AMATH 582
HW2: Classifying Digits

Daniel Leon

February 26, 2022

Abstract

Real world data about Portuguese red wine is used to investigate the use of kernel methods along with linear and ridge regression for constructing and training a model that takes data about eleven chemical properties makes predictions about the quality of an unknown wine based on the scores for those wines from professional wine raters. The linear least squares model resulted in $MSE_{train} = 0.4803$ and $MSE_{test} = 0.5690$, and predicted an average score of 5.8 for the 5 new wines. The 10-fold cross-validation was performed with both Gaussian and Laplacian kernels to find optimal trade-off between minimizing MSE_{train} and MSE_{test} given computational and time constraints. For the Gaussian kernel, we obtained $MSE_{train} = 0.3627$ and $MSE_{test} = 0.4791$ with $\sigma = 3.0847$ and $\lambda = 1.2723$. Those hyperparameter values gave the 5 new wine batches an average score of 5.6 with the Gaussian. Then for the Laplacian kernel, we obtained $MSE_{train} = 0.0224$ and $MSE_{test} = 0.4623$ with $\sigma = 3.3390$ and $\lambda = 1.2387$. Those hyperparameter values gave the 5 new wine batches an average score of 6.

1 Introduction & Overview

We have been hired by a winery in Portugal as a data scientist to develop models that can predict the quality of a batch of wine simply from a Series of measurements of chemical properties (pH, sulphates, alcohol %, etc), We have a curated dataset containing 11 measurements for 1,592 red wines along with an integer score on a scale between 0 - 10, as judged by a professional wine tester. We must construct and evaluate three different models, first with linear regression, then using ridge regression with the kernel method using Gaussian and Laplacian kernels.

When classifying data by its features we often associate a certain outcome to be associated with a cluster of feature values that is distinct from the clusters of of other outcomes. In simple cases, a line (plane or hyperplane) can be found by using linear regression to separate the clusters, but in more complex cases, simple linear regression won't cut it. Kernels offer an appealing approach by embedding the data in a suitable feature space patterns can be discovered as linear relations. We will go into more detail about kernel space in the next section, but for now we will mention that the kernel has a parameter σ that needs to be optimized to obtain an optimal fit.

After obtaining optimized parameters σ & λ for both kernel ridge regression models, the models will be used to predict the score from some new data that doesn't have an accompanying score labels. We will not be able to compute the MSE for the new batch because we do not have the true value of the quality, but we will do our best to evaluate the strength of the predictions.

2 Theoretical Background

2.1 Preprocessing the Data

The data needs to be split split into training and testing data sets, with the training set being larger than the test set. Before undergoing kernel regression, the data needs to be scaled to have a mean of 0 and a unit variance, with respect to the training data. So the testing data and all data for which the model is used as a predictor must be centered and scaled using the training data mean and standard deviation. Without this step, the procedure will not work.

2.2 Choice of Regression Model

We can first consider the problem of primal linear regression, where we want to find a homogeneous real-valued linear function

$$g(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle = \mathbf{w}^* \mathbf{x} = \sum_{i=1}^{n} w_i x_i \tag{1}$$

that does the best job possible interpolating the training set $S = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_\ell, y_\ell)\}$ of data point/label pairs $(\mathbf{x}_i, y_i)$. If X^*X is invertible, then the solution to the linear least squares problem is

$$\mathbf{w} = (X^*X)^{-1}X^*y = X^*X(X^*X)^{-2}X^*y = X^*\alpha \tag{2}$$

which is known as *dual representation*, and is found by taking the derivative of the loss function of the optimization problem ($\mathcal{L}(\mathbf{w}, S)$). Thus we have a linear combination of the training points $\mathbf{w} = \Sigma_{i=1}^{\ell} \alpha \mathbf{x}_i$.
For a variety of reasons, some problems are ill conditioned, and they require a regularization that restricts the bias. The simplest regularizer is to prefer functions with a small norm. In the case of linear least squares, we get the ridge regression minimizer of the form

$$\min_{\mathbf{w}} \mathcal{L}_\lambda(\mathbf{w}, S) = \min_{\mathbf{w}} \lambda ||\mathbf{w}||^2 + \Sigma_{i=1}^{\ell} (y_i - g(\mathbf{x}_i))^2 \tag{3}$$

with $\lambda \in \mathbb{R}^+$.
As before, we have to take the derivative of the loss function w.r.t. the parameters and set it equal to zero. That results in a prediction function given by

