STSCI 4740 Final Project Report
Names: Michael Zhou (mgz27), Minqian Hou (mh2256), Jiayue Pan (jp987), Tong Sha (ts646)

Introduction

We used the Wine Quality Dataset from the UCI Machine Learning Repository to predict the red and white wine qualities, given their physicochemical attributes. We tried the following methods: linear and logistic regression, random forests, XGBoost, SVMs, Naive Bayes, ridge and lasso regression. Our results show that SVMs outperform just about any other model in terms of test accuracy, while the test accuracies for red wines are consistently greater than white wines; however, the accuracies themselves are not as high due to the overall lack of correlation between predictors.

About the Dataset

We used the Wine Quality Dataset from the UCI Machine Learning Repository as our dataset for analysis **[1]**. It contains two datasets, each consisting of red wines and white wines. Each dataset contains 12 variables: fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol, and quality (integers ranging from 0 to 10). The goal is to predict the wine quality. There are 1599 red wine samples and 4898 white wine samples.

Class Distribution

We plotted the class label distributions for red wines and white wines separately, as displayed in figure 1. We can see that the majority of labels are either a 5, 6, or 7, with very few labels outside this range. The lowest quality given in both datasets is a 3. We see a few white wine samples with a quality of 9, while the maximum quality in all red wine examples is 8. Looking at this plot, this dataset is highly imbalanced.

Correlations Among Predictors

We then plotted the correlations between each predictor for the red (left) and white (right) wine datasets, in the form of heat maps (figure 2) and histograms (figure 3). We notice that most of the predictor pairs in the heat map not on the main diagonal have a light orange color, which denotes a weak inter-predictor relationship. We also see that most of these predictor pairs have a correlation closer to 0, which similarly denotes very little correlation among predictors. We can see later that this weak correlation among predictors, as well as the highly imbalanced class distribution, makes our models very difficult to yield high performances. Very importantly, we can see that the red wine dataset has more variable pairs with higher correlations, which makes it easier for models to pick up patterns within predictors; as we see later, this contributes to models predicting more accurately for red wines than white wines.

Approaches & Experimental Process

In this report, we mainly talk about four different methods that we've used in detail: linear and logistic regression, random forests, XGBoost, and SVM. We considered this wine quality prediction problem in both regression and classification settings. We ran both linear and logistic regression with red and wine samples in the same dataset and in separate datasets. We ran XGBoost with red and wine samples in the same dataset, along with random forest models and SVM models with red and white wine samples sundered in two datasets.

In our notebooks, we sampled the proportion of predicted values within 0.5 and 1 of the actual class, as inspired by the reference paper in **[2]**. The latter is significantly higher because of how close the majority of class labels are skewed together. For regression problems in general, we only use the proportion of predicted values within 0.5 of the ground truth label as our actual test accuracy, as this is equivalent to rounding the predicted values to their nearest class. We simply use the classification accuracy to evaluate classification methods. For all our models, we always use a 80-20 train-test split, with all random seeds set to 1 for reproducibility.

Method 1: Linear & Multinomial Logistic Regression

The first methods we tried are linear and (multinomial) logistic regression, as we tried to treat wine quality prediction as both a regression and classification problem. Both linear and logistic regression are treated as baseline models that are normally attempted first. We tried running both methods on the combined dataset (red and white wines in the same dataset) and on separate datasets.

First, we tried using linear regression on the combined dataset. For preprocessing, we scaled all our predictors (including factor variables, which are converted into numeric) in the entire dataset with mean 0 and standard deviation 1. Using a 80-20 train-test split and assuming default hyperparameters, we obtain a test accuracy of 0.5692308. As a reminder, for all regression models we tried, the test accuracy is the proportion of (continuous) predictions values within 0.5 of the ground truth. When we obtained the model summary (shown in figure 4), we found out that citric.acid and chlorides had a p-value higher than any reasonable threshold, so we fit another model without these variables. Although all our remaining variables were significant in this model using p-value 0.05 (shown in model summary in figure 5), our test accuracy dipped slightly to 0.5675923.

We also trained linear regression on separate red and white datasets. We standardized the red and white datasets to have mean 0 and variance 1. Using the same process as described above, we obtained a test accuracy of 0.5285714 on the white dataset and 0.6375 on the red dataset, for a weighted average of 0.5553846. Once again, when we obtained the model summaries, we found out that four of the 11 predictors had a p-value higher than 0.05 for the white dataset as shown in figure 6, and six of them for the red dataset as depicted in figure 7. We then removed these insignificant predictors, and refitted these models. Our test accuracy for white wines stayed the same, while the red dataset now yields 0.634375 test accuracy, for a weighted average of 0.5546154. All variables for both the white and red datasets were significant, as they are lower than any reasonable p-value threshold as shown in figures 8 and 9.

We also tried the logistic regression models for classification. We performed standardization procedures the same way as in linear regression. We obtained a 0.5653846 test accuracy for the combined dataset. Unlike in linear regression, we did not perform any variable pruning, as multinomial logistic regression models do not print out the p-values for each variable. When the white and red wine samples are sundered into separate subsets, test accuracies of 0.5561224 and 0.621875 were obtained for the white and red subsets respectively, for a final weighted accuracy of 0.5723077.

