AMATH 482/582: HOME WORK 3

SHENGBO JIN

CFRM Program, University of Washington, Seattle, WA
Shengboj@uw.edu

Abstract. This report aims at training predictive models to predict the quality of wine from a series of chemical measurements. The main methods consist of linear regression and kernel ridge regression (Gaussian and Laplacian kernel). Firstly, fit linear regression and kernel ridge regression without hyperparameter tuning. Then, using 10-fold cross-validation, fit kernel ridge regression to get better performance. Finally, evaluate model performances and predict with new data.

1. Introduction and Overview

Our goal is to develop an algorithm that predicts the quality of wine from a series of chemical measurements. The data set is split into training and test sets. The training set consists of 1115 instances, and the test set has 479 instances. Each instance of the data has 11 features and corresponding output, which is the quality of the wine on a scale of 0 to 10. Linear regression, Gaussian (RBF) kernel ridge regression and Laplacian kernel ridge regression are considered as predictive models. In addition, we should use 10-fold cross-validation to tune hyperparameters in the last two models. Finally, use the built models to predict the qualities with new data.

In this report, under Python environment, linear regression, kernel ridge regression and hyperparameters tuning are realized by NumPy[1] and Sklearn[3] package. Matplotlib[2] is used for visualization.

2. Theoretical Background

Ridge regression

Ridge regression is a method of estimating the coefficients of multiple-regression models in scenarios where linearly independent variables are highly correlated. It has been used in many fields including econometrics, chemistry, and engineering.

In standard linear regression, an $n \times 1$ column vector y is to be projected onto the column space of the $n \times p$ design matrix X (typically $p \ll n$) whose columns are highly correlated. The ordinary least squares estimator of the coefficients $\beta \in \mathbb{R}^{p \times 1}$ by which the columns are multiplied to get the orthogonal projection $X\beta$ is

$\widehat{\beta} = (X^T X)^{-1} X^T y$ (where X^T is the transpose of X).

By contrast, the ridge regression estimator is

$\widehat{\beta}_{\text{ridge}} = (X^T X + kI_p)^{-1} X^T y$ where I_p is the $p \times p$ identity matrix and $k > 0$ is small. The name 'ridge' refers to the shape along the diagonal of I.

Kernel ridge regression

Kernel ridge regression (KRR) combines ridge regression (linear least squares with l2-norm regularization) with the kernel trick. It thus learns a linear function in the space induced by the respective kernel and the data. For non-linear kernels, this corresponds to a non-linear function in the original space.

Date: February 25, 2022.

The form of the model learned by KRR is identical to support vector regression (SVR). However, different loss functions are used: KRR uses squared error loss while support vector regression uses epsilon-insensitive loss, both combined with l2 regularization. In contrast to SVR, fitting a KRR model can be done in closed-form and is typically faster for medium-sized datasets. On the other hand, the learned model is non-sparse and thus slower than SVR, which learns a sparse model for epsilon>0, at prediction-time.

Gaussian kernel

This kernel is defined as:

$k(x, y) = exp(-\frac{1}{2\sigma^2}||x - y||^2)$

where x and y are the input vectors.

Laplacian kernel

This kernel is defined as:

$k(x, y) = exp(-\frac{1}{\sigma}||x - y||_1)$

where x and y are the input vectors and$||x - y||_1$ is the Manhattan distance between the input vectors.

K-fold cross validation

In k-fold cross-validation, the original sample is randomly partitioned into k equal sized subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining k 1 subsamples are used as training data. The cross-validation process is then repeated k times, with each of the k subsamples used exactly once as the validation data. The k results can then be averaged to produce a single estimation. The advantage of this method over repeated random sub-sampling (see below) is that all observations are used for both training and validation, and each observation is used for validation exactly once. 10-fold cross-validation is commonly used, but in general k remains an unfixed parameter.

