QUALIFYING RED WINE

ABHISHEK SHARMA

Mechanical Engineering Department, University of Washington, Seattle, WA
as711@uw.edu

Abstract. In this work, we develop an algorithm that predicts the quality of red wines from a series of chemical measurements. We train two predictors using Kernel Ridge Regression with Gaussian and Laplacian kernels and compare them to a baseline linear regressor.

1. Introduction and Overview

In this work, we model the wine preferences for the red variant of the Portuguese "Vinho Verde" wine. We have a dataset comprising 1115 training and and 479 test instances of wine with their labelled quality ratings. The dataset is sampled from [1] The input features for each wine sample are the following chemical properties of the wine- 1) fixed acidity, 2) volatile acidity, 3) citric acid, 4) residual sugar, 5) chlorides, 6) free sulfur dioxide, 7) total sulfur dioxide, 8) density, 9) pH, 10) sulphates, 11) alcohol. The output score is a scalar value between 0-10. We use regression to predict the wine quality score from the input features. Three methods are used for regression: Linear regression (Baseline), and Kernel Ridge regression using two kernels Gaussian kernel, and Laplacian Kernel. We compare the performance of the three methods on training and test sets using. We report the mean squared error (MSE) as the error metric for train and test sets. Then, we predict the scores for a new unlabelled batch comprised of 5 wine samples.

2. Theoretical Background

Solving Machine Learning problems requires transforming data into appropriate representations of the data. If the right features are selected, complex decision boundaries can be made simple or linear, for example the classic example of drawing decision boundary between data in two concentric rings in a 2D plane is hard to solve with a linear boundary but can be easily linearly separable if the data is projected into a 3 dimensional space like in the Support Vector Machines (SVM) [2]. Projecting into higher dimensional space using feature maps (F) can make the decision boundaries simpler but also make it computationally expensive to store a long list of features. *Kernel trick* is a way to mitigate that. According to Mercer's Theorem [3], each set of features, $F(\underline{x}) := (F_0(\underline{x}), F_1(\underline{x}), F_2(\underline{x}), ...)$ have a corresponding Kernel $K : R^n \times R^n \to R$ associated with them (See eq. 1).

$$K(\underline{x}, \underline{x}') = \sum_{j=0}^{\infty} F_j(\underline{x}) F_j(\underline{x}') \tag{1}$$

where $F_j : R^n \to R$, and (x and $x' \in R^n$) are distinct points in the training data.

Date: February 26, 2022.

Thus, a Kernel can represent an infinite set of features (example the Gaussian Kernel [4]), with only a finite number of pairwise computations between the points (x, x'). This can be a powerful method for small to medium dataset, provided an appropriate kernel is chosen.

Usually the end goal in machine learning it to build models the predict outcomes, dependent variables or classify data, This involves learning a mapping from the features in the data to the desired output, which is accomplished by training a classifier or a regressor on a part of data called *Training set*. Then, the performance of the learned model is estimated on a *Test set* which is an approximation to how the model with perform in the real world setting.

The simplest regression method is *linear regression* [5] which fits a linear model to the data i.e. the outputs are the weighted sum of the input features. The weights are the coefficients ($\hat{\beta} \in R^k$) of the linear model learned by minimizing the error between the predictions and the desired label, as shown in eq 2.

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} ||A\beta - y||^2 \tag{2}$$

where $A \in R^{m \times k}$ is the input matrix with m training examples and n features, $y \in R^m$ are the associated ground truth.

However, this method can be inadequate for learning complex non-linear relationships between input and output variable. Learning non-linear functions requires engineering appropriate features from the input data, as we described above. Then, a linear relationship can be learned between constructed features and the desired output. *Kernal Ridge Regression* [6] is a method to do so using the *Kernel Trick* (see above). *Kernal Ridge Regression* involves solving the optimization problem in eq 3.

$$\hat{\underline{a}} = \underset{\underline{a}}{\operatorname{argmin}} ||\Theta\underline{a} - y||^2 + \lambda \underline{a}^T \Theta \underline{a} \tag{3}$$

