PREDICTING QUALITY OF RED WINE WITH KERNEL REGRESSION

RYAN MOUW

Applied Mathematics Department, University of Washington, Seattle, WA
rmouw@uw.edu

Abstract. The task was to create three models that will predict the batch quality of Portuguese wine on a 0-10 scale. Chemical measurements from the wine were used to train three models using linear regression, Gaussian kernel regression, and Laplacian kernel regression. The nonlinear structure of the data resulted in the Laplacian kernel model having the best predictive power (lowest mean squared error). Even with differing mean squared error, all three models predicted almost exactly the same score on the 0-10 scale, with the Gaussian predicting only a single wine differently compared to the linear and Laplacian models.

1. Introduction and Overview

In this problem we are given a dataset containing 11 different chemical measurements of Portuguese wine. The data originates from the University of California-Irvine Machine learning repository.[1] Along with the 11 measurements, each wine is given a quality score from 0-10. The task is to use the 11 chemical measurements to train linear regression, Gaussian kernel regression, and Laplacian kernel regression models that will predict the quality of the wine.

The training set provided has 1115 instances of wine, while the test set has 479. Additionally, a "new batch" of five wines is provided. This new batch does not include a wine quality score, but instead will be predicted using the three models created. Figure 1 shows the scatter plots of the 11 chemical measurements versus their corresponding wine rating. Note that the chemical measurements and wine ratings have all been normalized ($\mu = 0$ and $\sigma = 1$). This is an important part of the process and is explained in detail in section 3.

2. Theoretical Background

The primary exploration of this problem revolves around kernel regression. We use linear regression to contrast the kernel approaches, so I will *very* briefly explain linear regression. As the name suggests, linear regression is line fitting. In simple linear regression, there is an independent variable x and a dependent variable y, and the goal is to find an m and b such that equation 1 has the least squared error possible.

$$y = mx + b \tag{1}$$

The y value in equation 1 is the estimated dependent variable, given the independent variable x. In this problem, we will be using multiple linear regression, which just extends simple linear regression to include more than one independent variable.

The problem is that many phenomena are not well modeled using lines, but are rather non-linear or non-parametric. This is where kernel regression can help. Kernel regression allows for non-linear relationships in predictive modeling. Broadly speaking, a kernel is a function K, defined as [2]:

Date: February 26, 2022.

$$K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R} \tag{2}$$

Kernel regression allows us to estimate a prediction value $y^\star$ for any potential input value x^*. This is a form of interpolation. In kernel regression we predict the output $y^\star$, given as a function of $x^\star$, by summing the training data after it has been multiplied by a certain weight. The kernel itself is what decides these weights. Mercer's Theorem then connects a kernel, K, of a dataset to its features via the identity [3]:

$$K(x^\star, x) = \sum_{i=0}^{\infty} F_i(x^\star)F_i(x) \tag{3}$$

Where $F_i : \mathbb{R}^N \to \mathbb{R}$ are the features of K.

In our problem we used two different kernels, Gaussian (Equation 4) and Laplacian (Equation 5).

$$K(x^\star, x_i) = \exp \frac{-||(x_i - x^\star)||^2}{2\sigma^2} \tag{4}$$

$$K(x^\star, x_i) = \exp \frac{-||(x_i - x^\star)||_1}{2\sigma^2} \tag{5}$$

Where x_i is the i^{th} instance of training data. Looking at equation 4 and equation 5, we can see that the largest weight assigned to an input $x^\star$ will come from the training data (x_i) that is nearest it, and the least weight assigned will come from the training data that is farthest.

Just as in Ridge or Lasso regression, the goal is to minimize a loss function, given some penalization parameter. In the case of Kernel regression, because of the fact that we are interpolating, we are able to define the penalization parameter as:

$$\lambda||f(x^\star)||^2 = \lambda|| \sum_{i=0}^{N-1} a_i K(x^\star, x_i)||^2 \tag{6}$$

Where a_i is some real valued coefficient and λ is the hyperparameter that weights the penalization parameter.