In general, we do not see much of a difference in terms of test accuracy between all six of these models above. The highest accuracy thus far is obtained via logistic regression on separate datasets, with over 57% test accuracy, while the lowest is around 55.46% for linear regression on split dataset and only significant predictors. Removing insignificant variables did not affect our test accuracy by very much, in fact it actually decreased by tenths of a percentage point. Removing insignificant variables only changes the proportions of predictors that are significant to the model, but this does not mean that it will improve the generalization accuracy of the model. For both linear and logistic regression, we see that the red wine dataset has a higher test accuracy compared to the white wine dataset. This is counterintuitive, since we know that a larger number of data samples usually yield a larger test accuracy, there are less red samples (1599) than white samples (4898). We hypothesize that both models are picking up patterns in red wines more accurately.

Multinomial logistic regression also has a slightly longer runtime in R compared to linear regression. We can simply use the glm function to get a list of continuous values for prediction. However, we had to use a special library designed for multinomial logistic regression, nnet, which takes longer to train. The same glm function we used in linear regression can only handle binary classification, while our quality labels are multi-class. While standardizing predictor variables is not required for linear or logistic regression, it is useful for faster convergence during training (especially for logistic regression). Notice that the logistic regression converged with default settings (100 maximum iterations) for both the white and red subsets, thanks to standardization. Such convergence would not be possible without standardization, according to our previous trials.

Method 2: Random Forests

In an attempt to improve the logistic and linear regression models, we tried random forests (regression and classification methods) with red and white wine datasets run separately. Random forests decorrelate the trees by randomly sampling m << p predictors (if a dataset has p predictors in total). The number of trees (B) is not important, as we usually choose a large number, since random forests do not overfit. We thought that ensemble methods, including random forests, were suitable for our problem.

For all models, we looped through the number of sampling variables m from 1 to 11, calculated and plotted the test accuracies and fit times, and recorded the model yielding the maximum test accuracy (including the optimal m, its fit time, as well as the test accuracy itself). All other parameters were left as default. Note that the default number of trees is B = 500.

Let's talk about the regression models first. It takes 36 seconds to finish fitting models for each m on the red dataset and yield a final test accuracy of 0.634375; whereas, on the white subset, it takes 2-and-a-half minutes to complete the total CV search and muster a 0.5795918 accuracy; overall, we get a weighted average test accuracy of 0.5930769.

Moving on to random forest classification models. It takes about 17 seconds to finish fitting models for each m on the red dataset and yield a final test accuracy of 0.61875; whereas, on the white subset, it takes over a minute to complete the total CV search and muster a 0.5826531 accuracy; overall, we get a weighted average test accuracy of 0.5915385.

Once again, we see that the red subset outperforms the white subset in terms of test accuracy, and is also much faster in terms of runtime. The latter makes sense, as there are about 4 times as many white samples than red samples. Like linear and logistic regression, random forests are also more adept at predicting red wines than white wines, as they are picking up patterns in red wines more accurately.

We can see from figures 10 and 12 that there is no clear relationship between test accuracies and the number of variables sampled for random forests, as the optimal m is different for each model. In general, we cannot tell which number of predictors sampled will yield the largest test accuracy, since this is highly dependent on the dataset. As corroborated in figures 11 and 13, we can see that the fit times increase linearly for each increase in m, since sampling more variables means using more predictors to build trees, which increases the runtime linearly (since there are no predictors). We would expect that the fewer variables we sample on, the less time it takes to fit the random forests model, as fewer variables mean a smaller feature dimension space, which also prevents us from overfitting!

While random forests seem like a reasonable method to try, they are usually not congenial for imbalanced class distributions, as they are built on decision tree models. Since each tree chooses features in a uniform distribution, trees will usually be biased in majority classes **[4]**. Common techniques to circumvent include oversampling and undersampling data samples during the bootstrapping process **[3]**, which are outside the scope of this class. We tried this technique mainly because it was covered in lecture.

Method 3: XGBoost

We then tried the XGBoost method. XGBoost, or "eXtreme Gradient Boosting", is a boosting method implementation specifically designed to optimize computational speed, model performance, and scalability, with slightly more sophisticated training procedures **[5]**. In recent years, XGBoost has dominated machine learning, data mining, and Kaggle competitions. For boosting trees in general, we can choose any tree depth from 1 to 4, and a shrinkage rate (learning rate) of either 0.01 or 0.001, while choosing the number of tree estimators carefully.

For our training and validation process, we used a combined dataset consisting of both red and white wine samples to train XGBoost, mainly because we had trouble trying to set the proper configurations to run XGBoost for separate red and white wine datasets; specifically, the xgboost package in R requires that the class labels start from 0 and the same number of factor levels, and there were different class label ranges for red and white wines (3 to 8 for red, but 3 to 9 for white). We fit an XGBoost model for each number of tree estimators from 200 to 2000, with an interval of 200. We fixed the shrinkage rate and tree depth parameters to 0.01 and 2 respectively, used the "multi::softprob" objective function, with the evaluation metric set as "merror". For each hyperparameter configuration (number of tree estimators), we obtained the test accuracies from the model. Because of the nature of the inputs for XGBoost in R, we only used this method for classification.

