ST563 Final Project

Himanish Ganjoo, Hunter Jiang, Liqiang Hou, and Yu Jiang

North Carolina State University

1 INTRODUCTION

1.1 Purpose

For this project, we will be analyzing the wine quality data[1] collected from the Vinho Verde region in Portugal.

We are going to use the quality as the response and others as predictors to fit several models to see which predictors are related to the response, that is, how the quality is affected by other factors. The model part is divided into three section: regression methods, classification methods and unsupervised methods. More details will be discussed later.

1.2 Data Summarizations

Wine quality dataset consists of 1599 samples related to physicochemical and sensory properties of wine. Those variables contain fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol and the target **Quality**. The histograms of these variables are shown below.

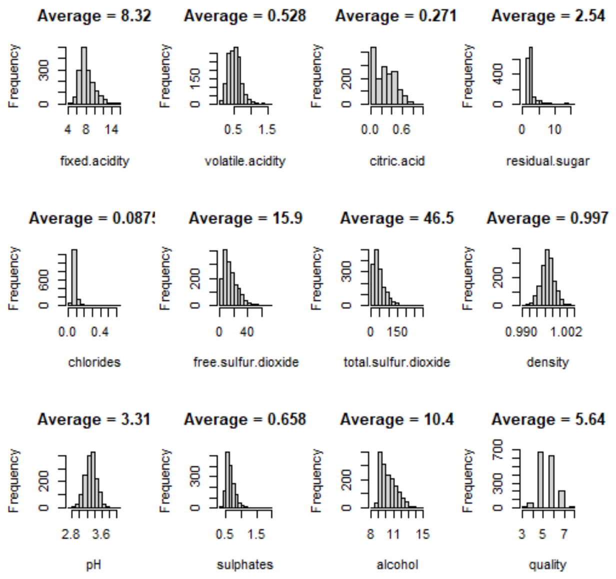


Figure 1. Histograms of Variables.

From the histograms above, **quality** is discrete variable rating from 1 (poor) to 10 (excellent). The goal is to predict quality based on 11 analytic continuous variables. We apply 6 regression methods, 7 classification methods, and a clustering method to analysis it.

Before fitting models, we have checked the original data set to see whether there is any missing data. After checking, there is no missing data and we can split the data set into the train set and the test set and then fit the data to the train set and evaluate the model to the test set. Also, through this paper, if there is no specific statement, we split the data into training and testing sets. Then find the best parameter using the 10-fold CV: We split the 1599 samples into ten parts, and using each one of them as testing test after training models using remaining nine parts of data.

[1]The data is available at https://archive.ics.uci.edu/ml/datasets/Wine+Quality

```
# Set seed
set.seed(563)

# Slice the data, 80% to the train and 20% to the test
train <- sample(1:nrow(red), size = nrow(red) * 0.8)
test <- dplyr::setdiff(1:nrow(red), train)

TrainDat <- red[train,]
TestDat <- red[test, ]
```

2 REGRESSION METHODS

In statistical modelling, regression analysis is a set of statistical processes for estimating the relationships between a dependent variable (often called the 'outcome variable') and one or more independent variables (often called 'predictors', 'covariates', or 'features'). [2]

We can make a conclusion that regression analysis is a way of mathematically figuring out which of those predictors does significantly have an influence. It can be used in statistics to find a tendency in the development of data and can also provide an equation for a graph so that we can make predictions about the data.

For the regression part, we are going to use the R machine learning **caret** package to build some common regression models. The beauty of this package is that it is well optimized and can handle maximum exceptions to make our job simple, we just need to call functions for implementing algorithms with the right parameters and change the method part.

Caret package provides **train()** method for training our data for various algorithms. We just need to pass different parameter values for different algorithms. Before **train()**, we will first use **trainControl()** method to control the computational nuances of the **train()** method.

```
# Use the trainControl() method
trctrl <- trainControl(method = "repeatedcv",
number = 10, repeats = 1)
```

Later, we can fit these regression models with different arguments **method = "..."** and use the multiple linear regression as an example to demonstrate how to fit a model with **caret** in detail and all the results of these models will be shown in the **Results** section.

2.1 Multiple Linear Regression

It is the simplest form of regression and it has assumed that the relationship between the predictors and the response is linear and the general equation is

$$Y = \beta_0 + \beta_1 X_1 + \beta_3 X_3 + ... + \beta_K X_K + \varepsilon,$$

Where Y is the reponse, β_js are correspond partial-slope to each predictor, and ε follows $N(0, \sigma^2)$.

After using the **trainControl()** method, we can train the multiple linear regression model with **method = "glm-StepAIC"**.

```
# Fit the MLR model
mlr_fit <- train(quality~., data = TrainDat, method = "glmStepAIC",
trControl = trctrl,  preProcess = c("center", "scale"))

# Trained MLR model result
mlr_fit
mlr_fit$finalModel
```

After fitting this multiple linear regression model, we are going to fit the model to the test set to evaluate it.

```
# Test set prediction for fit
mlr_pred <- predict(mlr_fit, newdata = TestDat)
```

[2]Regression Analysis, Wikipedia (https://en.wikipedia.org/wiki/Regression_analysis)

```
# MSE of the model mlr_fit
mean((mlr_pred - TestDat$quality)^2)
```

These code chunks above can be also applie to other regression models and the only thing needs to be changed is **method = "..."**. In the proceeding models, we will only mention the argument **method = "..."** for every models.

See Appendix for the entire and detialed R code.

2.2 Ridge Regression

In the ridge regression, we add a constraint on the sum of squares of the regression coefficients. Our objective is to minimize

$$\sum_{i=1}^{n}(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij})^2+\lambda\sum_{j=1}^{p}\beta_j^2=RSS+\lambda\sum_{j=1}^{p}\beta_j^2,$$

For the objective function, a ridge regression model looks for estimates to fit the data well by making RSS as small as possible but the second term $\lambda\sum_{j=1}^{p}\beta_j^2$, called a shrinkage penalty, which is small when $\beta_j=0, j=1,..,p$ are close to zero and thus it can shrink the estimates of β_j towards zero. The tuning parameter $\lambda\geq 0$ i needs to be chosen to control the relative impact of these two terms.

We have adopted the augment **method = "glmnet"** for the ridge regression.

2.3 Lasso Regression

Lasso stands for Least Absolute Shrinkage and Selection Operator, and the goal of it is to minimize the objective function

$$\sum_{i=1}^{n}(y_i-\beta_0-\sum_{i=1}^{p}\beta_j x_{ij})^2+\lambda\sum_{i=1}^{p}|\beta_j|=RSS+\lambda\sum_{i=1}^{p}|\beta_j|$$

After comparing the objective function of lasso regression with the previous one, ridge regression, it is evident that there is something in common between these two objectives.

