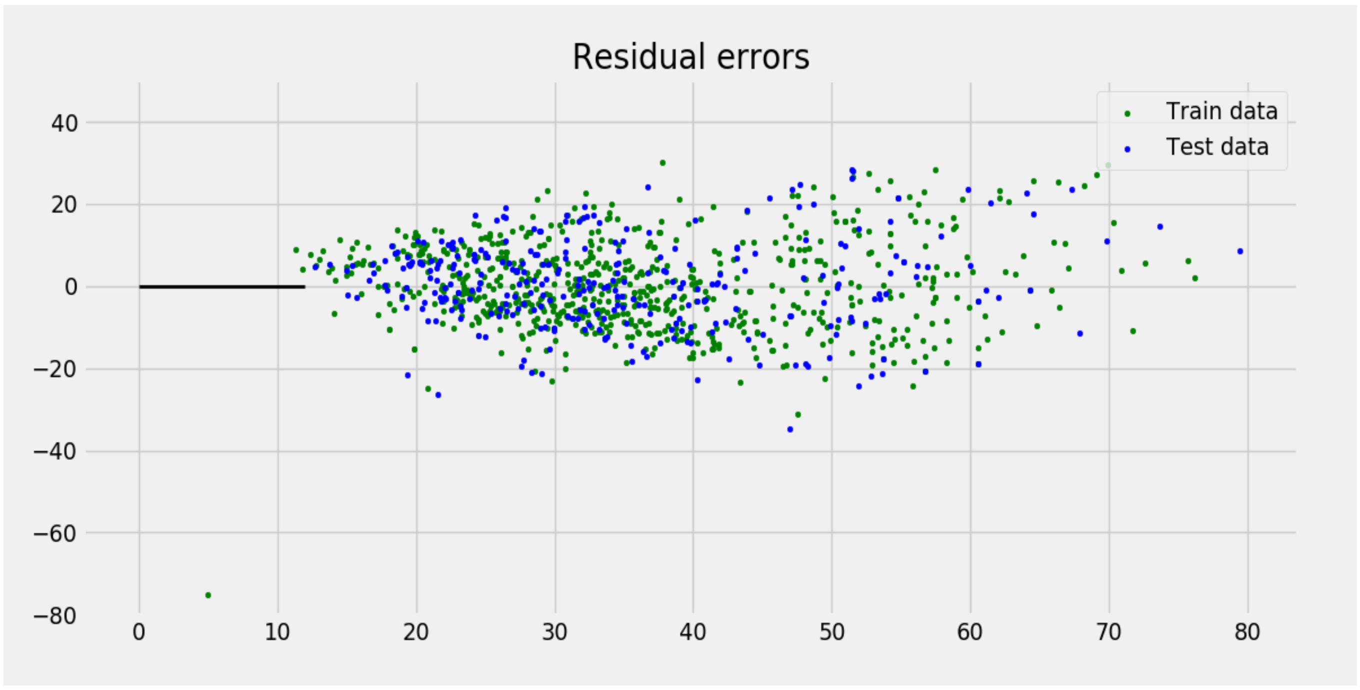

Figure 4: Residual Errors with Linear Regression.

4 Tuning Hyperparameters

4.1 Kernel Ridge Regression

The hyperparameters were tuned by running them in a series of nested loops that executed KRR algorithms for each combination of the parameters within a reasonable range. The definition of a "reasonable" range for all positive floats is certainly debatable, but for this project I used integer values 1-50 for alpha and gamma at first. This took 1760 seconds to run. The results of this indicated that the best MSE always occurs at the very beginning of this procedure, where both alpha and gamma equal 1. Specifically,

- alpha=1
- gamma=1
- kernel = 'poly'
- MSE = 41.0923932844002
- Confidence Interval upper bound = 37.12013925
- Confidence Interval lower bound = 32.75032446

So, the second iteration of this experiment, involved iterating float values between 0.0 and 1.0 (excluding 0.0 itself since positive values are required). This utilized NumPy's linspace function,[8] and the default step value used to increment alpha and gamma, which came out to checking roughly every 0.02 values. In this second test, the best hyperparameters were:

