Final Project: Wine Quality Analysis Using Machine Learning

Ian Strawn

2025-06-09

Introduction

Humanity has been crafting wines for some 8000 years, and our pursuit of creating the perfect wine has yet to slow down. Winemaking techniques have become so advanced, in fact, that many winemakers now take chemical measurements of their wines in order to ensure that certain production standards are met. These chemical measurements are almost certainly linked to the quality of the wines being produced, but how *well* do these chemical measurements predict a wine's quality? To answer this question, we can use machine learning regression techniques.

The purpose of this paper is twofold. The primary goal is to take a variety of machine learning (henceforth referred to as ML) regression techniques and apply them to data sets with large sample sizes and relatively small amounts of predictors to see how they perform. Determining what the "best" modeling process is will be discussed in the methods section. The two data sets being used involve red and white Portuguese wines, and due to their different production techniques, their chemical compositions are likely to be different. As such, once the models for the data sets have been created and selected, the secondary goal is to compare the results from the two wine types to try to draw qualitative conclusions about the differences between red and white wines and what goes into the quality of those wines.

Data & Methods

Data

The data sets that I'll be using to analyze the effectiveness of these models come from a wine study published in 2009. In the study, which was conducted from 2004-2007, samples of batches of Portuguese wine had chemical measurements done on them and were subsequently rated on their quality by wine experts. Each sample was rated by at least 3 experts, and the median response was taken as that sample's quality score. Both white and red wines were produced, and we thus have two data sets to analyze - one for each type of wine. The data set involving the red wines has 1599 total samples, while the data set involving the white wines has 4898 total samples.

The following are the 12 variables included in the two data sets (the variables remain the same between the two data sets):

- `quality`: The quality score of the sample, measured from 0-10 (with 0 being terrible and 10 being excellent)
- `fixed.acidity` (grams of tartaric acid/dm^3)
- `volatile.acidity` (grams of acetic acid/dm^3)
- `citric.acid` (grams of citric acid/dm^3)

- `residual.sugar` (grams/dm^3)
- `chlorides` (grams of sodium chloride/dm^3)
- `free.sulfur.dioxide` (milligrams/dm^3)
- `total.sulfur.dioxide` (milligrams/dm^3)
- `density` (grams/cm^3)
- `pH`: The pH of the sample
- `sulphates` (grams of potassium sulphate/dm^3)
- `alcohol`: % volume of alcohol of the sample

Model Types

There are 5 models that I'll create to try to predict `quality` from the other variables. The first model will be multiple regression, to serve as a baseline comparison for the other ML techniques. The ML techniques themselves will be LASSO, PCA, PLS, and Ridge. Each will be discussed here.

Multiple Regression

Multiple Regression assumes that we can take a numeric response variable with n observations, $\mathbf{y} = (y_1, y_2, ..., y_n)$, and model it using a series of p predictors $\mathbf{x} = (x_1, x_2, ..., x_p)$. It assumes that the relationship between y and each of the predictors is linear, and the model itself can be expressed as follows:

$$\hat{y_i} = \beta_0 + \sum_{j=1}^{p} x_{ij} \cdot \beta_j$$

Where $\hat{y}$ represents our predicted value of y, i represents the ith observation of the variables ($i = 1, 2, ..., n$), and each $\beta_j \in \mathbb{R}$ are some constant that we multiply each of our variables by. To find our optimal β's, we first define $\mathbb{X}$ as a matrix of all of our predictors, with each of $(x_1, x_2, ..., x_p)$ getting its own column, and also adding a leading column of 1's (to account for an intercept term that doesn't change regardless of the values of the predictors). Using ordinary least squares to estimate each value of β, we find that our optimal values of the β_j's are found with

$$\hat{\boldsymbol{\beta}}_{OLS} = (\mathbb{X}^T\mathbb{X})^{-1}\mathbb{X}^T\mathbf{y}$$

for $\boldsymbol{\beta} = (\beta_0, \beta_1, ..., \beta_p)$.