The only difference is that we have used $|\beta_j|$ to replace β_j^2. It has also shown that λ is regularization parameter and the intercept term is not regularized. For the lass regression, we do not need to assume that the error terms are normally distributed and it needs standardization.

We have adopted the augment **method = "lasso"**.

2.4 Elastic Net Regression

Elastic Net regression is preferred over both ridge and lasso regression when one is dealing with highly correlated independent variable and the goal is to minimize the objective function

$$\sum\varepsilon^2+\lambda_1\sum\beta^2+\lambda_2\sum|\beta|=\sum[y-(\beta_1+\beta_2X_2+\beta_3X_3+...+\beta_KX_K)]^2\lambda_1\sum\beta^2+\lambda_2\sum|\beta|$$

Also, like the ridge and the lasso regression, elastic net regression does not need to be assumed normality.

We have used **method = "enet"**.

2.5 Principal Components Regression

PCR is a regression technique which is widely used when you have many independent variables OR multicollinearity exist in the data. It is divided into 2 steps: getting the Principal components and run regression analysis on principal components and we can use a graphic to demonstrate it. For the Principal Components Regression,

We have adopted **method = "pcr"** and plot the result.

2.6 Partial Least Squares Regression

PLS is an alternative method of PCR when there exist highly correlated predictors and it also makes sense when there are a great number of predictors. The Figure 2 is about different steps between PLS and PCR as well as the results.

We have adopted **method = "pls"** and plot the result.

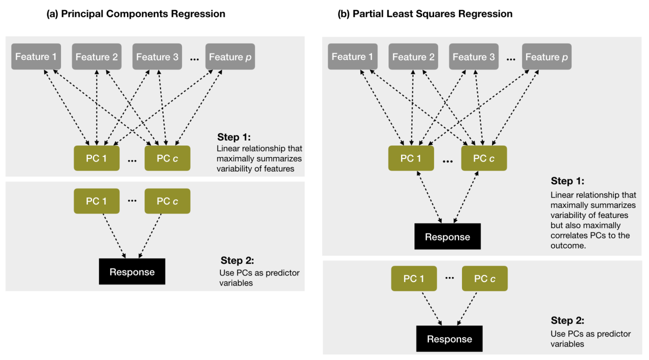
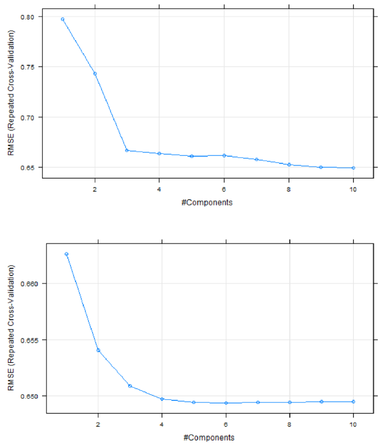

(a) Principal Components Regression
Feature 1
Feature 2
Feature 3
Feature p
PC 1
PC c
Step 1: Linear relationship that maximally summarizes variability of features
Response
Step 2: Use PCs as predictor variables
(b) Partial Least Squares Regression
Step 1: Linear relationship that maximally summarizes variability of features but also maximally correlates PCs to the outcome.
Step 2: Use PCs as predictor variables


Figure 2. Difference between PCR and PLS

2.7 Results

After fitting these models, we can obtain the optimal pamarater for each model.

By comparing criteria values for these six regression models above, we can make a conclusion that the MSE of the multiple linear regression is the smallest among them.

Table 1. Trained Model Results for Regression Methods

Method	Parameters	MSE	RMSE	R^2	MAE
MLR	-	**0.428**	0.651	0.346	**0.507**
Lasso	$f = 0.9$	0.429	0.653	0.341	0.511
Ridge	$\alpha = 0.10, \lambda = 0.0075$	0.431	0.668	0.329	0.538
EN	$f = 1,\ \lambda = 0.1$	0.429	**0.649**	**0.355**	0.506
PCR	$n = 10$	0.432	**0.649**	0.349	**0.507**
PLS	$n = 6$	0.432	0.650	0.347	0.508

f: fraction;

n: the number of components

Therefore, we can adopt the MLR as the optimal regression one for further prediction.

Optimal Regression Model :

$$\begin{aligned}\hat{Y} = \quad & 5.642 - 0.178\text{volatile.acidity} - 0.112\text{chlorides} + 0.067\text{free.sulfur.dioxide} \\ & -0.116\text{total.sulfur.dioxide} - 0.071\text{pH} + 0.160\text{sulphates} + 0.293\text{alchohol}.\end{aligned}$$

Some discussions:

1. Both Lasso and Elastic Net choose a fraction that is near to 1: all variables should be included in the model. From the result of Ridge regression, we can find out that the collinearilty in the data set is not strong.
2. We choose MLR, an unbiased estimator, as our final regression model here, which definitely make sense because we tend to choose a full model for the wine dataset.
3. We use MSE and MAE as main performance matrics to select an best model, while others also perform relatively well on this dataset. The best model on RMSE and R-Square statistics are not MLR.
4. From the optimal model above, we can see that the change of the predictor - alchohol makes the most sense to the response since it has the largest coefficient, 0.293, if we control all other predictors the same.

3 CLASSIFICATION METHODS

Classification methods want to predict a qualitative response, instead of a quantitative one, for the collected data. Mathematically speaking, classification predictive modelling is the task of approximating a mapping function(f) from predictors(X) to discrete response variables(y). In this section, we introduce several classifiers, the way to make them work on a regression dataset, and how to deal with the unbalancedness.

When working on a dataset $T = \{(x_i, y_i)\}_{i=1}^n$, we want to find a classification model who maximizes the accuracy rate (or other performance metrics):

$$\max \sum_{i=1}^{n} I(\hat{y}_i, y_i), \text{ s.t. } f_{\hat{y}_i}(x_i) > 0, \; f_j(x_i) < 0, \; j \neq \hat{y}_i.$$

Here $I(a,b)$ denotes the identical function which returns 1 if $a = b$, and 0 otherwise; $f_j(x)$ is the predictive domain decided by class j's decision boundary. Note that here we use the idea of One vs All(1VA) process when the number of class is larger than 2.

3.1 Descrption

We will introduce two kinds of discriminant analysis, the logistic regression, k-nearest neighbour, and three kinds of tree-based model. All these training methods should require independent sample data.

3.1.1 Discriminat Analysis

The Linear Discriminant Analysis (LDA) and the Quadratic Discriminant Analysis (QDA) both assume that the k-th class cames from a multivariate Gaussian distribution $N(\mu_k, \Sigma_k)$. Using Bayes rules, a LDA model assigns an observation $X = x$ to the class j which has the largest

$$\delta_j(x) = x^T \Sigma^{-1} \mu_j - \frac{1}{2} \mu_j^T \Sigma^{-1} \mu_j + log \pi_j.$$

Note that in the LDA k Gaussian distribution share the same covariance matrix, so the sub k is emitted in the expression.