- alpha = 1e-05
- gamma = 1e-05
- kernel = 'laplacian'
- MSE = 21.521602305867255
- Confidence Interval upper bound = 37.46523276
- Confidence Interval lower bound 32.82982408

This was the final tuned hyperparameters used. Both iterations also cycled through all kernel options (except for linear kernel since instruction specifically asked us not to use this one), training KRR, recording the MSE and confidence intervals for each hyperparameter pairing result, and returning the hyperparameter combination that resulted in the lowest MSE.

Unfortunately, there were far too many hyperparameter combinations to visualize and graph each one's relation to the overall confidence intervals and MSE, but the code to reproduce my results in attached to this paper for any verification that might be necessary.

4.2 K-Neighbors Regression

The results of testing out different k values of the range 1-50 (inclusive) for KNN are displayed in figure 5. This experiment ran in 303.377 s. While there is no objective best k value that simultaneously locally minimizes train and test loss at a k value not equal to one, train loss comes to a natural plateau when k >= 5. Thus, the k value I decided to work with was k = 7, which locally minimized testing loss at 0.985. When k = 7, training loss was 0.851.

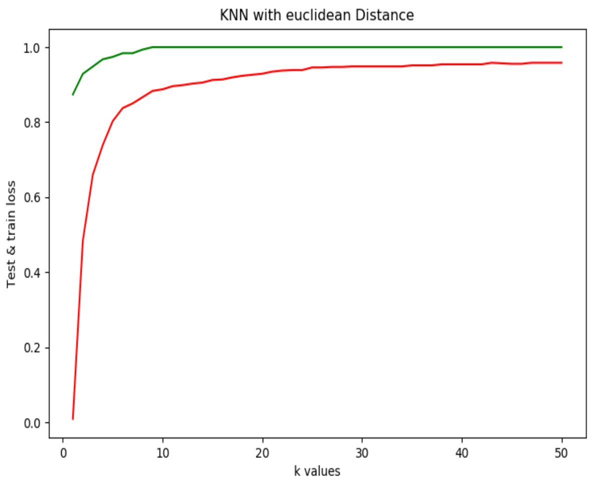

Figure 5: Test to observe the ideal k value. Green represents test loss and red is train loss.

5 Comparing Algorithm Performance

The heatmap of the correlation between input features of the concrete dataset is displayed in Figure 6. This is available for reference to help interpret results.

[8] NumPy documentation available at: https://numpy.org/

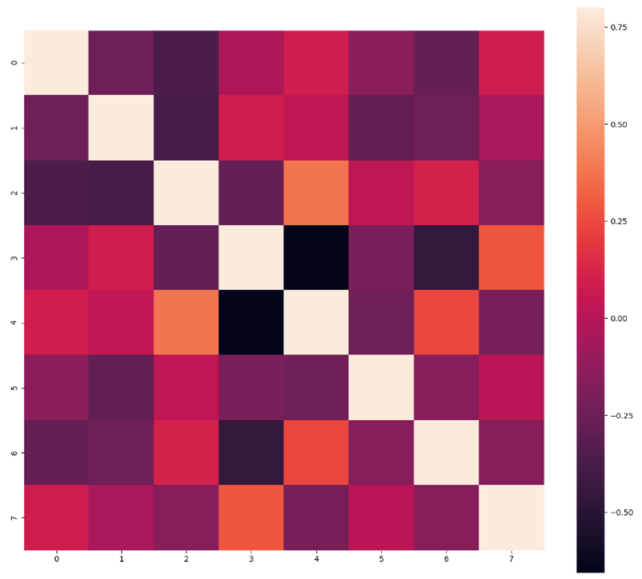

Figure 6: Input Correlation Heatmap. (Numbers of inputs correspond to the names listed in figure 1.)

The MSE was calculated for each regression model and is displayed in figure 7.