PCR

PCR, short for Principle Component Regression, is an unsupervised dimension reduction technique, meaning that the predictors are transformed, and then a linear model is fit using the transformations. The dimension-reduction aspect of PCR is what makes it useful - by looking at transformations of the explanatory variables, we can *reduce* the effective number of moving parts in our model without losing too much information. The first step is to standardize our predictors, so that each of our x_j's has a mean of 0 and a variance of 1. Assuming we have p predictors, we create $M < p$ linear combinations of the predictors as follows:

$$Z_m = \sum_{j=1}^{p} \phi_{jm} X_j$$

where $\phi_{1m}, \phi_{2m}, ...\phi_{pm}$ are constants for each of our $m = 1, 2, ..., M$ linear combinations. Our linear model using PCR is created exactly like linear regression:

$$\hat{y_i} = \sum_{m=1}^{M} \hat{\theta}_m z_{im}$$

The tricky part with PCA comes with estimating each of the ϕ's and optimizing M. We try to pick the ϕ's to maximize the variance of the Z_m's, which allows us to retain as much information as possible given the size of M. In general, Z_1 is the linear combination with the highest variance (also called the principle component), Z_2 is the component with the largest variance *of all of the remaining linear combinations that are uncorrelated with* Z_1, etc, and that process continues for all subsequent Z_m's. Once the Z_m's are created, we determine what our optimal M is according to a criteria for measuring the model's error. This penalty will be described in the Methods section.

PLS

PLS (also known as Partial Least Squares) does essentially the same thing that PCR does, except it uses our response vector **y** to inform how it creates the linear combinations of the explanatory variables. In general, PCR does a better job of fitting the predictors closely, while PLS does a better job of explaining the response (so I expect it to perform better when we begin training it on data).

We begin by standardizing the p explanatory variables, as we did with PCR. The first direction of PLS, Z_1, calculates each of its ϕ_{j1} by regressing Y onto X_j and finding the slope coefficient from that simple regression. This essentially means that the ϕ_{j1}'s are created to have greater weight for the X_j's that are more highly correlated with Y. To determine the second direction Z_2, we regress each variable on Z_1 and take the residuals from that regression. This orthogonalized data can then be used to compute Z_2 in exactly the same way that we computed Z_1 (i.e. by regressing the residuals onto each X_j and finding the slope coefficients).

We do this M times to identify as many components as we deem useful, treating M as a tuning parameter. Once M has been identified, we use OLS to fit a linear model to predict Y from $Z_1, Z_2, ..., Z_M$, just like we did with PCR.

LASSO

LASSO (or Least Absolute Shrinkage and Selection Operator) is a regression technique which combines penalization of predicted coefficients with some low-level variable selection. In other words, LASSO can penalize certain variables that may not be as useful to our model and completely toss others. Recall that with Multiple Regression, our goal was to compute $\hat{\boldsymbol{\beta}}$, which we could multiply by the values of our explanatory variables to try to predict response values. With LASSO, our goal remains the same, but we formulate our requirements differently:

$$\hat{\boldsymbol{\beta}}_{LASSO} = argmin_{\beta}\{\frac{1}{2}\sum_{i=1}^{n}(y_i - \mathbf{x}_i^T\beta)^2 + \lambda\sum_{j=1}^{p}|\beta_j|\}$$

This looks noticeably different than what was written for multiple regression, but it's just a different formulation of the same idea, with an added penalty term (the term in the equation with the λ). This penalty term serves to shrink each of our predicted coefficients, and the fact that we have an absolute-value involved means that it can even force some of the coefficients to be 0. λ is another tuning parameter here, which we try to optimize to minimize model error. No closed-form solution to the equation above exists, so we leave it in that form, but we can solve it computationally to get estimates for each of our β_j's.

Ridge

Ridge behaves very similarly to Lasso in that it penalizes regression coefficients, but it does so in a different manner. Rather than having an L1 penalty like Lasso does (the absolute value in the equation in the Lasso section), Ridge instead imposes an L2 penalty, which involves squaring the predicted coefficients, not taking the absolute value of them. This gives it the following form:

$$\hat{\boldsymbol{\beta}}_{\lambda}^{R} = argmin_{\beta}\{\sum_{i=1}^{n}(y_i - \mathbf{x}_i^T\beta)^2 + \lambda\sum_{j=1}^{p}\beta_j^2\}$$

Notice the β^2 term, which is our penalty, and the tuning parameter λ, which operates the same way that it does for Lasso. We aim to find a λ that minimizes our model error, and then find the regression coefficients once that λ has been identified.