Looking at Equations 4, 5, and 6 we can see that σ and λ are both hyperparameters in the Gaussian and Laplacian kernels. These hyperparameters modulate the "smoothness" of the kernels. In the cases of the Gaussian and Laplacian kernels, the higher the value of σ or λ, the smoother the function. To put it another way, increasing the value of either σ or λ, increases the amount of bias in the model. Decreasing either σ or λ will increase the amount of variance in the model. These hyperparameters have to be chosen carefully, as they will drastically change the way the model predicts.

3. Algorithm Implementation and Development

Kernel Regression is sensitive to the normalization, as the loss function considers the norm of the Kernel. The first thing that is done is to normalize the training, test, and new batch data sets using the standard deviation and mean of the columns of the training set. The normalized data is plotted in Figure 1 against the normalized quality ratings.

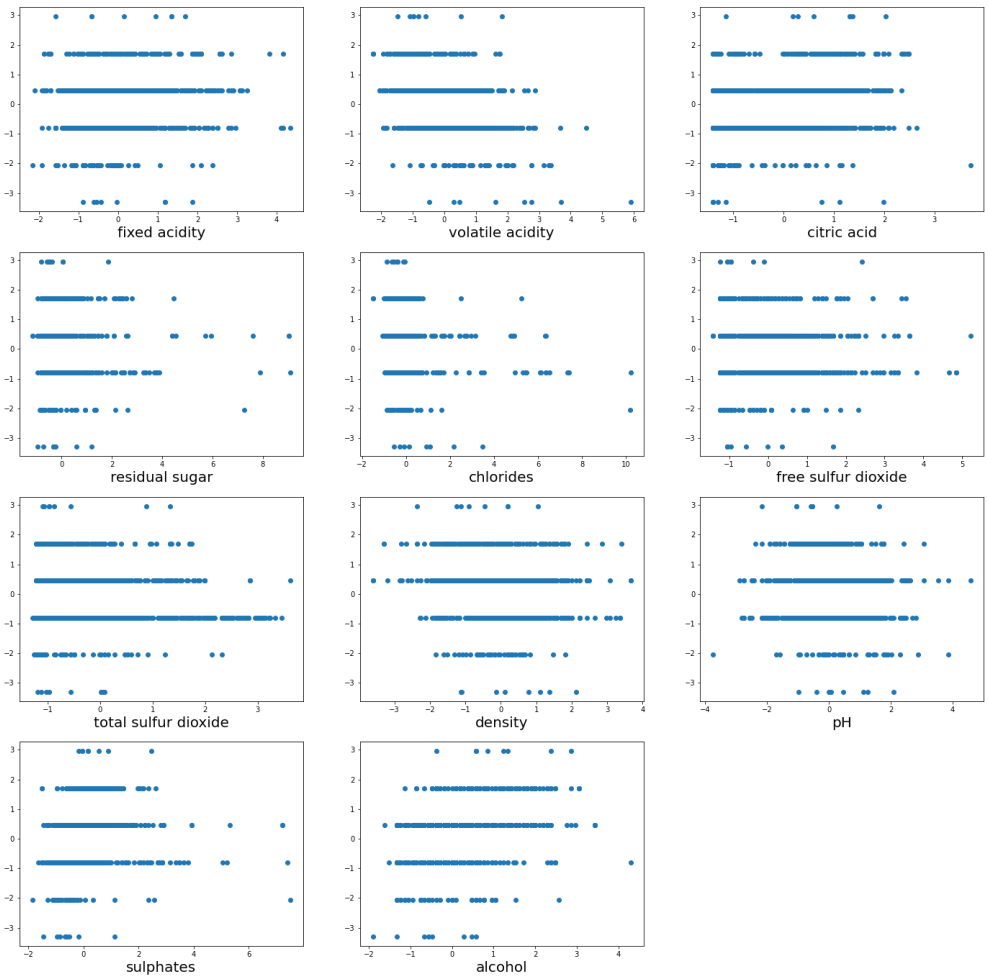

FIGURE 1. Normalized Data "Features" and Wine Quality Score

The LinearRegression() package from scikit-learn was used on the training set to fit the linear regression model. Both the Gaussian and Laplacian kernel regression models were trained on the training set using the KernelRidge() package in conjunction with the GridSearchCV() package. Both packages are also from scikit-learn. GridSearchCV was used to tune the hyperparameters λ and σ.[4] GridsearchCV uses the kernel that is provided (Either Gaussian or Laplacian in this case) and searches over a range of hyperparameter values, while incorporating cross-fold validation, and returns the highest scoring hyperparameters in the range. [4]