Regression Type	**MSE for Test Set**	**MSE for Validation Set**
Linear	122.18478	109.08736
KRR	21.52160	19.734383
KNN	102.79409	99.89190
NN	48.30109	30.20845

Figure 7: Mean Squared Error Comparison.

The confidence interval of the data with 95% confidence is a range of values that are 95% likely to contain the mean of the data. The confidence intervals for the predictions of each output are also listed in the following table. Smaller range of the upper and lower bounds indicate a higher probability that the reported MSE is correct.

Regression Type	**Upper Bound**	**Lower Bound**	**Range**
Linear	36.823836	32.688364	4.135472
KRR	37.465233	32.829824	4.635409
KNN	36.082495	32.185044	3.897451
NN	36.828537	31.563858	5.264680

Figure 8: Confidence Interval Comparison.

Taking all these results into consideration, here is what we observe. KRR returned the best MSE by far, for both test and validation sets. NN ran a second by this metric, followed by KNN. All three regression methods performed better than linear regression – a good sign (and sanity check) for non-linear data!

	Training Time (sec)
KRR	0.452
KNN	303.377
NN	27.61

Figure 9: Training time.

Figure 8 lists the confidence intervals by the upper bound, lower bound, and difference of these for each regression method. KNN return the most limited confidence interval range. Generally, with 95% confidence intervals, and smaller difference between upper and lower bounds indicates more “confidence” of predicting correctly. Overall, no regression models drastically underperformed by giving unexpectedly large confidence intervals.

For training time, as listed in figure 9, my implementation of KNN took the longest, and no others took that long. KNN’s hyperparameter tuning is the simplest, putting it at advantage compared to the other two regression methods. Even so, for a dataset of 1030 entries, comparatively small when faced with some massive datasets that are available, this still renders KNN much less practical on account of long fit time.

Taking all these factors into consideration, KRR performs best overall, with NN running next, and KNN third. When analyzing data from a similar source, especially relating to the compressive strength of concrete, Kernel Ridge Regression with a Laplacian kernel is my strong recommendation.

6 Conclusion

The compressive strength of concrete is a well-known nonlinear function of its age and components. This project implemented three supervised learning regression methods to comparatively analyze the performance. Taking into consideration many factors, Kernel Ridge Regression (with a non-linear kernel) demonstrates to be fast and the most accurate. While the hyperparameters of KRR take the most time to fine tune, the payoff is worth it is significantly more accurate results than the neural net and K-neighbors regression.

7 Acknowledgements

Any material not cited here is based on my own knowledge from machine learning, or based on class assignments including code provided for projects in classes at Georgia Tech, including CS 4641.

Dataset:

- I-Cheng Yeh, "Modeling of strength of high performance concrete using artificial neural networks," Cement and Concrete Research, Vol. 28, No. 12, pp. 1797-1808 (1998).

Code: Any code from the internet was modified to fit my specific project and data. Any code adapted from public repositories online is cited here as well as in the Python files as comments next to the function or class in which they were used where appropriate.

- https://github.com/scikit-learn/scikit-learn/blob/master/examples/plot_kernel_ridge_regression.py
- https://github.com/DHKLeung/MNIST_5-layer_Neural_Networks/blob/master/MNIST_5-layer_Neural_Networks.ipynb
- https://www.geeksforgeeks.org/linear-regression-python-implementation/
- https://towardsdatascience.com/deep-neural-networks-for-regression-problems-81321897ca33
- https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/kernel_ridge.py

Python Libraries: Some libraries not related to ML are left out because their use in this project was essentially implied (for example, simple math or graphing tools). These can be easily viewed in the imports list at the top of the Python files.

- NumPy: https://numpy.org/
- Pandas: https://pandas.pydata.org/docs/
- Sklearn: https://scikit-learn.org/stable/
- SciPy: https://www.scipy.org/docs.html
- Keras: https://keras.io/
- XGboost: https://xgboost.readthedocs.io/en/latest/python/python_intro.html
- Stats Models API: https://devdocs.io/statsmodels/