Methods

With these modeling techniques in mind, the goal will be to analyze which of them best models the data sets at hand. To generate the models, I'll be doing a form of cross-validation (CV), where the data is separated into 10 "folds" (samples of equal size) and the folds are used to train the model and then test its effectiveness. One fold will be designated as the "testing" fold, while the remaining 9 folds will be the data on which the model will be generated. Once that model has been generated from the training folds, we'll plug in the explanatory variable values from the testing fold into the model to generate predictions, and then compare those predictions to the actual response values from that set. Mathematically, that prediction difference looks like the following:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)$$

where y_i is the ith response value, and $\hat{y}_i$ is the corresponding prediction. From here onwards, I'll be referring to the MSE as the CV error to stay consistent with what we've been doing in class. We repeat this process 10 times so that *every* fold gets treated as the testing set exactly once, and we get 10 CV errors for each model.

Additionally, with all of the model types except for multiple regression, the entire CV process will be done twice - one within the other. The functions that we use to generate these models *automatically perform CV on the data that they're given*, so by manually performing cross validation *and* allowing the functions to cross validate the data they're fed, we're performing CV within CV. This process helps us to avoid overfitting, and helps us ensure that our models are maximizing the correctness of their predictions.

Model Analysis

Model generation was done in R using various packages, predominantly the `glmnet`, and `pls` packages. Table 1 shows the average CV error from each of the 5 models for both wine types, and it can be seen that the average errors for all 5 models are surprisingly similar. For red wine, the maximum CV error came from LASSO with an average error of 0.4363, and the minimum came from ridge, with an error of 0.4344. For white wine, the maximum error came from PCR (0.5781), while the minimum came from Ridge (0.5759).

For the models that have interpretable coefficients (Multiple Regression, Lasso, and Ridge), the average values for each of the coefficients from the red wine samples are listed in table 2, while the average values for the ones from the white samples are listed in table 3.

For specifics on the model generation process, see the Code section of the appendix.

Table 1: Average CV Errors for Each Model Type

Wine Type	Multiple.Regression	PCR	PLS	LASSO	Ridge
Red	0.4359	0.4346	0.4348	0.4363	0.4344
White	0.5760	0.5781	0.5775	0.5766	0.5759

Table 2: Average Variable Coefficient Values for Red Wine

Variable Name	Multiple.Regression	Lasso	Ridge
intercept	21.893	13.663	31.894
fixed.acidity	0.024	0.015	0.030
volatile.acidity	-1.081	-1.055	-1.017
citric.acid	-0.184	-0.104	-0.091
residual.sugar	0.016	0.009	0.019
chlorides	-1.879	-1.774	-1.816
free.sulfur.dioxide	0.004	0.003	0.004
total.sulfur.dioxide	-0.003	-0.003	-0.003
density	-17.782	-9.593	-28.033
pH	-0.422	-0.388	-0.319
sulphates	0.926	0.876	0.898
alcohol	0.276	0.280	0.254

Table 3: Average Variable Coefficient Values for White Wine

Variable Name	Multiple.Regression	Lasso	Ridge
intercept	152.4710	110.8040	68.5370
fixed.acidity	0.0670	0.0280	-0.0040
volatile.acidity	-1.8630	-1.8720	-1.7950
citric.acid	0.0210	0.0150	0.0210
residual.sugar	0.0820	0.0650	0.0460
chlorides	-0.2350	-0.4050	-1.0290
free.sulfur.dioxide	0.0040	0.0040	0.0040
total.sulfur.dioxide	-0.0002	-0.0002	-0.0006
density	-152.5810	-110.1040	-67.0310
pH	0.6910	0.5050	0.3650
sulphates	0.6360	0.5440	0.4940
alcohol	0.1910	0.2350	0.2640

Discussion

There are a few interesting attributes from the models that are worth our attention. First, based on the CV error, it seems like Lasso performed relatively poorly (it was the third-worst model for predicting white wine quality and the worst-performing model for red). Lasso works best in situations where the underlying model is relatively sparse (i.e. the “true” best model does not include all of the predictors available), and since it performed relatively poorly, it seems likely that our underlying model is not sparse at all.