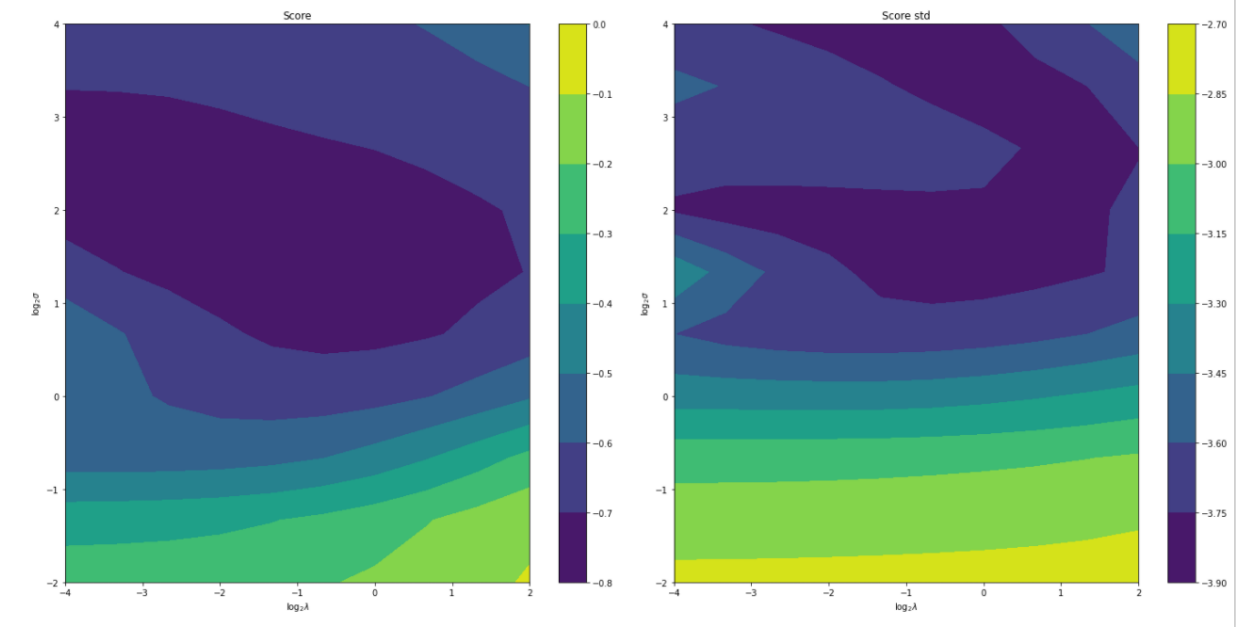
where $\Theta \in R^{m \times m}$ with m training samples and $\Theta_{ij} = K(\underline{x}_i, \underline{x}_j)$ and λ is the regularization term. Then the learned function that approximates $y = f(x)$ is given by $\hat{f}(x) = \sum_{j=1}^{m} \hat{a}_j K(\underline{x}_j, \underline{x})$ (Representer Theorem). The hyperparameters for Kernel Ridge Regression are the regularization term λ and the parameter σ associated with the chosen Kernel. For example, for the Gaussian kernel $K_{rbf}(x, x') = exp(-\frac{||x-x'||_2^2}{2\sigma^2})$ and Laplacian Kernel $K_{lap}(x, x') = exp(-\frac{||x-x'||_1}{\sigma})$, the length scale σ is a hyperparameter. λ and σ are tuned using cross validation [7] to find an optimal model with low bias and low variance. (See Bias-variance tradeoff [8]) The optimal model achieves the best generalization performance. The commonly used performance metric for regression problems is mean squared error (MSE). The MSE can be computed as shown in eq. 4