$$g(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle = y^* X (X^*X + \lambda I)^{-1} \mathbf{x} \tag{4}$$

where I is the identity matrix. For ridge regression, we have the dual problem of the form

$$\mathbf{w}_{ridge} = \lambda^{-1} X^* (y - X\mathbf{w}) = X^* \alpha \tag{5}$$

showing again we can express it as a linear combination of the training points $\mathbf{w} = \Sigma_{i=1}^{\ell} \alpha \mathbf{x}_i$, where $\alpha = \lambda^{-1}(y - X\mathbf{w})$. We will investigate two ridge regression models and search for an optimal λ value in the cross-validation process. [5] [2] [3]

2.3 Kernels

Sometimes the data exhibits non-linear patterns in its standard feature space. We can choose a kernel with functions $\phi(\mathbf{x})$ that projects the data into a space where the patterns appear linear, meaning that they can be separated by a line. A kernel is a function : $\mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, and the most useful ones to us are non-negative definite and symmetric (NDS) matrices. If that is the case, then we have

$$\kappa(\mathbf{x}, \mathbf{x}') = \Sigma_{j=0}^{\infty} \phi_j(\mathbf{x}) \phi_j(\mathbf{x}') \tag{6}$$

where $\phi_j : \mathbb{R}^N \to \mathbb{R}$ are the features of $\mathbf{K}$. The corresponding space of functions $\mathcal{H}_\kappa$ is called a *reproducing kernel Hilbert space*, or RKHS. This tells us that if x, x' in the RKHS have a small norm $||x - x'||$, then x and x' will be pointwise close. There are many types of kernels to choose from, but here we will use the Gaussian (rbf) kernel and the Laplacian (lap) kernel

$$\kappa_{rbf}(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||_2^2}{2\sigma^2}\right), \kappa_{lap}(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||_1}{\sigma}\right), \text{for } \mathbf{x}, \mathbf{x}' \in \mathbb{R}^{11} \tag{7}$$

[5] [3] [1]

2.3.1 Mercer's Theorem

If we have a set of vectors $S = \{\mathbf{x}_1, ..., \mathbf{x}_\ell\}$, then the Gram matrix is defined as the $\ell \times \ell$ matrix $\mathbf{G}_{ij} = \langle \mathbf{X}_i, \mathbf{X}_j \rangle$. When using a kernel κ with features ϕ, then we have the following Gram matrix

$$\begin{aligned} \mathbf{G}_{ij} &= \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle = \kappa(\mathbf{x}_i, \mathbf{x}_j) \\ \Rightarrow \mathbf{K} &= \begin{pmatrix} \kappa(\mathbf{x}_1, \mathbf{x}_1) & \cdots & \kappa(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \cdots & \vdots \\ \kappa(\mathbf{x}_N, \mathbf{x}_1) & \cdots & \kappa(\mathbf{x}_N, \mathbf{x}_N) \end{pmatrix} \end{aligned} \tag{8}$$

If $\mathbf{K}$ is NDS for any set of inputs $\{\mathbf{x}_{i=1}^N\}$, the the kernel is a *Mercer kernel*, which is important because of *Mercer's theorem.*

Theorem 1 *(Mercer's Theorem):*
If $\mathbf{K}$ *is NDS and integrable then there exists an orthonormal basis* $\{\phi_j|_{j=0}^{\infty} \in L^2(\mathbb{R}^n)\}$ *such that*

$$\kappa(\mathbf{x}, \mathbf{x}') = \Sigma_{j=0}^{\infty} \lambda \phi_j(\mathbf{x}) \phi_j(\mathbf{x}')$$

with $\lambda_j \geq 0$ *are the eigenvalues, and* ϕ_j *are the eigenfunctions of* $\mathbf{K}$

Since $\mathbf{K}$ is NDS, it has an eigenfunction expansion, which allows us to write the entries of the kernel matrix as the inner product of some feature vectors that are implicitly defined by the eigenvectors of the kernel matrix. Generally, if you have a Mercer kernel, then there exists a function ϕ which maps $\mathbf{x} \in \chi \to \mathbb{R}^D$ such that

$$\kappa(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x}_i)^* \phi(\mathbf{x}_j) \tag{9}$$

where ϕ depends on the eigenfunctions of , which means that the dimesion D could be infinite. The possibility of trying to deal with an infinite number of features is resolved by the kernel trick. [2] [1] [3]