It's also worth noting that PCR and PLS were the two worst-performing models with white wine, but the 2nd and 3rd best performers with the red wine data set. Since PCR and PLS inherently keep all of the predictors in their analysis, combined with the fact that LASSO performed poorly with red but did better with white, this seems to suggest that while the underlying red model is likely not sparse, the underlying white model could be. Taking this one step further, it could be argued that the chemical measurements taken on wine samples are better calibrated for red wine samples rather than white wine samples, but this is currently speculatory - further analysis would have to be conducted to confirm that suspicion.

Another detail of note is that multiple regression, the simplest of the 5 regression techniques, performed better than I would've anticipated. It was the second-worst performing model for red wine, but the second-best for white wine. I would've predicted that the other four models would have outperformed multiple regression in both categories, seeing as how they generally accommodate for more complication in the data. However, MR's solid performance implies that simpler models are sometimes better.

Looking at the coefficients in tables 2 and 3, we can also draw some qualitative conclusions about some of the chemical properties of the wines in relation to their quality. First (and arguably most notably), wines of higher density tend to have much lower quality, as evidenced by the large negative coefficients in each of the models. Additionally, citric acid concentration behaves differently for white wine than for red wine - for red wine, higher citric acid levels detract from a wine's quality, while it does the opposite for white wines. pH has a similar opposite relationship between the two wines, in that a higher pH predicts a lower quality of red wine, but a higher quality of white wine. Finally, it's also curious to see that a higher volume of alcohol improves the quality of both white and red wines. The taste of straight alcohol is not generally a good one, so I can only imagine that the higher alcohol content improves the quality of the samples for its… "uplifting" qualities.

Conclusion

In short, while some models performed better in each data set than others, the differences between all of them were so close that any of the model types would do a decent job of predicting wine qualities. If I were to do this project again, I would also choose to add in some of our other modeling techniques, such as elastic net and spline fitting. I feel that elastic net would likely outperform both Ridge and Lasso (seeing as how it finds a healthy medium between the two), and I could see the spline fitting working nicely as well, as it seems likely to me that at least one of the explanatory variables doesn't have a perfectly linear relationship with a wine sample's quality. If nothing else, at least now I have a good idea about what questions I should be asking about a wine's chemical compositions at my next wine-tasting endeavor.

Appendix

Code

Data Cleaning

```r
#Here's the data
red_wine = read.csv2("winequality-red.csv")
white_wine = read.csv2("winequality-white.csv")

#Necessary Libraries
library(tidyverse)
library(glmnet)
library(pls)
library(gt)

#Organizing the Data for our functions
y_red = red_wine[,12]
y_white = white_wine[,12]
x_red = apply(as.matrix(red_wine[,-12]), 2, as.numeric)
x_white = apply(as.matrix(white_wine[,-12]), 2, as.numeric)
```

Multiple Regression

```r
#Multiple Regression Stuff!
set.seed(11)

#Some additional variables
K = 10
n_white = nrow(white_wine)
n_red = nrow(red_wine)

red_reg_errs = rep(0,K)

white_reg_errs = rep(0,K)

mult_betas_white = matrix(0, nrow = K, ncol = ncol(red_wine))
mult_betas_red = matrix(0, nrow = K, ncol = ncol(red_wine))

#Beginning CV (This is just one CV, not 2 like the others)

#Model Creation
for(k in 1:K){
    #Splitting Data
    test.inds.red <- (floor((k - 1) * n_red / K) + 1) :
      ceiling(k * n_red / K)
    test.inds.white <- (floor((k - 1) * n_white / K) + 1) :
      ceiling(k * n_white / K)

    #Kth fold becomes testing data
    x.test.red <- x_red[test.inds.red,]
    y.test.red <- y_red[test.inds.red]

    x.test.white <- x_white[test.inds.white,]
    y.test.white <- y_white[test.inds.white]

    #Training Data
    x.train.red <- x_red[-test.inds.red,]
    y.train.red <- y_red[-test.inds.red]

    x.train.white <- x_white[-test.inds.white,]
    y.train.white <-y_white[-test.inds.white]

    #Model Creation
    temp_mod_red = lm(y.train.red ~ .,
                      data = data.frame(y.train.red, x.train.red))
    temp_mod_white = lm(y.train.white ~ .,
                        data = data.frame(y.train.white, x.train.white))

    #Storing the betas from each model
    mult_betas_red[k,] = coef(temp_mod_red)
    mult_betas_white[k,] = coef(temp_mod_white)

    #Making Predictions
    mult_preds_red = predict(temp_mod_red,
                              newdata = as.data.frame(x.test.red))
    mult_preds_white = predict(temp_mod_white,
                                newdata = as.data.frame(x.test.white))

    #Finding Errors
    red_reg_errs[k] = mean((y.test.red - mult_preds_red)^2)
    white_reg_errs[k] = mean((y.test.white - mult_preds_white)^2)
}

#CV errors
mean(red_reg_errs)
mean(white_reg_errs)

#Average values of each coefficient
#Red
for(j in 1:ncol(mult_betas_red)){
  print(mean(mult_betas_red[,j]))
}
#White
for(j in 1:ncol(mult_betas_white)){
  print(mean(mult_betas_white[,j]))
}
```