$$MSE = \frac{1}{m} \times ||\hat{f}(x) - y||_2^2 \tag{4}$$

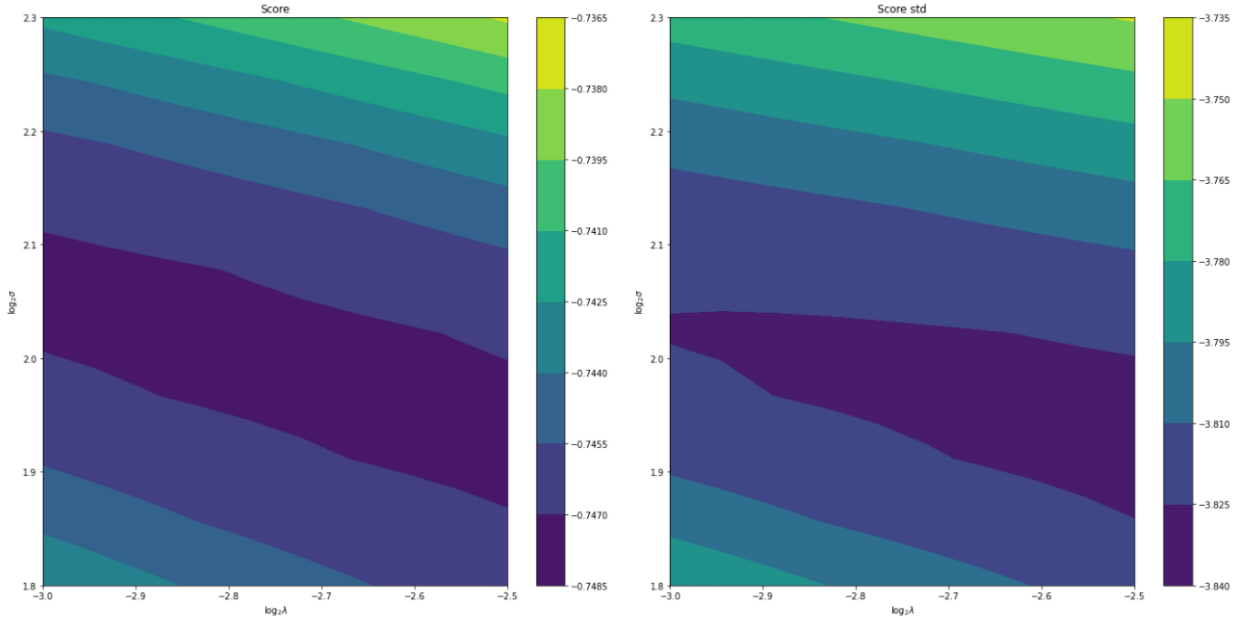
where $x \in R^{m \times n}$, m is the number of samples, n is the dimensions of input, $\hat{f} \in R^m$ are the predicted values, $y \in R^m$ is the ground truth.

3. Algorithm Implementation and Development

All the algorithms were implemented in Python [9]. Numpy package [10] was used for data array manipulations, sklearn [11] was used for Linear Regression and Kernal Ridge regression. All the visualizations and figures were generated using matplotlib [12]. We used grid search over different scales for multiple values of the hyperparameters (σ and λ). For each combination of hyperparameter values (σ and λ), we performed *10-fold cross validation* using the *model_selection.cross_val_score* method from scikit-learn. Regularization parameters (λ) were selected on a logarithmic



