AMATH 582: ASSIGNMENT 3

JON STAGGS

Applied Mathematics M.S. Student, University of Washington, Seattle, WA
jrstaggs@uw.edu

Abstract. We train linear and kernel regression models on data containing information on the properties of different wine batches along with their overall quality score. Using our regression models we compare their predictive accuracy of quality score on a test set then use these models to predict the quality of a new batch of wine.

1. Introduction and Overview

We utilize Kernel Ridge Regression methods to train two models, corresponding to different Kernels, to predict the quality of a new batch of wine. We use Cross Validation to tune our model by finding hyperparameters to minimize the training and testing errors. We will provide an overview of how Kernels are used to form a Ridge Regression problem which can work on nonlinear data. We discuss the relevant Python libraries and algorithm used to carry out the training and testing of our model. Finally, we present how we found our tuned hyperparameters, the errors of each model, and the use these models to predict the quality of a new batch of wine.

2. Theoretical Background

We will build up the necessary definitions and notions of Kernels and show the equivalent formulation of Ridge Regression in the Kernel framework. This allows us to train and predict more nonlinear problem by taking our data into a Kernel Hilbert Space where we can use regression to build a model. We also provide an overview of the Cross Validation process.

2.1. Kernels and Reproducing Kernel Hilbert Spaces.

For the background of Kernels we follow along with [2]. We define a *Kernel* to be a function $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$. We say that a Kernel is symmetric if

$$K(x, x') = K(x', x), \ \ \forall x, x' \in \mathbb{R}^n. \tag{1}$$

Here we restrict ourselves to Kernel's whose matrix associated associated with the Kernel K, $(K)_{i,j} = K(x_i, x_j)$ is non-negative definite ($x^T K x \geq 0$) and symmetric (NDS) [2], [7]. We will make extensive use of Mercer's Theorem which states,

Theorem 1. *If K is NDS and integrable then there exists an orthonormal basis $\{\psi_j\}_{j=0}^{\infty} \in L^2(\mathbb{R}^n)$ such that*

$$K(x, x') = \sum_{j=0}^{\infty} \psi_j(x)\psi_j(x') \tag{2}$$

where the λ_j are eigenvalues of K and $\psi_j(x)$ are eigenfunctions.

Date: February 25, 2022.

Mercer's Theorem provides us with a generalization of an eigendecomposition of NDS matrices. [2]. We can define a function which is a scaling of our eigendecomposition, $F = (F_0(x), F_1(x), ...) \in \ell^2(\mathbb{R})$ where $F_j(x) = \sqrt{\lambda_j}\psi_j(x)$. Mercer's Theorem gives us that $K(x, x') = \langle F(x), F(x')\rangle$.

We now define a *Reproducing Kernel Hilbert Space* (RKHS). The definition of a generic Hilbert space is complete normed vector space that is - a vector space with an inner product that induces a norm to form a complete metric space (in a Cauchy complete sense). Given an NDS Kernel it's RKHS, $\mathcal{H}_k$, is the space of functions $f : \mathbb{R}^n \to \mathbb{R}^n$ that have the form,

$$f(x) = \sum_{j=0}^{\infty} c_j F_j(x) \text{ such that } \sum_{j=0}^{\infty} c_j^2 < 0. \tag{3}$$

The RKHS of a Kernel is the set of functions that can be represented as a linear combination of the scaled eigenfunctions, $F_j(x) = \sqrt{\lambda_j}\psi_j(x)$. We can see from two views what functions live in the RKHS: 1) F_j determines the functions since F_j must be able to build up these functions, 2) we select a Kernel and that Kernel will yield some features, F_j (think about selecting a matrix and that gives us eigenvalues/vectors). Mercer's Theorem tells us that these views are equivalent. We now state the famous Reproducing property a.k.a the Kernel trick [2]. Suppose we select a Kernel, K, and consider $K(x, x')$ for fixed x' and take some $f \in \mathcal{H}_k$ we have that,

$$f(x') = \langle f(\cdot), K(x', \cdot)\rangle. \tag{4}$$

In other words, the point value of a function in your RKHS can be found by taking the inner product of that function with the Kernel.