Lasso

```r
#Tuning the LASSO models using CV-within-CV
set.seed(12)
```

```r
#Some variables to store our errors and lambdas
K = 10
n_white = nrow(white_wine)
n_red = nrow(red_wine)

white_lasso_errs <- rep(0, K)
white_lasso_lambdas = rep(0, K)

red_lasso_errs <- rep(0, K)
red_lasso_lambdas = rep(0,K)

red_lasso_coefs = matrix(0, nrow = K, ncol = ncol(red_wine))
white_lasso_coefs = matrix(0, nrow = K, ncol = ncol(white_wine))

## CV-within-CV
for(k in 1:K){

    #Splitting Data
    test.inds.red <- (floor((k - 1) * n_red / K) + 1) :
      ceiling(k * n_red / K)
    test.inds.white <- (floor((k - 1) * n_white / K) + 1) :
      ceiling(k * n_white / K)

    #Kth fold becomes testing data
    x.test.red <- x_red[test.inds.red,]
    y.test.red <- y_red[test.inds.red]

    x.test.white <- x_white[test.inds.white,]
    y.test.white <- y_white[test.inds.white]

    #Training Data
    x.train.red <- x_red[-test.inds.red,]
    y.train.red <- y_red[-test.inds.red]

    x.train.white <- x_white[-test.inds.white,]
    y.train.white <-y_white[-test.inds.white]

    #Tuning lambda with cv.glmnet
    lasso_cv_red <- cv.glmnet(x.train.red,
                              y.train.red,
                              alpha = 1,
                              nfolds = 10)
    lasso_k_red <- lasso_cv_red$glmnet.fit

    lasso_cv_white <- cv.glmnet(x.train.white,
                                y.train.white,
                                alpha = 1,
                                nfolds = 10)
    lasso_k_white <- lasso_cv_white

    #Making predictions on test set with tuned lambda
    lasso_preds_red <- predict(lasso_k_red,
                               newx = x.test.red,
                              s = lasso_cv_red$lambda.min)

    lasso_preds_white <- predict(lasso_k_white,
                                newx = x.test.white,
                                s = lasso_cv_white$lambda.min)

    #Error rate on test data, best lambda, and
    #coefficients of best model
    red_lasso_errs[k] <- mean((y.test.red - lasso_preds_red)^2)
    red_lasso_lambdas[k] = lasso_cv_red$lambda.min
    red_lasso_coefs[k,] = as.vector(coef(lasso_cv_red$glmnet.fit,
                              s = lasso_cv_red$lambda.min))

    white_lasso_errs[k] = mean((y.test.white - lasso_preds_white)^2)
    white_lasso_lambdas[k] = lasso_cv_white$lambda.min
    white_lasso_coefs[k,] = as.vector(coef(lasso_cv_white$glmnet.fit,
                                          s = lasso_cv_white$lambda.min))

#Progress Bar
    print(k*10)
}

#Red Errors/Lambdas
mean(red_lasso_errs)
mean(red_lasso_lambdas)

#White Errors/Lambdas
mean(white_lasso_errs)
mean(white_lasso_lambdas)

#Average Coefficient Values
print("Red Coefficients")
for(j in 1:ncol(red_lasso_coefs)){
  print(mean(red_lasso_coefs[,j]))
}

print("White Coefficients")
for(j in 1:ncol(white_lasso_coefs)){
  print(mean(white_lasso_coefs[,j]))
}
```