Assuming a quadratic decision boundary and an unequal covariance, the QDA model find the $\sigma_j(x)$ that is the largest among

$$\delta_j(x) = -\frac{1}{2}(x - \mu_j)\Sigma_j^{-1}(x - \mu_j) + x^T \Sigma_j^{-1} \mu_j - \frac{1}{2} \mu_j^T \Sigma_j^{-1} \mu_j + log \pi_j - \frac{1}{2} log |\Sigma_j|.$$

In these models, we need to estimate Σ_j, μ_j, and π_j before plugging them into the correspond formula. The decision domain for the class j is the set where $\delta_j(x)$ is the largest among $\{1, 2, ..., c\}$.

3.1.2 Logistic Regression

Based on the assumption that there is little or no multicollinearity among predictors, Logistic Regression (LR) is a binary classifier which maximizes the likelihood function:

$$\ell(\beta) = \prod_{i:y_i=1} p(x_i, \beta) \prod_{i':y_{i'}=1} [1 - p(x'_i, \beta)], \text{ where } p(x, \beta) = \frac{e^{\beta_0 + \beta_1 x_1 + ... + \beta_p x_p}}{1 + e^{\beta_0 + \beta_1 x_1 + ... + \beta_p x_p}}$$

Function $p(x, \beta)$ is the so-called *logistic function*, which always gives the outputs between 0 and 1 for all values of x. The decision bounday satisfies $p(X, \hat{\beta}) = 0.5$, and the decision domain for the positive class is $D = \{x | p(x, \hat{\beta}) - 0.5 > 0\}$.

3.1.3 k-Nearest Neighbor

To apply k-Nearest Neighbor (kNN) on our data, we need to introduce a measurement of the distance first, and Euclidean Distance (ED) is a popular choice in the Euclidean space. Then, kNN assign observation i to class j which is the mode of its k nearest neighbor $n(X)$.

As discussed, all the points from the class j can be expressed as the set of point which satisfies $D_j = \{X | \text{mode of } n(X) \text{ is } j\}$. Note that the decision boundary for each class j is non-linear here.

3.1.4 Tree-based Methods
We use three different ways to classify our wine dataset: Decision Tree, Bagging, and Random Forest.

The Optimal Decision Tree (ODT) is a two-step method. We record the full path of growing a classification tree. In each path, we divide the predictor space one more time, finally into c distinct and non-overlapping regions, based on a measure of total variance across the c class such as Gini index (G) and cross-entropy (D) given by

$$G = \sum_{i=1}^{c} \hat{p}_{mc}(1-\hat{p}_{mc}),$$
$$D = -\sum_{i=1}^{c} \hat{p}_{mc} log \hat{p}_{mc}.$$

Note that we will have many subtrees for the full Decision Tree. The second step penalizes each of them for its complexity, and claim the optimal one that minimizes the loss function. Note that tree-based methods are nonlinear to the predictors.

The bagging (BAG) procedure is an aggregation of several Decision Trees based on the bootstrap resampled dataset T_k^*. We first train n trees based on resampled data and full variable list from 1 to p, then assign the mode of individual predictions as the final result. Such an approach inherits the advantages of bootstrap sampling, which has smaller standard deviations when doing predictions.

The Random Forest (RF) method is an improvement over BG by way of a small decorrelates the trees. Instead of using all predictors, RF only choose m (normally $m = \sqrt{p}$) of them in each split. That is, at each split in the tree, the algorithm is not even allowed to consider a majority of the available predictors.

3.2 Two class and Beyond
Most classification methods are originally designed for binomial analysis, but the multi-class classification is a very common task in the real life. Use our case study dataset as an example, where the response variable of interest is the quality of the sample wine, we definitely can collapse the ratings into a low/high level so that we can apply all methods on it. However, we do not have any theoretical or practical reason for doing that. How about collapsing the rating into low/medium/high? Or ideally, can we directly work on the original data in the classification settings?

The answer to these questions is yes. From real-life experiences, a model that can predict the wine quality as low/medium/high seems “superior” to the binary classifier with outputs between low/high. If that is the case, then why should not we do the classification problem by all possible outcomes? For those binary classifiers, the way of doing this is the procedure called One vs All that trains c models, instead of 1, to divide class i and other classes. After treating overlapping among decision domains, the final response will be unique. For datasets like wine quality, another way to do this is using ordinal regression methods, which trains $c-1$ models to classify class i and $i+1$.

However, owing to the availability of the data, we cannot always fit classification models on all possible outcomes. For example, QDA will report an error of “some group is too small” if we do not collapse quality. Moreover, normally we spend more time when we have more classes, with inferior results. Thus, such a problem is actually a trade-off between the number of classes and model performance (accuracy&time). In our numerical experiment, we will use 2 and 3 classes as examples.

3.3 Imbalanced Dataset
When collapsing the quality rating, it is not hard to find that the distribution of the wine quality is unbalanced: we only have a small amount of sample in the “high” quality, which totally makes sense in the real life. Mathematically speaking, we are facing an imbalanced dataset.

Classification results may be skewed, and the performance metrics may disguise when working with imbalanced data. There are several ways to deal with this, and we choose Over-/Under- Sampling technics in this case study.

The result of over-/under-sampling is the creation of a balanced dataset that the former procedure makes the number of observations in each class equal to the class with maximum observations, and the latter forces each class only have a size which is equal to the minimal number of observations. In our example, we use random sampling technics to implement such an idea, and our code is written based on the “sample()” function in R.

3.4 Results

We run this section of simulation in R environment, and we have 6 datasets drown from the wine quality data. The "2-Class" and the "3-Class" dataset are relabeled wine quality datasets with 2 and 3 class; the collapsing rule are Low$(1-5)$/High$(6-10)$ and Low$(1-5)$/Medium(6)/High$(7-10)$. Following over-/under-sampling procedure, we yield 4 datasets and name them by "ClassNumber-MethodAbbr", e.g. "2-Over" is the result of over-sampling on "2-Class" dataset.

Here are the settings for each model. For the kNN model, we choose k from $1,3,5,7$. The number of trees in tree-based models is fixed as 100; the RF model will use mtry=5 at each split. Finally, 7 model described above is compared by the performance in 10-fold CV. The performance metrics we use is accuracy defined by ACC = Number of correct predictions / Total number of predictions.