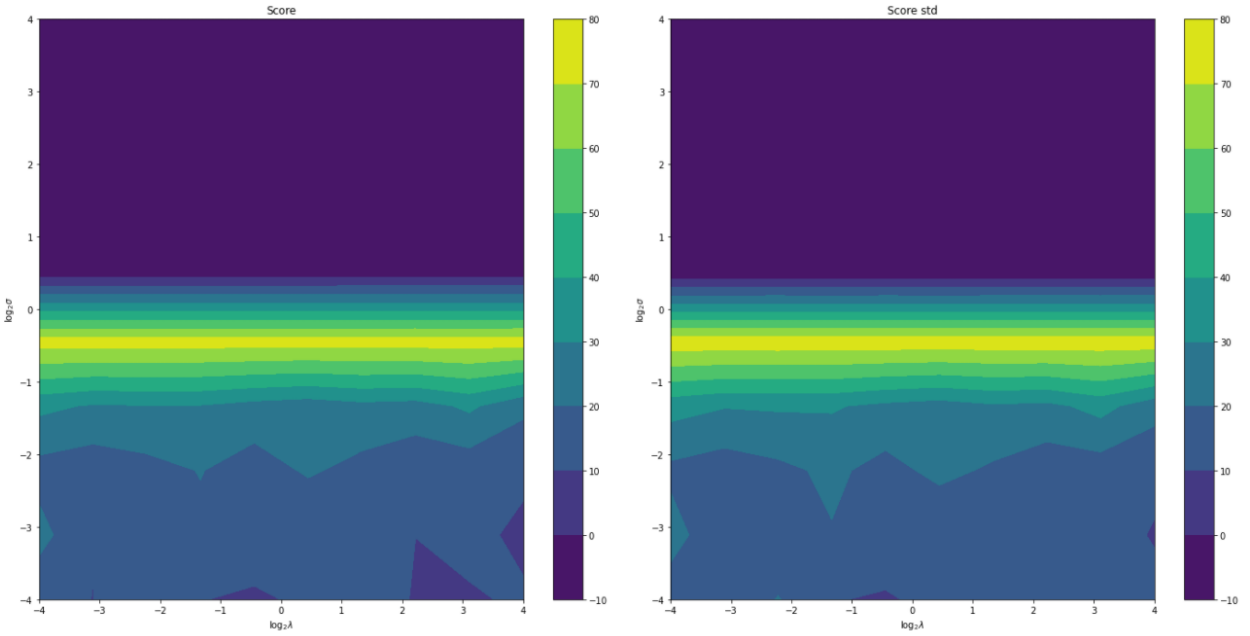
(A) Iteration 1



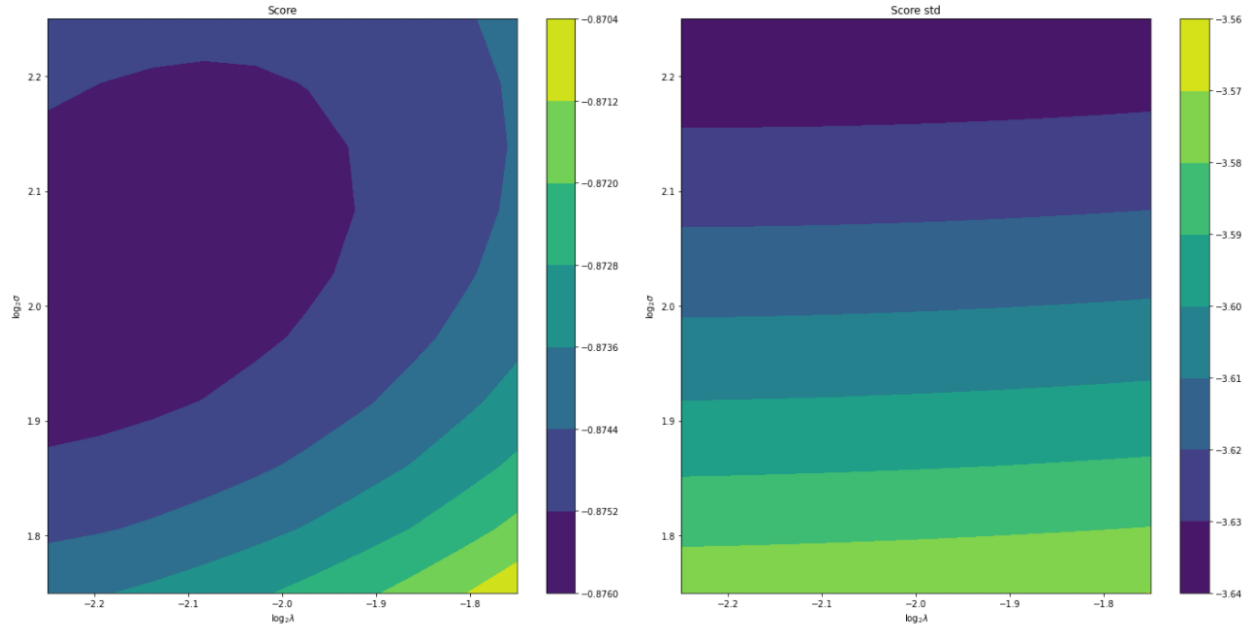
(B) Iteration 4

FIGURE 1. Contour plots for First and Fourth iteration of hyperparameter search for the Gaussian Kernel by successively refining the search grid.

scale with base 2. Initially a course grid was selected the values of MSE were plotted as a function of σ and λ. After visual inspection a finer grid was chosen in the next iteration. And the process was repeated until the improvements in performance became small. The contour plots for the hyperparameter optimization process are shown in Fig. 1 for the Gaussian Kernel and in Fig. 2 for Laplacian Kernel. All iterations are not shown due to space limitations.