2.3.2 Kernel Trick

Instead of defining the feature vectors in terms of kernels, the kernel trick allows us to replace any inner products $\langle \mathbf{x}, \mathbf{x}' \rangle$ with a call to the kernel function, $\kappa(\mathbf{x}, \mathbf{x}')$. This is huge because we can combine that with the dual representation interpretation in eq (2) which results in

$$\Sigma_{j=0}^{N-1} \alpha \kappa(\mathbf{x}_j, \mathbf{x}_i) = y_i, \Rightarrow \Theta \alpha = \mathbf{y} \tag{10}$$

where $\Theta_{ji} = \kappa(\mathbf{x}_j, \mathbf{x}_i)$. That is a finite sum, and thus it is calculable. So we get Θ for free by calculating $\kappa(\mathbf{x}, \mathbf{x}')$ [3] [5] [2]

2.4 Cross-validation

Along with optimization strategies, determining whether a given model is under-fit or over-fit with respect to the data is of great concern in data science. As discussed before, it would be a mistake to evaluate the models performance with data that was used to train it. Over-fitting can result data leakage, from excessive tweaking of parameters, or from having a model that is overly complex. [4]

We can optimize our regression fit by iteratively computing it across a range of λ and σ values and selecting the best MSE_{test} values.

2.5 Error Reporting

Measuring the error rate for a given model gives a measure of it effectiveness as a model. One way to gauge the *fitness* of a fit is by minimizing the mean squared error (MSE) of the test set of data (MSE_{test}). The MSE is given by

$$MSE_t = \frac{1}{\text{length of } b_t} \times ||\mathbf{x}_t \hat{\beta} - b_t||_2^2 \tag{11}$$

where the subscript t is a placeholder for either *train* or *test*. The MSE_{train} will naturally be smaller than MSE_{test}, but we don't want to $over-fit$ the training data, so there is a balance.

3 Algorithm Implementation & Development

All algorithms were developed in python 3.8.5 in the Spyder integrated development environment. Packages used include numpy, scikit-learn, plotly, and matplotlib.
There are 3 main steps to the algorithm, but there are some sub-steps along the way.

3.1 Loading Data

The data needed to be loaded from separate .csv files containing the training data, test data, and the new batch data. The **X** and **Y** data needed to be extracted from the main dataframe. Headers were added to the pandas dataframe to aid in presentation.

3.2 Centering Data and standardizing the data

All data was standardized using the mean and standard deviation of the training set.

3.3 Cross Validation set up

For cross validation (skipped for regular linear regression), we set a range for both σ (acts as the effective width of the kernel) and λ (regularization parameter for ridge regression). A grid spacing must be chosen as well to pick how many points to test within each range. The more points chosen, the finer the grid, so you can get more precise values, but that is at the expense of computational cost. We picked a wide range at first and iteratively fine tuned the range as we went forward.
We should also note that we used the relationships for $\gamma = \frac{1}{2\sigma^2}$ for the Gaussian kernel and $\gamma = \frac{1}{\sigma}$ for the Laplacian kernel. We used $\alpha = 2^{\lambda}$ for both models. For the sake of reporting values, these are the relations between them and the model. We knew about the function GridSearchCV, but chose to implement the CV scheme manually using 'for' loops.

3.4 Linear or Ridge Regression Fit

First we initialize a new instance of the regression class with the current iteration of the parameters, then we *fit* it using the X_{train}/Y_{train}. For ridge regression, chose the kernel you want to use.

3.5 Predict

X_{test} is used in the recently fit model to make a prediction $\tilde{Y}_{test}$. we also use the model to predict $\tilde{Y}_{train}$ from X_{train}. After obtainin a prediction, that prediction neede to be un-normalized and rounded to the nearest integer before scoring against the original non-normalized training/testing qualities.

3.6 Error Evaluation and CV Iteration

Compute MSE_{test} and MSE_{train}. Iterate if needed: if the hyper parameters had their optimal value at either edge of the range we set, then we re-ran with a shifted range. If the converged value was somewhere in the middle of the range, we would close the range around the point. If we had the computational power, we would make a finer grid to search.

4 Computational Results

4.1 Task 1: Linear Regression

The linear regression did not have any parameters that needed tuning. Instructions say to include all of the data into a single table under task 4, so look ahead for results.