2.2. **Kernel Ridge Regression.** We would like to use the properties above to show that the original Ridge Regression problem is equivalent to the Kernel Ridge Regression Problem. The Ridge Regression problem seeks a solution to

$$\min_\beta ||A\beta - Y|| + \lambda||\beta||^2, \text{ A is feature matrix, Y is label/output matrix.} \tag{5}$$

Here the elements, $A_{i,j} = F_j(x_i)$, that is the jth feature map of the ith element. We know that there is a Kernel from which these features are coming from and vice versa that we can build these features by selecting a certain Kernel. To get to our Kernel ridge, we want a function

$$f(x) = \sum_{j=0}^{\infty} \beta_j F_j(x) \approx y(x). \tag{6}$$

The power of the Kernel trick is that it allows us to represent f equivalently as,

$$f(x) = \sum_{j=0}^{\infty} a_j K(x_j, x). \tag{7}$$

We now use a theorem provided in [2] which states that if we want to minimize eqn. (7) then we can actually minimize a finite sum instead. With these in hand we can use the Representor Theorem which states:

Theorem 2. *A function $\hat{f} \in \mathcal{H}_k$ is a minimizer of*

$$min_{f\in\mathcal{H}_k}||f(X) - Y||^2 + \lambda||f||^2_{\mathcal{H}_k} \tag{8}$$

if and only if $\hat{f} = \sum_{n=0}^{N-1} \hat{a}_n K(x_n, x)$ with $\hat{a} = (a_0, ..., a_{N-1}) \in \mathbb{R}^N$ being a minimzer of

$$min_{\alpha\in\mathbb{R}^n}||\Theta_a - Y||^2 + \lambda a^T \Theta a \tag{9}$$

where $\Theta_{i,j} = K(x_i, x_j)$.

This means that if we solve the Θ Ridge problem we get our original Ridge problem solution for free. Moreover, the fact that we are able to do so with finitely many terms of a linear combination means that we have found a feature space where we can find the solution to be a hyperplane.

The punchline is that the Kernel trick allows us to write f as a sum of weights and the Kernel (being evaluated at some point) and this lets us use the Representor Theorem to instead solve the Ridge problem for our Θ matrix which gives us the a_j weights to represent f in our original feature space and as mentioned above this yields a hyperplane in our feature space.

2.3. **Cross Validations.** The idea behind Cross Validation is that we want tune our model as well as the Ridge Regression parameter λ. If we have a large training set, but maybe not a high quality or large test set then this process can be difficult by simply guessing and checking parameters. We use the notation from [2]. Take an N-fold Cross Validation will first randomly permute our training data then chop it up into N pieces - call each of these pieces $\tilde{X}_i, \tilde{Y}_i$. In this view we are creating a test and training sets where we solve our ridge regression problem, for a particular parameter value, on $N-1$ of the permuted subsets then evaluate/test on the remaining permuted subset. We obtain N models that are fit and we define the cross validation score/cost by computing the average of some metric across our new subsets [2]. We use the notation of subscript $-k$ to mean all the subsets except the kth subset. Our cost is then defined as,

$$\mathrm{CV}(\hat{f}, \lambda, \sigma) = \frac{1}{N}\sum_{k=0}^{N} ||\hat{f}_{-k}(\tilde{X}_{-k}, \lambda_{-k}, \sigma_{-k}) - \tilde{Y}_{-k}||^2. \tag{10}$$

The score is the average of the norm squared of the prediction and true value where each term in the sum is the norm square of all but one of the permuted subsets. We then do this for each value of our parameter. In this application we use an additional parameter value that is associated with the standard deviation of our chosen Kernel. Our goal is to pick the λ and σ that minimize the cross validation loss.

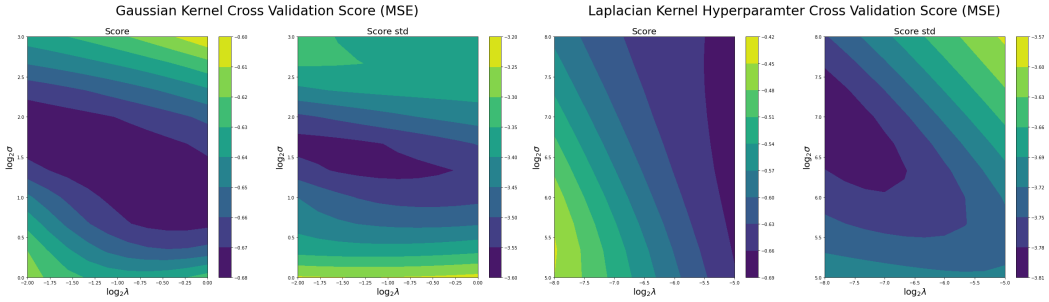
3. Algorithm Implementation and Development

We use Python version 3.8 [8] with SciKit-learn [4], NumPy [1], and MatplotLib [3] libraries and Sypder as our IDE [5]. We will first describe the main SciKit-learn functions that we use then discuss the implementation for obtaining the Kernel Regressions.