For example, setting k = 2 results in 2-fold cross-validation. In 2-fold cross-validation, we randomly shuffle the dataset into two sets d0 and d1, so that both sets are equal size (this is usually implemented by shuffling the data array and then splitting it in two). We then train on d0 and validate on d1, followed by training on d1 and validating on d0.

When k = n (the number of observations), k-fold cross-validation is equivalent to leave-one-out cross-validation.

In stratified k-fold cross-validation, the partitions are selected so that the mean response value is approximately equal in all the partitions. In the case of binary classification, this means that each partition contains roughly the same proportions of the two types of class labels.

In repeated cross-validation the data is randomly split into k partitions several times. The performance of the model can thereby be averaged over several runs, but this is rarely desirable in practice.

3. Algorithm Implementation and Development

Step 1 Data preprocessing

Initialize scaler by *preprocessing.StandardScaler()* and fit it with train set. Then use *scaler.transform()* to normalize and center X_train, Y_train and X_test , Y_test.

Step 2 Base setting model

1. Initialize linear regression model as lin by *LinearRegression()* and fit it with X_train and Y_train. Then use *lin.predict()* and *mean_squared_error()* to get train_mse and test_mse.
2. Initialize Gaussian kernel ridge regression model as krr_rbf by *KernelRidge(kernel='rbf')* and fit it with X_train and Y_train. Then use *krr_rbf.predict()* and *mean_squared_error()* to get train_mse and test_mse.

3. Initialize Laplacian kernel ridge regression model as krr_laplacian by *KernelRidge(kernel=' laplacian')* and fit it with X_train and Y_train. Then use *krr_laplacian.predict()* and *mean_squared_error()* to get train_mse and test_mse.

Step 3 Hyperparameter tuning

1. For Gaussian kernel, use *GridSearchCV(cv=10)* and *KernelRidge(kernel='rbf')* to tune the length scale σ and the regularization parameter λ. This step starts with a few values over a wide range and refine successively.
2. For Laplacian kernel, use *GridSearchCV(cv=10)* and *KernelRidge(kernel=' laplacian')* to tune the length scale σ and the regularization parameter λ. This step starts with a few values over a wide range and refine successively
3. For above models, use cv_results_, best_params_, *np.log2()*, *tricontourf()* and *colorbar()* to show the relationships between model scores and hyperparameters; Use *scatter()* to show the difference between true values and predicted values; Use *mean_squared_error()* to get train_mse and test_mse.

Step 4 New batch predicts

Use *scaler.transform()* to normalize and center X_new_batch, then get predicted values by *predict()*. Finally, inverse transform predicts by *scaler.scale_* and *scaler.mean_*, then scale to 0-10 with *round()*.

4. Computational Results

Task 1&2

Without 10-fold CV, firstly fit linear regression, Gaussian kernel ridge regression and Laplacian kernel ridge regression with X_train and Y_train under default settings. MSEs of all three models:

	linear regression	Gaussian kernel	Laplacian kernel
MSE_train	0.6278484956554817	0.4474673899912707	0.3586005082612414
MSE_test	0.7471696905187133	0.6665992233666976	0.6479752488240716

Table 1. MSEs without hyperparameter tuning

Task 3

For Gaussian kernel, use 10-fold CV to tune the length scale σ and the regularization parameter λ. Firstly, search $log_2\sigma$ and $log_2\lambda$ params in [-10, 10] with length=20, further narrow the length of range into 2, even 0.5.