Table 2. Numerical Results for Classification Methods (Best performance is in **bold** font)

	LDA		QDA		LR		kNN	
	ACC(SD)	TIME	ACC(SD)	TIME	ACC(SD)	TIME	ACC(SD)	TIME
2-Class	0.74(0.03)	0.11	0.71(0.03)	0.14	0.74(0.03)	0.06	0.75(0.04)	**0.03**
2-Over	0.74(0.04)	**0.04**	0.72(0.02)	0.17	0.74(0.04)	0.04	0.73(0.04)	0.04
2-Under	0.73(0.04)	0.07	0.72(0.02)	**0.02**	0.74(0.04)	0.04	0.71(0.04)	0.03
3-Class	0.62(0.04)	0.04	0.58(0.04)	0.08	0.52(0.05)	0.33	0.64(0.05)	**0.00**
3-Over	0.58(0.04)	0.08	0.54(0.05)	0.07	0.58(0.05)	0.46	0.63(0.04)	**0.00**
3-Under	0.59(0.03)	0.03	0.52(0.04)	0.03	0.59(0.03)	0.14	0.54(0.03)	**0.02**
	ODT		BG		RF		**BEST**	
	ACC(SD)	TIME	ACC(SD)	TIME	ACC(SD)	TIME	ACC	TIME
2-Class	0.68(0.05)	0.89	0.79(0.02)	1.78	**0.79(0.02)**	1.77	0.79(**RF**)	0.03(**kNN**)
2-Over	0.71(0.03)	0.83	**0.79(0.02)**	1.90	0.78(0.02)	1.71	0.79(**BG**)	0.04(**LDA**)
2-Under	0.71(0.04)	0.84	0.76(0.04)	1.58	**0.77(0.04)**	1.58	0.77(**RF**)	0.02(**QDA**)
3-Class	0.59(0.05)	0.91	0.69(0.04)	2.04	**0.69(0.04)**	1.83	0.69(**RF**)	0.00(**kNN**)
3-Over	0.57(0.04)	1.09	0.67(0.05)	2.64	**0.68(0.05)**	2.46	0.68(**RF**)	0.00(**kNN**)
3-Under	0.55(0.06)	0.60	0.60(0.04)	0.71	**0.61(0.03)**	0.63	0.61(**RF**)	0.02(**kNN**)

Here is some discussion about the result of the wine quality dataset, which has 10 predictors and about 1599 observations.

1. Tree-based methods, especially the random forest, perform very well among 6 datasets while focusing accuracy, but the gap between RF/BG and others are not large enough to declare they "outperform" others. From the RF model, we find that *alcohol*, *sulphates*, and *volatile.acifity* are three most important variables in the classification settings.
2. When training/predicting time is of vital importance, then LDA/QDA/kNN are preferred choices. Note that we have a relatively small dataset with few predictors, and believe this is the reason why kNNs dominate others in such a scenario.
3. In general, 3-Class predictions' accuracy is higher than 2-Classes', and the difference varies from $10-22$ percent. The running time varies among these models: LDA/QDA/kNN runs faster when collapsing the dataset into 3 classes. However, Logistic Regression/Tree-Based Models are slower in 3-Class case.
4. In this dataset, applying over-sampling technics do not have significant performance differences. Inspecting from the 3×3 confusion matrixes we can find out that even though we have more correct predictions to the "high" level, the false prediction of the "high" level also rises. In the wine dataset, balancing data by resampling, which does not add any information in the training set, does not improve the performance of these 7 methods.
5. The under-sampling worsens the prediction result by $5-10$ percent because it does not fully use the available data, which indicates that we still need more data about wines' quality on each category, especially those high-rated wines, to get better performance.

4 UNSUPERVISED METHODS

In this section, we explore the application of clustering methods on the dataset. *Clustering* techniques look for subgroups or *clusters* in the data based on the given features. These clusters are organized on the basis of similarity: observations within a cluster are similar to each other. We try to form clusters among the wine data observations using K-means clustering.

4.1 Data Preparation

In order to identify a subset of the features to be used to find clusters, we first looked at the correlation matrix of the features. Figure 3shows this. For clustering, we will use only the variables: sugar, chlorides, total.sulfur.dioxide, density, sulphates and alcohol. The rest are correlated with these seven, and their presence is redundant.

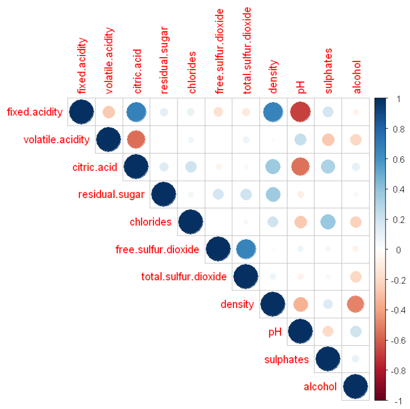


Figure 3. Correlation Matrix of the features represented as a graph

With these seven variables chosen, we perform a min-max scaling on the variables to *standardize* them, since all of them have different scales and units.

4.2 K-Means Clustering

K-Means is a clustering method that partitions observations into a finite number of non-overlapping groups. The number of groups, K, is specified. The problem of K-Means clustering amounts to assigning all observations to sets C_i such that,

$$C_1 \cup C_2 \cup \ldots C_N = \{1,\ldots,n\}, \;\; C_i \cap C_j = \emptyset \qquad \text{for} \qquad i \neq j$$

which means that every observation belongs to some cluster, and no observation belongs to more than one cluster. To optimize the assignment so that the most similar observations are clustered together, the method seeks to minimize the within-group variation measure W, which is defined as the sum of all pairwise Euclidean distances between the observations assigned to a cluster, divided by the number of observations in the cluster:

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2.$$

4.3 Results

Since the number of clusters has to be pre-specified in a K-Means clustering problem, we will run over an array of different values of K to determine the optimal value using the within-cluster sum of squares (WSS) in each case. The figure 4shows a plot of K versus WSS, and we observe that the WSS value stops dropping significantly after $K = 5$. A lower value of K has a high value of WSS, and a higher value will lead to too many clusters, which might hamper interpretability. We will thus use $K = 5$ to do our clustering.

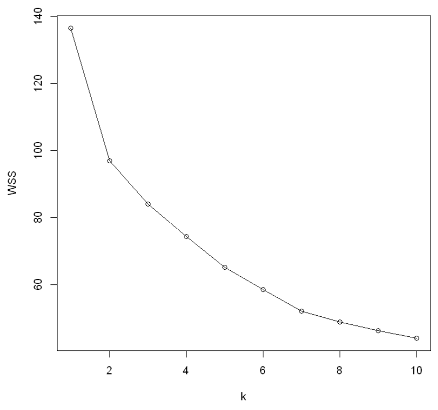


Figure 4. K vs WSS for different values of K.

After performing clustering, we make pairwise plots of 5 of the 7 variables chosen. We colour the points according to the cluster. Figure 5shows this.