Cross Validation is a way of ensuring that the training and testing set split of your data does not bias the model. Instead of having a single test set and a single train set, in cross validation, the dataset is divided into n equally sized pieces. In our case, we divided the dataset by 10. Each "fold" takes a turn acting as the test set, while the other $n-1$ sets are used for training. The learned parameters are then averaged over the n folds. Finally, the original test set is now referred to as the "validation set", and is still used to estimate the model's future performance. This process reduces the variance within the model. Along with 10-fold cross validation, a range of values for the hyperparameters (σ and λ) also needed to be explored to determine which values resulted in the best "scoring" model.

The scoring metric used for this problem was mean squared error. The initial values for σ and λ ranged from 2^{-5} to 2^5. The range was then repeatedly reduced until the it was small enough

to contain the parameters on repeated trials of 10-fold cross validation. One thing to note at this point is that the best performing values for σ and λ will change with each trial of 10-fold validation, as the 10 folds are chosen at random each time the model is fit. This means that the training sets will look slightly different each time the cross validation is implemented. By reducing the range of σ and λ as much as possible while still containing repeated iterations of 10-fold validation, we are allowing for a choice of hyperparameters that recognizes the randomness of the training set. Once the smallest range of hyperparameters was found, I then chose the center value of the range to be the hyperparameters of the model, as this location seemed to have the lowest standard deviation of the cross-validation mean squared error. This is an indication these particular hyperparameters were returning similar results on each training set, which is an important indication of an appropriate level of bias in the model.

4. Computational Results

The hyperparameter values that were chosen using the process described in Section 3 are shown in Table 1.

Kernel	σ	λ
Gaussian	3.9908	0.1515
Laplacian	1.4251	0.2227

Table 1. Best σ and λ values found during 10-fold cross validation

The Linear regression, Gaussian Kernel Regression, and Laplacian Kernel Regression mean squared error (MSE) of the normalized data and labels are in Table 2. Note that these values are from the normalized data. The Laplacian kernel regression model had the best performance, with a test MSE of .6067, and a relatively low training MSE of .0511.

Kernel	Train MSE	Test MSE
Linear	0.6278	0.7472
Gaussian	0.4583	0.6835
Laplacian	0.0511	0.6067

Table 2. MSE for various kernels

The Laplacian kernel was followed by the Gaussian kernel and then finally the linear regression model. The very low Laplacian training MSE indicates that GridSearchCV function was able to locate a local minimum MSE score in the 2D hyperparameter space. This can be seen below in Figure 2, which is a contour plot of a range values of σ and λ with their associated MSE on the left plot, and in the right plot, the standard deviation of the changing MSE amongst the 10-fold cross validation performed at each pair of hyperparameters.