4.2 Tasks 2&3: 10-Fold Hyperparameter Tuning

We were able to hone in on some pretty good parameter values through iteration. The 3D plots, made with plotly, were very useful for determining the correct adjustments to make to the range of the parameters, especially because the 3D plots from plotly can orbit by default with the mouse. Our final grid search use 60 points in each parameter.

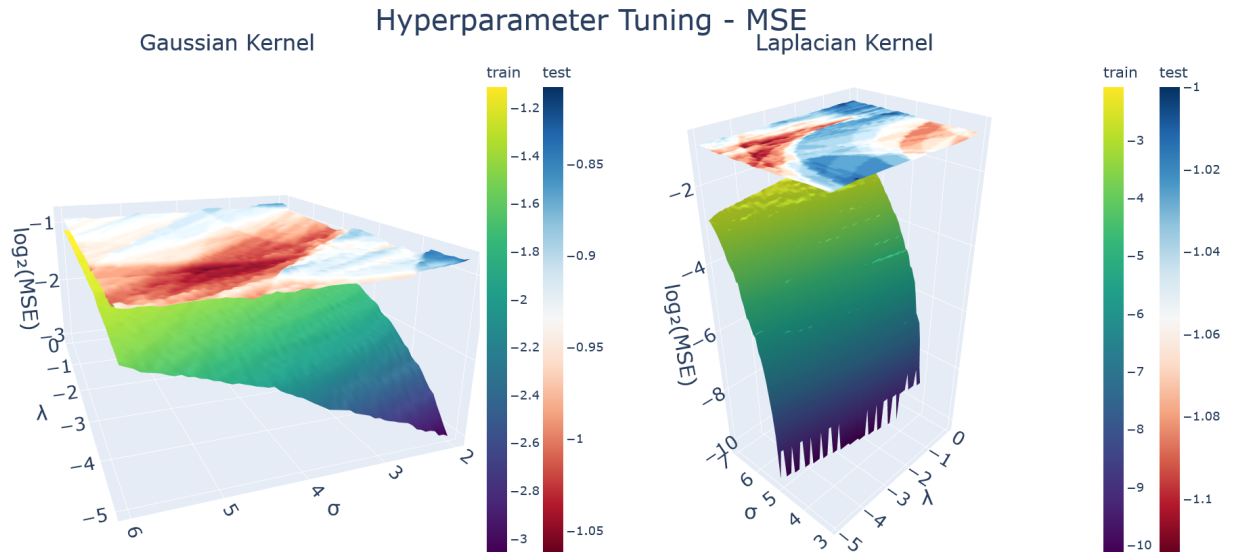

Figure 1: The 3D plot of MSE_{test} (top) and MSE_{train} (bottom) plotted vs. the range of σ and λ for Gaussian (left) and Lapacian (right) kernels. the grid spacing on the parameters was 60×60. Plot produced using plotly

Model	σ_{opt}	γ_{opt}	λ_{opt}	α	MSEtrain	MSEtest
Linear Regression	na	na	na	na	0.4803	0.5690
Gaussian Kernel	3.0847	0.0525	1.2723	2.4155	0.3627	0.4791
Laplacian Kernel	3.339	0.0448	1.2387	2.3600	0.0224	0.4623