3.1. **Scikit Learn Functions.** The two main classes that we utilize is SciKit-Learn's *kernel_ridge* class. From this class we use the *KernelRidge* method with our training input and labels which is solving the minimization problem in (9). We make note of the convention in which Scikit learn assigns the parameters for the Gaussian and Laplacian kernel- these take a standard deviation of $\gamma = 1/(2\sigma^2)$ and $\gamma = 1/\sigma$ for the Gaussian and Laplacian case respectively. Moreover, when we plot and compute we use the convention form [2] and raise our parameters to the power of 2 - ie $\alpha = 1/(2^\lambda)$, $\gamma = 1/(2(2^\sigma)^2)$ and similar for the Laplacian case. This simply acts as scaling of our variables. We also use *KernelRidge*'s *fit* and *predict* methods. The *fit* method which performs the Kernel Ridge Regression on the data provided and the *predict* method which uses the fitted model to compute the labels of the given input.

We use the *model_selection* class and the *cross_val_score* method for the cross validation process mentioned in Sec. 2.3. The arguments are the Kernel estimator, input data, labels/output data, the scoring metric (e.g. negative mean squared), and the n-fold cross validation desired.

3.2. **Implementation.** In [6] we discussed the implementation for training and testing a linear regression so here we only cover the implementation of the Kernel Regression. We have three major steps with the general outline of the code from lecture 16 of [2]: 1) Using cross validation to find appropriate hyperparameters σ and λ for each kernel, and 2) fitting



(A) Gaussian parameter Cross Validation (B) Laplacian parameter Cross Validation

FIGURE 1. Cross Validation Scores for Kernel hyperparameters λ,σ

the model, 3) predicting with the model. Our input data contains 11 points for each feature and 1 label which is the quality score. Since Kernel methods rely on things like a Gaussian or Laplacian distribution they are very sensitive to the mean and standard deviations so it is common practice to center your data at mean zero and scale to unit variance - we do that in this case. Now we proceed to the cross validation process.

First we create a "default" kernel regressor by providing just the desired kernel which have the default parameters of X Y. We establish a grid of desired length and mesh size which correspond to the space of parameter. We then do a double for loop in the outer loop and assign a sigma value corresponding to that grid row. In the inner loop we assign a λ value corresponding to a point in that row, we input these parameters into our regressor, and finally we use the *cross_val_score* method to compute the cross validation scores. For each inner loop we take the mean and standard deviation of the 10 fold scores and record that value to plot later (see Fig. 1).

To fit our model we simply use the *fit* method from the *KernelRidge* method. We then use *KernelRidge*'s *predict* to estimate our values in the feature space. For each we compute the mean square error between the predicted labels and the actual labels. Note that we convert our predicted and actual labels back to their original scale when computing the errors.

For the prediction of the new batch we simply provide the new batch input data to the model using the *predict* method. We then converted back to the original scale using the training data mean and standard deviation.

3.3. **Parameter Tuning Execution Time.** In our parameter tuning we initially tested out a grid domain of length twenty (in $\log_2$ scale) with twenty grid points and ten fold cross validation to find a local minimum region. The execution time was on average about 160 seconds. Once we reduced our grid points to ten with ten fold cross validation the execution time was about 40 seconds. Given the relatively short time we used the larger grid to search for local minimum then refined the length and used ten grid points.

4. Computational Results

4.1. **Parameter Tuning.** As referenced in Sec. 3.3 we initially started with a grid length of 20 in logarithmic scale and 20 grid points with the cross validation score is calculated with the negative mean squared error. Upon locating a minimum region we refined down the grid length to locate more precise value where grid lengths were reduced to at most 3. In Fig. 1a, we have a cleaner overlap with the Gaussian kernel between the score and score standard deviation so we settled on the values $\sigma = 1.5$ and $\lambda = -1$. For the Laplacian Kernel, Fig. 1b, we don't have a clean overlap in the λ variable. Using the eyeball norm we chose values that split the difference and minimized the score and score standard deviation as much as possible. We identified $\sigma = 6.5$ and $\lambda = -6$ for our Laplacian Kernel hyperparameters.

Data Set MSE	Linear Regression	Gaussian Kernel	Laplacian Kernel
Training MSE	.5135	.4829	.5095
Testing MSE	.5816	.7170	.5313

TABLE 1. Mean Squared Error for the testing and training data set predictions

Wine	Linear Regression	Gaussian Kernel	Laplacian Kernel
Wine 1	5	5	5
Wine 2	6	6	6
Wine 3	6	5	6
Wine 4	6	5	6

TABLE 2. Quality Score predictions from each method for each wine from the new batch

4.2. **Training and Testing Errors.** To calculate the mean squared error we converted back to the original scale of 1 to 10 using the training mean and standard deviation to convert the predicted values and rounded the predicted values to the nearest integer value. From Table 1, we see that our best testing error comes from the Laplacian Kernel. The author found it surprising that the basic linear regression performed so well, namely, better than the Gaussian Kernel. We will see in Sec. 4.3 the predicted scores deviate the most from the other two methods. Using these methods to predict quality of new batches we would use the Laplacian kernel.