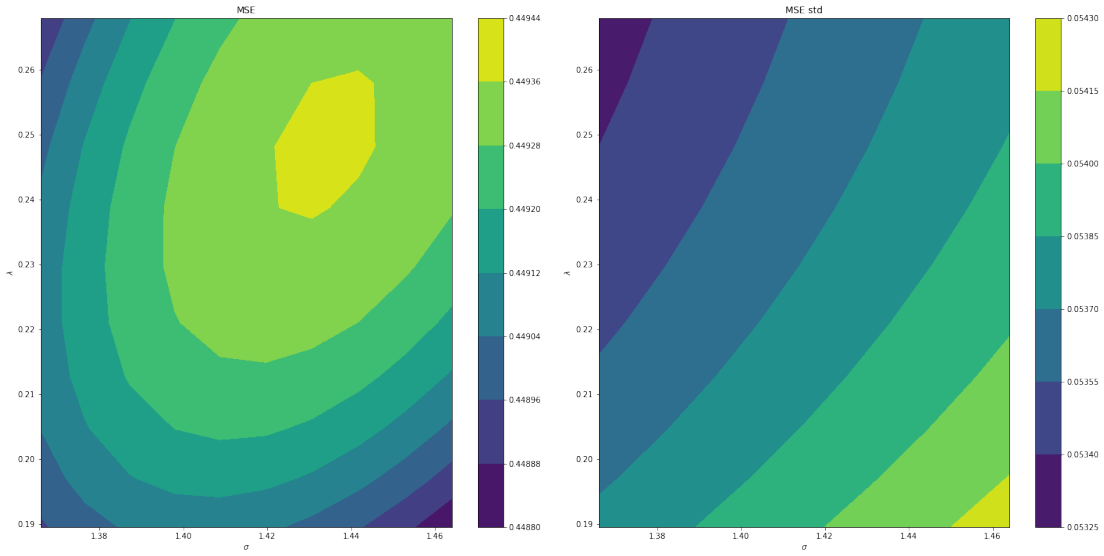

FIGURE 2. Laplacian Hyper Parameter MSE and MSE Standard Deviation

The same contour plot is shown for the Gaussian kernel regression hyperparameters in Figure 3. Contrasted with Figure 2, Figure 3 has a sort of ridge of values, with a less clear center.

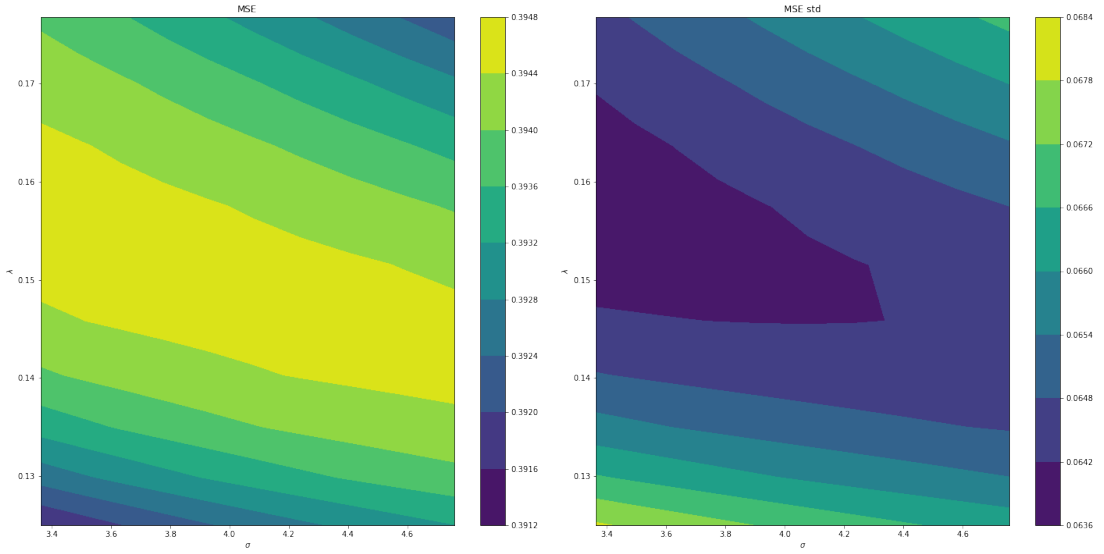

FIGURE 3. Gaussian Hyper Parameter MSE and MSE Standard Deviation

Using the three different models, a prediction was made on the quality rating of a new batch of 5 wines. The results of the prediction are shown in Table 3. Note that the prediction values have been returned to their original scale, and rounded to the nearest integer, matching the 0-10 scale given in the dataset. Interestingly, the model with the lowest and highest MSE make identical predictions (Laplacian and linear, respectively), while middle performing model (Gaussian) varies in its prediction of wine #3.

Wine	Linear	RBF	Laplacian
1	6	6	6
2	5	5	5
3	6	5	6
4	6	6	6
5	6	6	6

TABLE 3. Wine quality predictions on the new batch.

5. Summary and Conclusions

All three models had a relatively high testing MSE. The Laplacian kernel had a remarkably low training MSE compared to the other two models, which could explain the lower testing MSE. Figure 2 and Figure 3 would be interesting to explore further, especially the ridge-like area of better performing hyperparameters in Figure 3. The resulting predictions were also noteworthy. The best scoring and worst scoring models made the same predictions, while the middle scoring model differed. This could be a result of the randomness involved in 10-fold cross validation compounded with the rounding that was necessary to place the wines in the 0-10 scale.

References

[1] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553, 1998.
[2] B. Hosseini. Introduction to kernel methods, Feb 2022.
[3] B. Hosseini. Kernel ridge regression, Feb 2022.
[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.