PCR

```r
#Creating the PCR Models
set.seed(12)

#Same variables as before (cuz apparently I need that???)
K = 10
n_white = nrow(white_wine)
n_red = nrow(red_wine)

#Creating a couple more variables
```

```r
red_pcr_errs <- rep(0, K)
red_pcr_Ms = rep(0, K)

white_pcr_errs <- rep(0, K)
white_pcr_Ms <- rep(0, K)

## CV-within-CV
for(k in 1:K){
    #Splitting Data
    test.inds.red <- (floor((k - 1) * n_red / K) + 1) :
      ceiling(k * n_red / K)
    test.inds.white <- (floor((k - 1) * n_white / K) + 1) :
      ceiling(k * n_white / K)

    #Kth fold becomes testing data
    x.test.red <- x_red[test.inds.red,]
    y.test.red <- y_red[test.inds.red]

    x.test.white <- x_white[test.inds.white,]
    y.test.white <- y_white[test.inds.white]

    #Training Data
    x.train.red <- x_red[-test.inds.red,]
    y.train.red <- y_red[-test.inds.red]

    x.train.white <- x_white[-test.inds.white,]
    y.train.white <-y_white[-test.inds.white]

    #Tuning PCR
    pcr_k_red <-  pcr(y.train.red ~ x.train.red,
                  validation = "CV",
                  ncomp = 11,
                  segments = 10,
                  scale = T)
    pcr_k_white <- pcr(y.train.white ~ x.train.white,
                       validation = "CV",
                       ncomp = 11,
                       segments = 10,
                       scale = T)

    pcr_msep_red <- MSEP(pcr_k_red, intercept = F)
    pcr.ncomp.min.red <- which.min(pcr_msep_red$val[1,1,])

    pcr_msep_white <- MSEP(pcr_k_white, intercept = F)
    pcr.ncomp.min.white <- which.min(pcr_msep_red$val[1,1,])

    #Making Predictions on test set again
    red_pcr_preds <- predict(pcr_k_red,
                             newdata = x.test.red,
                             ncomp = pcr.ncomp.min.red)

    white_pcr_preds <- predict(pcr_k_white,
                               newdata = x.test.white,
                                ncomp = pcr.ncomp.min.white)

    #This makes predictions with tuned number of components

    #Error Rate and M
    red_pcr_errs[k] <- mean((y.test.red - red_pcr_preds)^2)
    red_pcr_Ms[k] = pcr.ncomp.min.red

    white_pcr_errs[k] = mean((y.test.white - white_pcr_preds)^2)
    white_pcr_Ms[k] = pcr.ncomp.min.white
    #Crude Progress Bar
    print(k*10)
}

mean(red_pcr_errs)
mean(red_pcr_Ms)

mean(white_pcr_errs)
mean(white_pcr_Ms)
```