4.3. **Predicting New Batch Quality.** In Table 2, we converted back to the original scale using the testing mean and standard deviation then rounded to the nearest integer. Interesting to note that the Laplacian and Linear Regression agreed completely on these predictions. The three methods only all agreed on Wine # 2. The rounded mean of the testing set was 6 (true mean was 5.63) so using the Laplacian Kernel we could conclude that the quality of these wines are just as good in quality as the previous batches.

In Fig. 2, we present the different input variable for the testing and training set along with their corresponding quality. We also plot the new batches input variable with the predicted quality from the Gaussian and Laplacian Kernel methods. One thing to note is the tendency to center these scores around the mean. We conjecture that this is due to the choice of Kernel - being strongly centered about the mean. This might be lessened if larger standard deviations from the Kernel were chosen at the risk of a loss of accuracy. This might also simply be explained that it's likely the new batch of wine is average quality (relative to the training data).

5. Summary and Conclusions

Using Kernel Regression methods and finding tuned hyperparamters for the ridge regression and Kernel we have predicted the quality of a new batch of wine. We found that the most accurate model was a Kernel Ridge Regression with the Laplacian Kernel. With this, we determined the quality of the new batch of wine to be about average quality as the previous batches.

Future considerations to explore are better methods for finding more optimal hyperparamters. The author would also like to explore using other Kernels as well as given a feature map how one can construct a Kernel. Moreover, if the work required to construct a Kernel from the feature yields significantly better results.

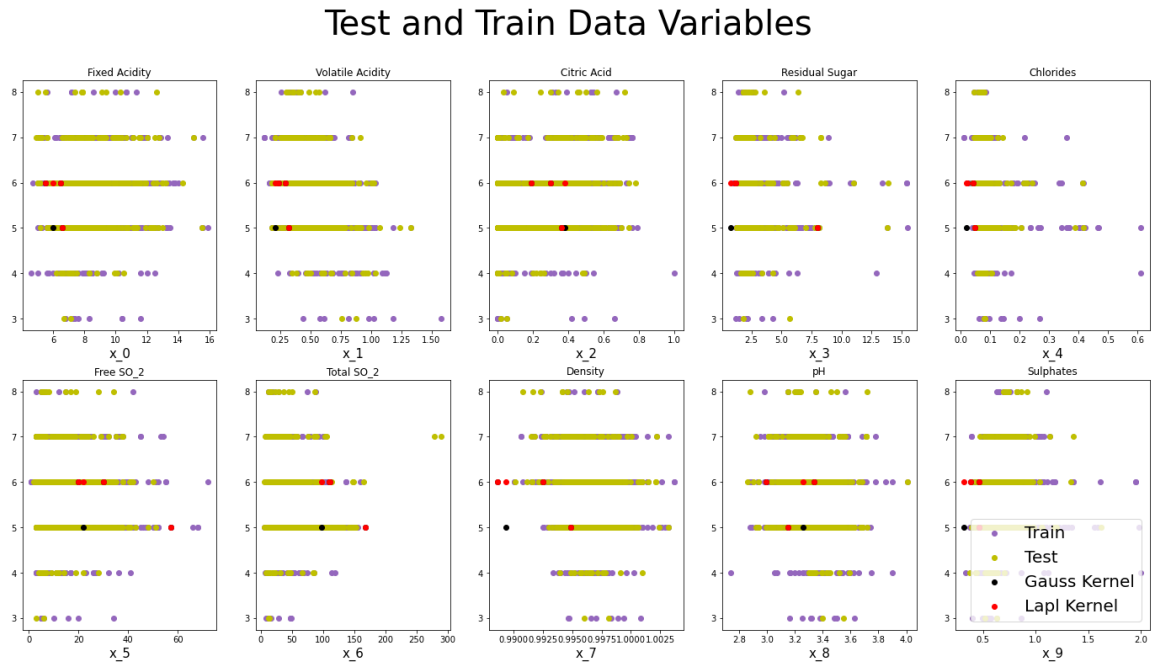


FIGURE 2. Properties of the wine and their corresponding score for each variable for the Training, Testing, and Predicted scores from the Kernel methods.

ACKNOWLEDGEMENTS

The author thanks fellow classmate Daniel Leon for generally helpful discussions about the theoretical aspects of the Kernel trick and for advice on plotting.

REFERENCES

[1] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020.

[2] B. Hosseini. Amath 582 course notes. Notes for Computational Methods for Data Analysis.

[3] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[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.

[5] P. Raybaut. Spyder-documentation. *Available online at: spyder-ide.org*, 2009.

[6] J. Staggs. Amath 582 assignment 2. Homework assignment 2 for amath 582.

[7] L. N. Trefethen and D. Bau III. *Numerical linear algebra*, volume 50. Siam, 1997.

[8] G. Van Rossum and F. L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.