Table 1: MSE values for optimized values of hyperparameters

4.3 Task 4: Optimized Models

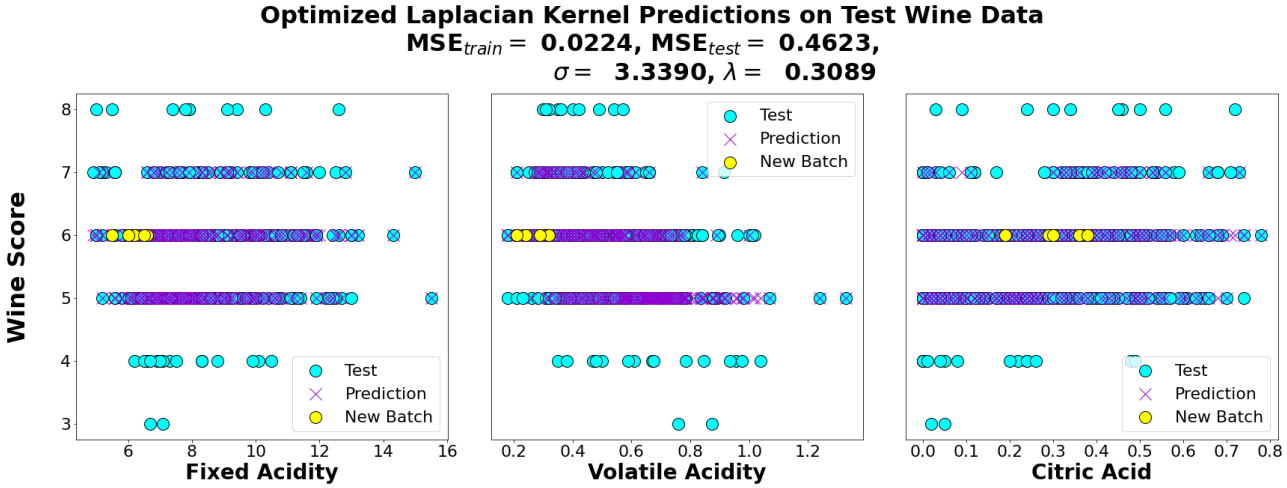

Figure 2: Depiction of the first three out of eleven features from test set with the true and predicted scores using the Laplacian kernel. The new batch data quality predictions are included as well. The full plot has the data for all 11 features, and there are similar plots for the training data as well as for the other regressions.

4.4 Task 5: New Batch Prediction

We had data for 5 new wines that didn't have a quality score. The following table shows the quality predictions from each of the models.

Model	Wine #: 1	2	3	4	5	mean
Linear Regression	6	5	6	6	6	5.8
Gaussian Kernel	6	5	5	6	6	5.6
Laplacian Kernel	6	6	6	6	6	6

Table 2: Predicted quality for each of the 5 wines with each model.

5 Summary & Conclusions

We used linear and ridge regression to develop models to predict the quality of wine based on chemical data. The linear least squares model resulted in $MSE_{train} = 0.4803$ and $MSE_{test} = 0.5690$, and predicted an average score of 5.8 for the 5 new wines. We used cross-validation on the parameters for both Gaussian and Laplacian kernels and found good trade-off between minimizing MSE_{train} and MSE_{test} given computational and time constraints. For the Gaussian kernel, we obtained $MSE_{train} = 0.3627$ and $MSE_{test} = 0.4791$ with $\sigma = 3.0847$ and $\lambda = 1.2723$. Those hyperparameter values gave the 5 new wine batches an average score of 5.6 with the Gaussian. Then for the Laplacian kernel, we obtained $MSE_{train} = 0.0224$ and $MSE_{test} = 0.4623$ with $\sigma = 3.3390$ and $\lambda = 1.2387$. Those hyperparameter values gave the 5 new wine batches an average score of 6.
Although the quality scale was from 0-10, out of the 1,592 wines in both the training and test sets, all had quality ratings between 3-8. Of the 1,114 wines in the training set, over 82% of them were either rated 5 or 6 (460 and 459 respectively). Of the remaining 18%, two-thirds of them were rated 7 (141). The remainder saw 38 wines rated with quality 4, and 8 each for the lowest and highes scores of 3 and 8. in this light, it is not surprising that all of the models tended to predict values in the middle of the range
Of all of the models, the regular linear regression had the worst performance, and the Laplacian kernel gave the best results, but the difference when it can to predictions of the new data was minor (see table 2). If we look at figure 2 we can see that the model pretty much only predicts 5's, 6's, and 7's. This is not unique to the features shown or to the Laplacian kernel alone, as every feature from every model pretty much looks like that, very sparse on the top and bottom. That probably has a lot to do with the makeup of the training set, which had 95% of its data being 5, 6, or 7. The basic linear regression was expected to be the worst, since it isn't doing any transforming to hopefully more ideal basis. The iteration over hyperparameters takes a long time when doing it the way that I did using for-loops. I now know that GridSearchCV has accommodations for parallelezation. So that would be a good way to speed up the CV analysis in future implementations.

6 Acknowledgements

I would like to thank Jon Staggs. Our conversations about this assignment early on helped me to formulate this general approach. we consulted when questions arose with our code and on interpretation of results.

References

[1] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[2] Bamdad Housseini. *AMATH 582, Lecture 15*, volume 50. University of Washington, 2022.

[3] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[4] et. al Pedregosa, F. Scikit-learn: Machine learning in Python.

[5] John Shawe-Taylor, Nello Cristianini, et al. *Kernel methods for pattern analysis*. Cambridge university press, 2004.