PLS

```
#Creating the PLS Models
set.seed(12)

#Same variables as before (cuz apparently I need that???)
K = 10
n_white = nrow(white_wine)
n_red = nrow(red_wine)

#Creating a couple more variables
red_pls_errs <- rep(0, K)
red_pls_Ms = rep(0, K)

white_pls_errs <- rep(0, K)
white_pls_Ms <- rep(0, K)

## CV-within-CV
for(k in 1:K){
    #Splitting Data
    test.inds.red <- (floor((k - 1) * n_red / K) + 1) :
      ceiling(k * n_red / K)
    test.inds.white <- (floor((k - 1) * n_white / K) + 1) :
      ceiling(k * n_white / K)

    #Kth fold becomes testing data
    x.test.red <- x_red[test.inds.red,]
    y.test.red <- y_red[test.inds.red]

    x.test.white <- x_white[test.inds.white,]
    y.test.white <- y_white[test.inds.white]
```

```r
    #Training Data
    x.train.red <- x_red[-test.inds.red,]
    y.train.red <- y_red[-test.inds.red]

    x.train.white <- x_white[-test.inds.white,]
    y.train.white <-y_white[-test.inds.white]

    #Tuning PLS
    pls_k_red <-  plsr(y.train.red ~ x.train.red,
                  validation = "CV",
                  ncomp = 11,
                  segments = 10,
                  scale = T)
    pls_k_white <- plsr(y.train.white ~ x.train.white,
                       validation = "CV",
                       ncomp = 11,
                       segments = 10,
                       scale = T)

    pls_msep_red <- MSEP(pls_k_red, intercept = F)
    pls.ncomp.min.red <- which.min(pls_msep_red$val[1,1,])

    pls_msep_white <- MSEP(pls_k_white, intercept = F)
    pls.ncomp.min.white <- which.min(pls_msep_red$val[1,1,])

    #Making Predictions on test set again
    red_pls_preds <- predict(pls_k_red,
                             newdata = x.test.red,
                             ncomp = pls.ncomp.min.red)

    white_pls_preds <- predict(pls_k_white,
                               newdata = x.test.white,
                               ncomp = pls.ncomp.min.white)

    #This makes predictions with tuned number of components

    #Error Rate and M
    red_pls_errs[k] <- mean((y.test.red - red_pls_preds)^2)
    red_pls_Ms[k] = pls.ncomp.min.red

    white_pls_errs[k] = mean((y.test.white - white_pls_preds)^2)
    white_pls_Ms[k] = pls.ncomp.min.white
    #Crude Progress Bar
    print(k*10)
}

mean(red_pls_errs)
mean(red_pls_Ms)

mean(white_pls_errs)
mean(white_pls_Ms)
```

Ridge

```
#Tuning the Ridge models using CV-within-CV
set.seed(9)

#Some variables to store our errors and lambdas
K = 10
n_white = nrow(white_wine)
n_red = nrow(red_wine)

white_ridge_errs <- rep(0, K)
white_ridge_lambdas = rep(0, K)

red_ridge_errs <- rep(0, K)
red_ridge_lambdas = rep(0,K)

red_ridge_coefs = matrix(0, nrow = K, ncol = ncol(red_wine))
white_ridge_coefs = matrix(0, nrow = K, ncol = ncol(white_wine))

## CV-within-CV
for(k in 1:K){

    #Splitting Data
    test.inds.red <- (floor((k - 1) * n_red / K) + 1) :
      ceiling(k * n_red / K)
    test.inds.white <- (floor((k - 1) * n_white / K) + 1) :
      ceiling(k * n_white / K)

    #Kth fold becomes testing data
    x.test.red <- x_red[test.inds.red,]
    y.test.red <- y_red[test.inds.red]

    x.test.white <- x_white[test.inds.white,]
    y.test.white <- y_white[test.inds.white]

    #Training Data
    x.train.red <- x_red[-test.inds.red,]
    y.train.red <- y_red[-test.inds.red]

    x.train.white <- x_white[-test.inds.white,]
    y.train.white <-y_white[-test.inds.white]

    #Tuning lambda with cv.glmnet
    ridge_cv_red <- cv.glmnet(x.train.red,
                              y.train.red,
                              alpha = 0,
                              nfolds = 10)
    ridge_k_red <- ridge_cv_red$glmnet.fit

    ridge_cv_white <- cv.glmnet(x.train.white,
                                y.train.white,
                                alpha = 0,
                                nfolds = 10)
```

```r
    ridge_k_white <- ridge_cv_white

    #Making predictions on test set with tuned lambda
    ridge_preds_red <- predict(ridge_k_red,
                               newx = x.test.red,
                               s = ridge_cv_red$lambda.min)

    ridge_preds_white <- predict(ridge_k_white,
                                 newx = x.test.white,
                                 s = ridge_cv_white$lambda.min)

    #Error rate on test data, best lambda, and
    #coefficients of best model
    red_ridge_errs[k] <- mean((y.test.red - ridge_preds_red)^2)
    red_ridge_lambdas[k] = ridge_cv_red$lambda.min
    red_ridge_coefs[k,] = as.vector(coef(ridge_cv_red$glmnet.fit,
                               s = ridge_cv_red$lambda.min))

    white_ridge_errs[k] = mean((y.test.white - ridge_preds_white)^2)
    white_ridge_lambdas[k] = ridge_cv_white$lambda.min
    white_ridge_coefs[k,] = as.vector(coef(ridge_cv_white$glmnet.fit,
                                     s = ridge_cv_white$lambda.min))

#Progress Bar
    print(k*10)
}

#Red wine errors/lambdas
mean(red_ridge_errs)
mean(red_ridge_lambdas)

#White wine errors/lambdas
mean(white_ridge_errs)
mean(white_ridge_lambdas)

#Average Coefficient Values
print("Red Coefficients")
for(j in 1:ncol(red_ridge_coefs)){
  print(mean(red_ridge_coefs[,j]))
}

print("White Coefficients")
for(j in 1:ncol(white_ridge_coefs)){
  print(mean(white_ridge_coefs[,j]))
}
```