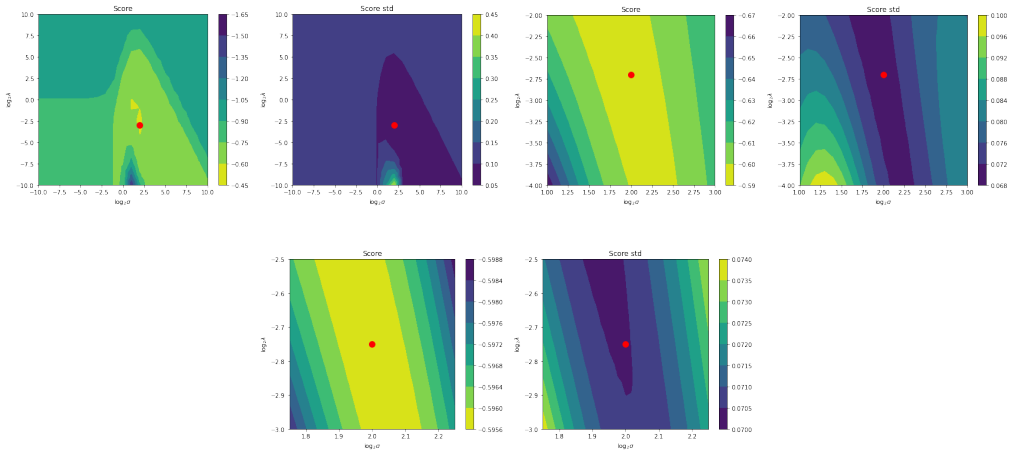


FIGURE 1. Relationships between test scores and hyperparameters

	Length=20	Length=2	Length=0.5
Best $\log_2\lambda$	-3.0	-2.7	-2.75
Best $\log_2\sigma$	2.0	2.0	2.0
Best λ	0.125	0.1538930516681145	0.14865088937534013
Best σ	4.0	4.0	4.0
Best gamma	0.03125	0.03125	0.03125

TABLE 2. Best hyperparameters in different range

The optimal values of $log_2\lambda$=-2.75, $log_2\sigma$=2.0, λ=0.14865088937534013, σ=4.0.

(A) true values and predicted values in train (B) true values and predicted values in test

FIGURE 2. true values and predicted values

The above scatterplots show the difference between true values and predicted values in train and test.

For Laplacian kernel, use 10-fold CV to tune the length scale σ and the regularization parameter . Firstly, search $log_2\sigma$ and $log_2\lambda$ params in [-10, 10] with length=20, further narrow the length of range into 2, even 0.5.

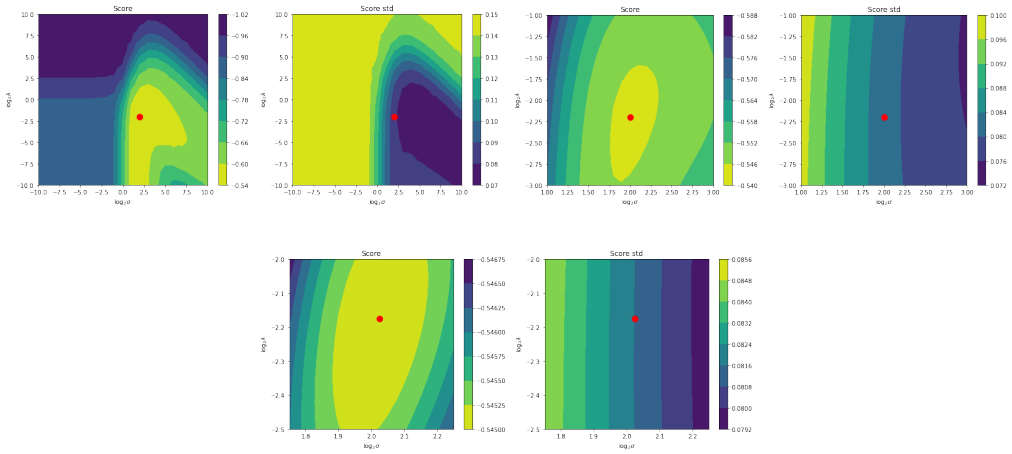
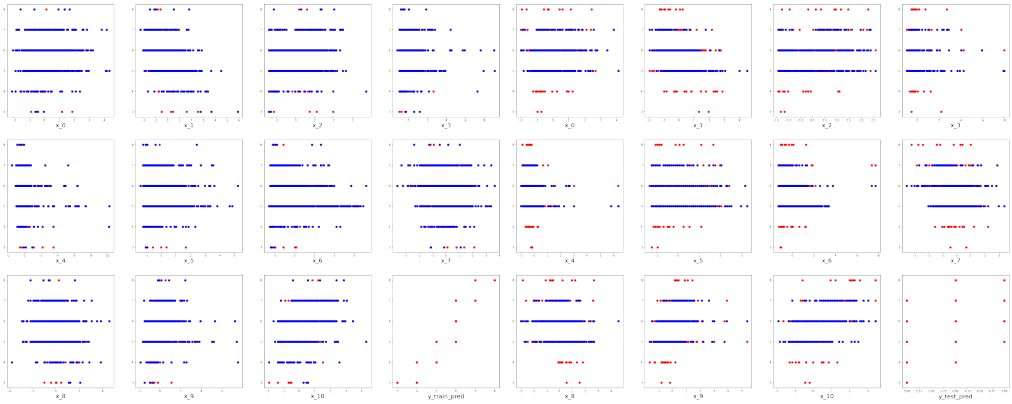