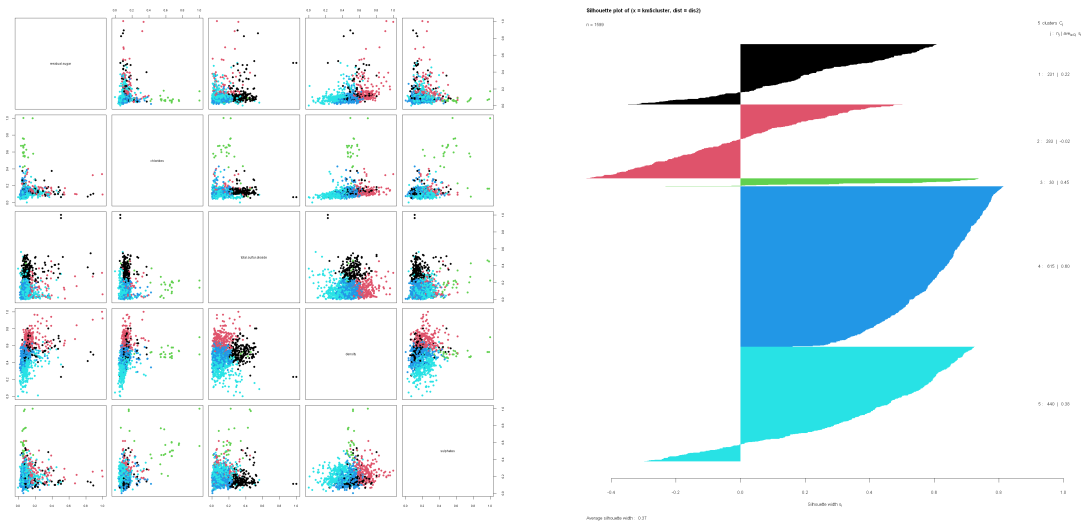


Figure 5. Pairwise plot of feature variables(left) and Silhoutte Analysis(right)

Based on Figure 5, we can draw a few conclusions about the clusters:

- The green cluster corresponds to a high value for the chlorides variable.
- The black cluster corresponds to high sulfur.dioxide value.
- The red and light blue clusters separate the high and low density wines.

Though not perfectly well-separated, the clusters do have some inferential value, as can be seen in the list of observations above. To quantify exactly how well-clustered the observations are, we will perform a *silhouette* analysis on the clusters.

4.3.1 Silhouette Analysis

A silhouette analysis is used to validate clustering and quantify how well-assigned the clusters are. Each point is assigned a silhouette value, which is a ratio of two measures: 1) *cohesion* or the closeness of a point to its own cluster members, and 2) *separation* or the similarity of a point to the points in other clusters. The silhouette value of each cluster is the average silhouette value of its member observations. One can then average over the silhouette values of all clusters to get a mean silhouette value for the clustering. A silhouette value ranges from -1 to 1, with a higher value indicating better clustering.

Assuming that observations have been assigned to clusters, we first calculate the cohesion of a point i belonging to cluster C_k:

$$a_i = \frac{1}{|C_k| - 1} \sum_{j \neq i, j \in C_k} d(i,j),$$

where $d(i,j)$ is the Euclidean distance between points i and j. The smaller the value of a, the better the assignment of the point to the cluster, as it indicates the mean distance of the point from its other cluster members. Similarly, we calculate the separation for the point:

$$b_i = \min_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d(i,j),$$

which is the minimum mean distance of the point to all members of another cluster. The silhouette value is defined as

$$s_i = \frac{b_i - a_i}{\max\{a_i, b_i\}}.$$

If a_i is small, it means the point is well-assigned to its own cluster, and if b_i is large, it means the point is well-separated from neighbouring clusters. Thus, if $a_i << b_i$, we have $s = 1$, which means perfect clustering.

Figure 5 shows the silhouette scores for all observations and groups graphically. From the graph, we can see the scores for each cluster.

Cluster #	Observations	Silhouette Score
1	231	0.22
2	283	-0.02
3	30	0.45
4	615	0.6
5	440	0.38

From these scores, we can see than clusters 3,4 and 5 (green, blue, light blue) are well-assigned. The overall average silhouette score is 0.38, which means the clustering has performed decently.

REFERENCES

Cortez, P., Cerdeira, A., Almeida, F., Matos, T., and Reis, J. (2009). Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553.

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An introduction to statistical learning*, volume 112. Springer.

Kuhn, M. et al. (2008). Building predictive models in r using the caret package. *Journal of statistical software*, 28(5):1–26.

Appendix

1 Liqiang's Code

1.1 Data Split and Collapse

As described in part I, we use 'sample' function in R to split wine data into training set and test set. For regression part, we build variables matrix X and quality for training set **quality.train** and test set **quality.test**.

```
set.seed(1)
mydata <- read.csv("winequality-red.csv", sep=";")
train = sample(1:nrow(mydata), nrow(mydata) * 0.8)
test = (1:nrow(mydata))[-train]

quality.train = mydata[train, ]$quality
quality.test = mydata[test, ]$quality
quality.all = mydata$quality
# Split X-Y
allX = subset(mydata, select = -quality)
trainX = subset(mydata[train, ], select = -quality)
testX = subset(mydata[test, ], select = -quality)
train.mat = model.matrix(quality~., data = mydata[train, ])
test.mat = model.matrix(quality~., data = mydata[test, ])

# Collapse quality
qualityLev.all = ifelse(quality.all > 6, "High",
ifelse(quality.all>5, "Medium", "Low"))
qualityLev.train = ifelse(quality.train > 6, "High",
ifelse(quality.train>5, "Medium", "Low"))
qualityLev.test = ifelse(quality.train > 6, "High",
ifelse(quality.train>4, "Medium", "Low"))
qualityLev.all = factor(qualityLev.all,
levels=c("Low", "Medium", "High"))
qualityLev.train = factor(qualityLev.train,
levels=c("Low", "Medium", "High"))
qualityLev.test = factor(qualityLev.test,
levels=c("Low", "Medium", "High"))

qualityLev.all = ifelse(quality.all > 5, "High", "Low")
qualityLev.train = ifelse(quality.train > 5, "High", "Low")
qualityLev.test = ifelse(quality.train > 5, "High", "Low")
qualityLev.all = factor(qualityLev.all,
levels=c("Low", "High"))
qualityLev.train = factor(qualityLev.train,
levels=c("Low", "High"))
qualityLev.test = factor(qualityLev.test,
levels=c("Low", "High"))
```

1.2 Regression

For the regression part, we are going to use the R machine learning **caret** package to build some common regression models. The beauty of this package is that it is well optimized and can handle maximum exceptions to make our job simple, we just need to call functions for implementing algorithms with the right parameters and change the method part.