(A) Iteration 1



(B) Iteration 4

FIGURE 2. Contour plots for First and Fourth iteration of hyperparameter search for the Laplacian Kernel by successively refining the search grid.

4. Computational Results

The performance of the 3 regression Models on our dataset are reported in the Table 1. We find that Linear Regression performs the worse of all the models. Both Gaussian and Laplacian Kernels perform better than the Linear model with the Laplacian Kernel has the best performance among the three methods on the test set. We also find that difference between the performance on training

set and Test set is much larger for the Laplacian Kernel. Thus, it seems to overfit to the training data indicating a greater model complexity. With greater model complexity its performance could improve further with more training data. With lesser training data it would tend to overfit and Gaussian Kernel would be a preferable option.

	Linear	Gaussian	Laplacian
Train MSE	0.6278	0.4564	0.0504
Test MSE	0.7471	0.6816	0.6068
optimal σ	-	$2^{1.96}$	$2^{2.03}$
optimal λ	-	$2^{-2.61}$	$2^{-2.19}$

TABLE 1. Predictor performance (MSE) on training and test sets for Linear Regression (Left), Gaussian Kernel Ridge Regression (Center) and Laplacian Kernel Ridge Regression (Right). The optimal hyperparameter values are also reported

Table 2 shows the quality scores for the new batch of wine, as predicted by the 3 trained models. The quality assessment by each model is quite similar.

	Wine 1	Wine 2	Wine 3	Wine 4	Wine 5
Linear	6.0046	5.2876	5.5636	6.0670	5.9424
Gaussian	5.9828	5.4435	5.3428	6.1255	6.0489
Laplacian	6.0363	5.4859	5.6224	5.9569	5.9931

TABLE 2. The prediction of quality scores for the new batch of wine using the 3 models.

5. Summary and Conclusions

We traIned 3 models: Linear Regressor, Gaussian Kernel Ridge and Laplacian Kernel Ridge to predict the quality scores of wine. We found that both Kernel methods performed better than the linear model on the test set. We, also found that Laplacian Kernel showed greater difference in performance between the training and the test set.

Acknowledgements

The author is thankful to Prof. Bamdad Hosseini for useful discussions about the Kernel Trick, Kernel Ridge Regression, Cross-validation and demonstrations associated with these methods. Furthermore, the discussion thread on canvas was extremely helpful.

References

[1] P. Cortez, A. L. Cerdeira, F. Almeida, T. Matos, and J. Reis, "Modeling wine preferences by data mining from physicochemical properties," *Decis. Support Syst.*, vol. 47, pp. 547–553, 2009.
[2] W. S. Noble, "What is a support vector machine?," *Nature biotechnology*, vol. 24, no. 12, pp. 1565–1567, 2006.
[3] H. Sun, "Mercer theorem for rkhs on noncompact sets," *Journal of Complexity*, vol. 21, no. 3, pp. 337–349, 2005.
[4] T. Hangelbroek and A. Ron, "Nonlinear approximation using gaussian kernels," *Journal of Functional Analysis*, vol. 259, no. 1, pp. 203–219, 2010.
[5] G. A. Seber and A. J. Lee, *Linear regression analysis*. John Wiley & Sons, 2012.
[6] V. Vovk, "Kernel ridge regression," in *Empirical inference*, pp. 105–116, Springer, 2013.
[7] P. Refaeilzadeh, L. Tang, and H. Liu, "Cross-validation.," *Encyclopedia of database systems*, vol. 5, pp. 532–538, 2009.
[8] M. Belkin, D. Hsu, S. Ma, and S. Mandal, "Reconciling modern machine-learning practice and the classical bias–variance trade-off," *Proceedings of the National Academy of Sciences*, vol. 116, no. 32, pp. 15849–15854, 2019.

[9] G. Van Rossum *et al.*, "Python programming language.," in *USENIX annual technical conference*, vol. 41, p. 36, 2007.
[10] T. E. Oliphant, *A guide to NumPy*, vol. 1. Trelgol Publishing USA, 2006.
[11] 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*, vol. 12, pp. 2825–2830, 2011.
[12] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in science & engineering*, vol. 9, no. 03, pp. 90–95, 2007.