Table Generation

```r
#Creating table 1, wihch looks at average CV errors
#Making a data frame for GT to use
df1 <- data.frame(
  Row_Name = c("Red", "White"),
  "Multiple Regression" = c(0.4359, 0.5760),
  "PCR" = c(0.4346, 0.5781),
  "PLS" = c(0.4348, 0.5775),
  "LASSO" = c(0.4363, 0.5766),
  "Ridge" = c(0.4344, 0.5759)
)

#Now creating the table
gt(df1) %>%
  cols_label(Row_Name = "Wine Type") %>%
  cols_align(align = "center") %>%
  tab_caption("Average CV Errors for Each Model Type")
```

```
#Now table 2, which gives the average coefficient value for each
#model type (red wine)

#Data frame
dfred <- data.frame(
  #Row names
  Row_Name = c("intercept", "fixed.acidity", "volatile.acidity", "citric.acid", "residual.sugar", "chlo
  #Actual Values now
  "Multiple Regression" = c(21.893, 0.024, -1.081, -0.184, 0.016,
                            -1.879, 0.004, -0.003, -17.782,
                            -0.422, 0.926, 0.276),
  "Lasso" = c(13.663, 0.015, -1.055, -0.104, 0.009, -1.774, 0.003,
              -0.003, -9.593, -0.388, 0.876, 0.280),
  "Ridge" = c(31.894, 0.030, -1.017, -0.091, 0.019, -1.816, 0.004,
              -0.003, -28.033, -0.319, 0.898, 0.254)
)

#Table
gt(dfred) %>%
  cols_label(Row_Name = "Variable Name") %>%
  cols_align(align = "center") %>%
  tab_caption("Average Variable Coefficient Values for Red Wine")
```

```
#Table 3, which gives the average coefficient value for white wine
dfwhite <- data.frame(
  #Row names
  Row_Name = c("intercept", "fixed.acidity", "volatile.acidity", "citric.acid", "residual.sugar", "chlo
  #Actual Values now
  "Multiple Regression" = c(152.471, 0.067, -1.863, 0.021, 0.082,
                            -0.235, 0.004, -0.0002, -152.581, 0.691,
                            0.636, 0.191),
  "Lasso" = c(110.804, 0.028, -1.872, 0.015, 0.065, -0.405, 0.004,
              -0.0002, -110.104, 0.505, 0.544, 0.235),
  "Ridge" = c(68.537, -0.004, -1.795, 0.021, 0.046, -1.029, 0.004,
              -0.0006, -67.031, 0.365, 0.494, 0.264)
)

#Table
gt(dfwhite) %>%
  cols_label(Row_Name = "Variable Name") %>%
  cols_align(align = "center") %>%
  tab_caption("Average Variable Coefficient Values for White Wine")
```

References

Dataset: https://archive.ics.uci.edu/dataset/186/wine+quality

Reference Paper:

Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, José Reis,

Modeling wine preferences by data mining from physicochemical properties,

Decision Support Systems,

Volume 47, Issue 4,

2009,

Pages 547-553,

ISSN 0167-9236,

https://doi.org/10.1016/j.dss.2009.05.016.

(https://www.sciencedirect.com/science/article/pii/S0167923609001377)