Caret package provides **train()** method for training our data for various algorithms. We just need to pass different parameter values for different algorithms. Before **train()**, we will first use **trainControl()** method to control the computational nuances of the **train()** method.

```
# Use the trainControl() method
trctrl <- trainControl(method = "repeatedcv",
number = 10, repeats = 1)
```

After using the **trainControl()** method, we can train the multiple linear regression model with **method = "glmStepAIC"** as follow.

```
# Fit the MLR model
mlr_fit <- train(quality~., data = TrainDat, method = "glmStepAIC",
trControl = trctrl,  preProcess = c("center", "scale"))

# Trained MLR model result
mlr_fit
mlr_fit$finalModel
```

We are going to fit the model to the test set to evaluate the MLR after fitting it to the train set.

```
# Test set prediction for fit
mlr_pred <- predict(mlr_fit, newdata = TestDat)
# MSE of the model mlr_fit
mean((mlr_pred - TestDat$quality)^2)
```

For the lasso regression, we have adopted **method = "lasso"**.

```
# Fit the MLR model
mlr_fit <- train(quality~., data = TrainDat, method = "lasso",
trControl = trctrl,  preProcess = c("center", "scale"))
```

We also provide naive R code here, but whose result is not included in the tables and figures in this paper.

```
# Regression

# Methods: Linear Regression
md.lm = lm(quality~., data=mydata, subset = train)
lm.pred = predict(md.lm, newdata = mydata[test, ])
lm.mse = mean((lm.pred - quality.test)^2)
print(lm.mse)

# Methods: LR + Subset Selection
library(leaps)
regfit = regsubsets(quality~., mydata, nvmax = 11)
```

```
reg.summary = summary(regfit)

## Plot
par(mfrow=c(2,2))
plot(reg.summary$rss, xlab="N of variables")
points(11, reg.summary$rss[11], col="red", cex=2, pch=20)
plot(reg.summary$adjr2, xlab="N of variables")
points(8, reg.summary$adjr2[8], col="red", cex=2, pch=20)
plot(reg.summary$cp, xlab="N of variables")
points(7, reg.summary$cp[7], col="red", cex=2, pch=20)
plot(reg.summary$bic, xlab="N of variables")
points(6, reg.summary$bic[6], col="red", cex=2, pch=20)
par(mfrow=c(1,1))

val.errors = rep(NA, 11)
for(i in 1:11){
  coefi = coef(regfit, id=i)
  pred = test.mat[, names(coefi)] %*% coefi
  val.errors[i] = mean((pred-quality.test)^2)
}
which.min(val.errors)
ss.mse = val.errors[11]


# Methods: LR + Lasso (not good)
library(glmnet)
grid = 10^seq(4,-4, length=100)
cv.lasso = cv.glmnet(train.mat[, -1], quality.train, alpha=1)
plot(cv.lasso)
bestlam = cv.lasso$lambda.min
md.lasso = glmnet(train.mat[, -1],
quality.train, alpha = 1, lambda = grid)
lasso.pred = predict(md.lasso, s=bestlam, newx = test.mat[, -1])
lasso.mse = mean((lasso.pred - quality.test)^2)
print(lasso.mse)
lasso.coef = predict(md.lasso,
type = "coefficients", s=bestlam)[1:12, ]
print(lasso.coef)

# Methods: Principle Component Regression (not good)
library(pls)
md.pcr = pcr(quality~., data=mydata, subset=train,
scale=TRUE, validation="CV")
summary(md.pcr)
validationplot(md.pcr, val.type = "MSEP")
pcr.pred = predict(md.pcr, mydata[test,], ncomp=11)
mean((pcr.pred-quality.test)^2)

# Methods: PLS
```

```
md.pls = plsr(quality~., data=mydata, subset=train,
scale=TRUE, validation="CV")
validationplot(md.pls, val.type="MSEP")
pls.pred = predict(md.pls, mydata[test,], ncomp=7)
pls.mse = mean((pls.pred-quality.test)^2)
print(pls.mse)

# Methods: Decision Tree
library(tree)
md.tree = tree(quality~., data = mydata, subset=train)
cv.tree = cv.tree(md.tree)
plot(cv.tree$size, cv.tree$dev, type = "b")
# * Found size 11 is the best
## Predict
tree.pred = predict(md.tree, newdata = mydata[test,])
tree.mse = mean((tree.pred - quality.test)^2)
## Print
plot(md.tree)
text(md.tree, pretty = 0)
print(tree.mse)

# Methods: Random Forest
library(randomForest)
## Examine use of  m and number of trees B
plot(5:305, seq(0.3, 0.6, 0.001),
type="n", main="Test␣MSE", xlab="number␣of␣trees", ylab="Test␣MSE")
for(n in seq(5, 305, 10)){
  md.rf = randomForest(quality~., data = mydata,
  subset = train, mtry=4, importance=TRUE, ntree=n)
  rf.pred = predict(md.rf, newdata = mydata[test, ])
  rf.mse = mean((rf.pred - quality.test)^2)
  points(n, rf.mse)
}
for(n in seq(5, 305, 10)){
  md.rf = randomForest(quality~., data = mydata,
  subset = train, mtry=6, importance=TRUE, ntree=n)
  rf.pred = predict(md.rf, newdata = mydata[test, ])
  rf.mse = mean((rf.pred - quality.test)^2)
  points(n, rf.mse, col="red")
}
for(n in seq(5, 305, 10)){
  md.rf = randomForest(quality~., data = mydata,
  subset = train, mtry=11, importance=TRUE, ntree=n)
  rf.pred = predict(md.rf, newdata = mydata[test, ])
  rf.mse = mean((rf.pred - quality.test)^2)
  points(n, rf.mse, col="yellow")
}
## Select n=301, mtry=4
md.rf = randomForest(quality~., data = mydata,
subset = train, mtry=4, ntree=301, importance=TRUE)
rf.pred = predict(md.rf, newdata = mydata[test, ])
rf.mse = mean((rf.pred - quality.test)^2)
print(rf.mse)

# Method: RF with Boosting
library(gbm)
md.boost = gbm(quality~.,
data=mydata[train, ], distribution="gaussian",
                 n.trees=5000, interaction.depth=4, shrinkage = 0.2)
summary(md.boost)
boost.pred = predict(md.boost,
                  newdata = mydata[test, ], n.trees=5000)
mean((boost.pred-quality.test)^2)

# Extra: XGboost
library(xgboost)
xtrain = subset(mydata[train, ], select = -quality)
xtest = subset(mydata[test, ], select = -quality)
dtrain = xgb.DMatrix(data = as.matrix(xtrain), label=quality.train)
dtest = xgb.DMatrix(data = as.matrix(xtest), label=quality.test)
watchlist = list(train=dtrain, test=dtest)
## Select n=50 by trying
md.xgb = xgb.train(data = dtrain, max.depth=2,
                   nthread=2, nrounds=50, watchlist=watchlist,
                   )
xgb.pred = predict(md.xgb, as.matrix(xtest))
mean((xgb.pred-quality.test)^2)
xgb.plot.tree(model = md.xgb)
```