FIGURE 3. Relationships between test scores and hyperparameters

	Length=20	Length=2	Length=0.5
Best log2λ	-2.0	-2.2	-2.175
Best log2σ	2.0	2.0	2.025
Best λ	0.25	0.217637640824031	0.22144187977559018
Best σ	4.0	4.0	4.069918768410745
Best gamma	0.25	0.25	0.2457051496363128

TABLE 3. Best hyperparameters in different range

The optimal values of $log_2\lambda$=-2.175, $log_2\sigma$=2.025, λ=0.22144187977559018, σ=4.069918768410745.



(A) true values and predicted values in train (B) true values and predicted values in test

FIGURE 4. true values and predicted values

The above scatterplots show the difference between true values and predicted values in train and test.

Task 4

	linear regression	Gaussian kernel	Laplacian kernel
MSE_train	0.6278484956554817	0.4578972706981647	0.05086498306536009
MSE_test	0.7471696905187133	0.6836021880811453	0.6067950903037753

Table 4. MSEs with hyperparameter tuning

To some large extent, both Gaussian kernel and Laplacian kernel have smaller MSE than linear regression in training set and test set, so they are better models. In addition, according to scatterplots about true values and predicted values, we can find Laplacian kernel has extraordinary performance in training set, but ordinary in test set with much larger test_MSE than train_MSE. This is to say, Laplacian kernel model is overfitting and has high variance.

Task 5

	Predicted value	Predicted value on the 0-10 scale
lin	[6.00469789 5.28767761 5.56363072 6.067022 5.94248207]	[6. 5. 6. 6. 6.]
rbf	[5.97810847 5.44072286 5.33280072 6.11517657 6.04015304]	[6. 5. 5. 6. 6.]
laplacian	[6.03508511 5.48684335 5.62196596 5.95552636 5.99181821]	[6. 5. 6. 6. 6.]

Table 5. Predicted value

5. Summary and Conclusions

Firstly, in model selection part, using cross-validation and grid search (sort of exhaustive search) to tune hyperparameters is very computational expensive. In this process, only adding one new optional value often extends computation time a lot. Thus, a good approach is to start with a few values over a wide range and refine search successively.

Secondly, good performance in training set may not be a good thing! Frequently, such models cannot reach desired performance on test set or new data, which is considered as overfitting and high variance. Overfitting model has low generalization ability with new data.

Acknowledgements

The author is thankful to Prof. Hosseini for lectures about signal processing. I am also thankful to Dr. Owens and Dr. Levin for careful guidance about the report. Furthermore, My peers Xuqing Hu and Xiangyu Gao were helpful in the use of Latex.

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

[3] 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.