2 Yu's Code

```
# Load the all libraries
library(psych)
library(ggplot2)
library(dplyr)
library(tidyverse)
library(caret)
library(randomForest)
library(tree)
library(gbm)
library(corrplot)
library(e1071)
library(leaps)
library(lars)
library(kernlab)
library(GGally)
library(broom)

# Access the data
red <- read.csv("winequality-red.csv", sep = ";")
attach(red)
# Check the data whether there is any missing data
sum(is.na(red))

# Numeric Summary
summary(red)
describe(red)

# Check the skewness
for(i in 1:12){
  print(skewness(red[[i]]))
}

# Response Count
table(red$quality)

# Create the histograms for every variable
par(mfrow = c(3,4))
for(i in 1:12){
  hist(red[[i]], main = paste("Average =", signif(mean(red[[i]]),3)),
       xlab = names(red)[i])
}

# Create the boxplots for every variable
par(mfrow = c(3,4))
for(i in 1:12){
  boxplot(red[[i]], xlab= names(red)[i])
}
```

```
# Set seed
set.seed(563)

# Slice the data, 80% to the training and 20% to the test
train <- sample(1:nrow(red), size = nrow(red) * 0.8)
test <- dplyr::setdiff(1:nrow(red), train)

TrainDat <- red[train,]
TestDat <- red[test, ]

# Use the trainControl() method
trctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 1)

# Fit the MLR model
mlr_fit <- train(quality~., data = TrainDat, method = "glmStepAIC",
                 trControl = trctrl,
                 preProcess = c("center", "scale"))

# Trained MLR model result
mlr_fit
mlr_fit$finalModel

# Test set prediction for fit
mlr_pred <- predict(mlr_fit, newdata = TestDat)
# MSE of the model mlr_fit
mean((mlr_pred - TestDat$quality)^2)

# Fit the ridge regression model
ridge_fit <- train(quality~., data = TrainDat, method = "glmnet",
                   trControl = trctrl,
                   preProcess = c("center", "scale"))

# Trained model result
ridge_fit
ridge_fit$results
# obtain the coefficients
coef(ridge_fit$finalModel, ridge_fit$bestTune$lambda)


# Test set prediction
ridge_pred <- predict(ridge_fit, newdata = TestDat)
# MSE of the model ridge_fit
mean((ridge_pred - TestDat$quality)^2)

# Fit the lasso regression model
lasso_fit <- train(quality~., data = TrainDat, method = "lasso",
                   trControl = trctrl,
                   preProcess = c("center", "scale"))
```

```
# Trained model result
lasso_fit
lasso_fit$results
# Obtain the coefficients
length(lasso_fit$finalModel$L1norm)
predict(lasso_fit$finalModel, type = "coef", s = 14)

# Test set prediction
lasso_pred <- predict(lasso_fit, newdata = TestDat)
# MSE of the model lasso_fit
mean((lasso_pred - TestDat$quality)^2)

# Fit the elastic net regression model
enet_fit <- train(quality~., data = TrainDat, method = "enet",
                  trControl = trctrl,
                  preProcess = c("center", "scale"))

# Trained model result
enet_fit
enet_fit$results
# Obtain the coefficients
predict(enet_fit$finalModel, type = "coef", s = 12)

# Test set prediction
enet_pred <- predict(enet_fit, newdata = TestDat)
# MSE of the model enet_fit
mean((enet_pred - TestDat$quality)^2)

# Fit the PCR regression model
pcr_fit <- train(quality~., data = TrainDat, method = "pcr",
                 trControl = trctrl,
                 preProcess = c("center", "scale"),
                 tuneLength = 10)

# Trained model result and plot it
pcr_fit
pcr_fit$results
plot(pcr_fit)

# Test set prediction
pcr_pred <- predict(pcr_fit, newdata = TestDat)
# MSE of the model pcr_fit
mean((pcr_pred - TestDat$quality)^2)

# Fit the PLS regression model
pls_fit <- train(quality~., data = TrainDat, method = "pls",
                 trControl = trctrl,
                 preProcess = c("center", "scale"),
                 tuneLength = 10)

# Trained model result
pls_fit
pls_fit$results
# Plot the model
plot(pcr_fit)

# Test set prediction
pls_pred <- predict(pls_fit, newdata = TestDat)
# MSE of the model pls_fit
mean((pls_pred - TestDat$quality)^2)
```

3 Hunter's Code

More on `https://github.com/HunterJiang97/WineQuality`.

```
# Init
rm(list = ls())
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
load("CVdataset.RData")
library(MASS)
library(glm.predict)
library(class)
library(tree)
library(randomForest)
library(adabag)
library(ROSE)

# model functions
model.LDA <- function(train, test){
  # Linear Discriminate Analysis
  model_LDA <- lda(quality ~., data = train)
  yhat_LDA <- predict(model_LDA, test)
  return(yhat_LDA$class)
}

model.QDA <- function(train, test){
  # Quadratic Discriminate Analysis
  model_QDA <- qda(quality ~., data = train)
  yhat_QDA <- predict(model_QDA, test)
  return(yhat_QDA$class)
}

model.Logit <- function(train, test){
  train$quality <- train$quality - 1
  # Logistic Regression
  model_Logit <- glm(quality ~ ., family = binomial(), data = train)
  p_Logit <- predict(model_Logit, newdata = test, type = "response")
  yhat_Logit <- rep(0, length(p_Logit))
  yhat_Logit[p_Logit > 0.5] <- 1
  return(yhat_Logit + 1)
}

model.knn <- function(train, test, K){
  # K nearest neighbor
  train.m <- as.matrix(train[,1:11])
  test.m <- as.matrix(test[,1:11])
  train.q <- train$quality
  yhat_knn <- knn(train.m, test.m, train.q, k = K)
  return(yhat_knn)
}
```

```
model.opttree <- function(train, test){
  # Best Tree model
  model_tree <- tree(as.factor(quality) ~ ., data = train)
  cv_tree <- cv.tree(model_tree, FUN = prune.misclass)
  model_opt <- prune.tree(model_tree, best = cv_tree$size[which.min(cv_tree$dev)])
  yhat <- predict(model_opt, test, type = "class")
  return(yhat)
}

model.rf <- function(train, test, k){
  # Random Forest
  train$quality <- as.factor(train$quality)
  model_rf <- randomForest(quality~.,train, mtry=k, ntree = 100, importance=TRUE)
  yhat <- predict(model_rf, test)
  return(yhat)
}

model.bag <- function(train, test){
  # Bagging
  train$quality <- as.factor(train$quality)
  model_bag <- randomForest(quality~.,train, mtry=10, ntree = 100, importance=TRUE)
  yhat <- predict(model_bag, test)
  return(yhat)
}

# cv function to use
CV_fun <- function(trainset, testset){
  # Train methods
  ptm1 <- proc.time()
  yhat_LDA <- model.LDA(trainset, testset)
  ptm2 <- proc.time()
  yhat_QDA <- model.QDA(trainset, testset)
  ptm3 <- proc.time()
  yhat_Log <- model.Logit(trainset, testset)
  ptm4 <- proc.time()
  yhat_knn1 <- model.knn(trainset, testset, 1)
  ptm5 <- proc.time()
  yhat_knn3 <- model.knn(trainset, testset, 3)
  ptm6 <- proc.time()
  yhat_knn5 <- model.knn(trainset, testset, 5)
  ptm7 <- proc.time()
  yhat_knn7 <- model.knn(trainset, testset, 7)
  ptm8 <- proc.time()
  yhat_tree <- model.opttree(trainset, testset)
  ptm9 <- proc.time()
  yhat_rf <- model.rf(trainset, testset, 5)
  ptm10 <- proc.time()
  yhat_bag <- model.bag(trainset, testset)
  ptm11 <- proc.time()
  # Find the time for each method
  time <- rep(0, 10)
  time[1] <- (ptm2 - ptm1)[3]
  time[2] <- (ptm3 - ptm2)[3]
  time[3] <- (ptm4 - ptm3)[3]
  time[4] <- (ptm5 - ptm4)[3]
  time[5] <- (ptm6 - ptm5)[3]
  time[6] <- (ptm7 - ptm6)[3]
  time[7] <- (ptm8 - ptm7)[3]
  time[8] <- (ptm9 - ptm8)[3]
  time[9] <- (ptm10 - ptm9)[3]
  time[10] <- (ptm11 - ptm10)[3]

  # Form a list
  Res <- list(yhat_LDA, yhat_QDA, yhat_Log, yhat_knn1, yhat_knn3,
                 yhat_knn5, yhat_knn7, yhat_tree, yhat_rf, yhat_bag, time)
  return(Res)
}

# stats to use
Stat_OR <- function(YList, YhatList){
  n <- length(YhatList)
  nn <- length(YhatList[[1]]) - 1
  time <- rep(0, nn)
  Stat <- array(NA, dim = c(n, nn))
  tbl <- matrix(0, nrow = 2, ncol = 2)
  for (ii in 1:n){
    for (jj in 1:nn){
      Y <- YList[[ii]]$quality
      Yhat <- as.numeric(YhatList[[ii]][[jj]])
      Stat[ii,jj] <- length(which(Y == Yhat)) / length(Y)
      time[jj] <- time[jj] + YhatList[[ii]][[11]][jj]
      tbl <- tbl + as.matrix(table(Y, Yhat))
    }
  }
  return(list(Stat, time, tbl))
}

# Over-Sample function
over.sample <- function(response, data){
  num <- table(response)
  maxn <- max(num)
  df <- list()
  for (ii in 1:length(num)){
    if (num[ii] == maxn){
      df[[ii]] <- data[which(data$quality == ii),]
    } else {
      tmp <- data[which(data$quality == ii),]
      indi <- sample(1:dim(tmp)[1], maxn, replace = T)
      df[[ii]] <- tmp[indi,]
    }
  }
  df1 <- rbind(df[[1]], df[[2]])
  return(df1)
}

# 10-Fold CV for two class
K = 10
YhatsLList <- list()
CVtrain2 <- CVtrain
CVtest2 <- CVtest
for (ii in 1:K){
  CVtrain2[[ii]]$quality[CVtrain2[[ii]]$quality <=5] <- 1
  CVtrain2[[ii]]$quality[CVtrain2[[ii]]$quality >=6] <- 2
  CVtest2[[ii]]$quality[CVtest2[[ii]]$quality <=5] <- 1
  CVtest2[[ii]]$quality[CVtest2[[ii]]$quality >=6] <- 2
  # Balanced dataset
  CVtrain2[[ii]] <- over.sample(CVtrain2[[ii]]$quality, CVtrain2[[ii]])
  train <- CVtrain2[[ii]]
  test <- CVtest2[[ii]]
  YhatsLList[[ii]] <- CV_fun(train, test)
}
OR_table <- Stat_OR(CVtest2, YhatsLList)
m <- colMeans(OR_table[[1]])
s <- apply(OR_table[[1]], 2, sd)
Result <- rbind(m, s, OR_table[[2]])
colnames(Result) <- c("LDA", "QDA", "Logistic", "KNN1", "KNN3", "KNN5", "KNN7" , "Optimal
rownames(Result) <- c("avg", "sd", "time")
Result2O <- Result
Result2O
OR_table[[3]]
OR_table[[3]] / sum(OR_table[[3]])
```

4 Himanish's Code

```
# Read data

data = read.csv('K:/Downloads/winequality-red.csv',sep=';')
summary(data)
# Data without quality label
data.uns = data[-c(12)]
summary(data.uns)
# Find out correlated variables.
cor(data.uns)
# Remove correlated variables
# Keep only sugar, chlorides,total.sulfur.dioxide,density,sulphates,alcohol as features.

data.sel = data.uns[c(4,5,7,8,10,11)]
summary(data.sel)
# Min-max scaling

data.sd = sapply(data.sel, function(x) (x - min(x))/(max(x) - min(x)))
data.sd = data.frame(data.sd)    # norm_data is a 'matrix'

#data.sd = scale(data.sel)

summary(data.sd)
set.seed(1)

ks = 1:10
ws = rep(0,length(ks))

for (k in ks)
{
    kmo = kmeans(data.sd,centers = k,nstart=50)
    ws[k] = kmo$tot.withinss
}

plot(ks,ws,type='l',xlab='k',ylab='WSS')
points(ks,ws,)
# There is no appreciable reduction in WSS after k = 5.

ncent = 5

km = kmeans(data.sd,centers=ncent,nstart=50)

#km
data.sd = cbind(data.sd,cluster=km$cluster)
data.sd = data.frame(data.sd)
data.sd$quality = data$quality
library(repr)
options(repr.plot.width=20, repr.plot.height=20,repr.plot.res=100,repr.plot.pointsize=20)
```

```
pairs(data.sd[c(1:ncent)],col=data.sd$cluster,pch=19)
# Silhouette

library(cluster)

dis2 = dist(data.sd[c(1:6)])^2

sil = silhouette(km$cluster,dis2)
plot(sil, col=1:ncent, border = NA)
```