We plotted our test accuracy rates (left) and fit times in seconds (right) for each number of estimators as shown in figure 14. We can clearly see that the test accuracy staggers across different numbers of iterations, with the highest occurring at 1800 estimators. After 1800 estimators, XGBoost starts to overfit. The fit times increase linearly as the number of estimators increase (with the longest fit time at 83 seconds), as more estimators mean more trees to be added in the model. Since the XGBoost algorithm builds up multiple trees (like random forests), it naturally takes slower to train.

After 509.400 seconds of fitting models for each configuration, we found out that the best model yielded 1800 estimators with a test accuracy of 0.5584615. It took 76.530 seconds to train this model. These results are recorded in the performance summary table.

Although XGBoost is advertised as an efficient, scalable boosting algorithm boasting high accuracy, it does not improve our overall test accuracy compared to any of the methods that we've mentioned so far. Unlike random forests, XGBoost (and any other boosting method) is prone to overfit the test data for a large number of estimators, and subject to a lower test accuracy.

Method 4: SVM

We have also tried the SVM classification and regression techniques separately for red and white wine samples. For both methods, we standardized our datasets before splitting them into training and test sets. We kept the default parameters for our SVM model, using a Gaussian RBF kernel. Our hyperparameter we searched over was the RBF kernel parameter gamma, which we used the same range gamma = {2^(-15), 2^(-14), …, 2^(3)} as recommended in the reference paper **[2]**, obtained the test accuracies, and found the gamma that yielded the highest test accuracy. We recorded and plotted the test accuracies and fit times for each gamma tested. We also recorded the total time for the CV search.

Our SVM classification method on the red wines dataset yielded a test accuracy of 0.628125 for gamma = 2^(-2), while on the white dataset yielded 0.6020408 test accuracy for gamma = 2^(-6), for a weighted average of 0.6084615. This is the first time thus far that we see a weighted test accuracy above 60%. For SVM regression, we defined our model type as 'eps-regression', which denotes the epsilon-insensitive loss function for regression, as the reference paper uses this method. This method applied on the red wines dataset yielded a test accuracy of 0.640625 for gamma = 2^(-9), while on the white dataset yielded 0.5836735 test accuracy for gamma = 2^(-3), for a weighted average of 0.5976923.

We also plotted the test accuracies and fit times for both methods on red and white subsets, as shown in figures 15 through 18. For each increasing gamma (starting from 2^(-15)), a common trend is that the test accuracy generally starts out small, increases first, and then decreases back again (although we do see more fluctuation for regression on the white subset). A general trend for the fit times is that they are usually low for smaller values of gamma, but start to increase when gamma reaches closer to 1 and surpasses it. The gamma parameter determines how much impact a single example has on predictions, with larger gamma values implying a more flexible decision boundary (two examples are similar only when closer together), which also means a higher proclivity of overfitting **[6]**. We can see that it takes more time to fit SVM with large gamma values, as it takes more time to demarcate the decision boundaries. Not surprisingly, it takes more time to train white samples than red samples as there are more of them.

Other Methods Attempted

In addition to the methods above, we tried the Naive Bayes model (with default settings) for the red and white datasets separately, yielding test accuracies of 0.5875 for red wines and 0.3938776 for white wines, for a weighted average accuracy of 0.4415385. We also attempted ridge and lasso regression, obtaining 0.64025 and 0.646875 test accuracy respectively for red wines, but a measly 0.5408163 and 0.5316327 for white wines. The 0.646875 test accuracy for lasso regression is the highest out of any model fit on the red wines subset, but it does not generalize well to white wines.

Why is our Accuracy So Low Across Models?

All four of our methods produced an extremely low test accuracy, with the highest performance just over 60% test accuracy. To see why this is, let's revisit the correlation plots between the predictors described before.

As shown in the heat maps and correlation histograms, there is very little correlation between the different predictors, so it is very difficult to find patterns among the variables, because there are not that many predictors (only 12 of them) and they are not very strong indicators. This contributes to low accuracies across different models, since we can't utilize any inter-predictor patterns to better learn about the data **[7]**.

Moreover, we can also see that the class labels (wine qualities) are extremely unbalanced, as the bulk of the labels are 5, 6, or 7, yet there are very few examples that are 4 or lower or 8 or above. The models, in turn, will also spit out class labels that are equally unbalanced, as the predictors will also predict mostly 5s, 6s, or 7s, while very rarely predicting any other quality value.

Performance Summary Table

The table below is a summary of the test accuracies, fit times, and CV times (if applicable) obtained from each approach. Random forests and SVM methods are trained on separate red and white wine datasets, while XGBoost is trained on the combined dataset. For methods trained on separate datasets, the overall test accuracies are calculated using a weighted average (based on the number of red and white samples). If a hyperparameter grid was used for search, the fit time is recorded for the model chosen. The highest test accuracy (red, white, overall) is highlighted in yellow.

Method	Regression or Classification?	Subset	Fit Time of Chosen Model (s)	CV Search Time (s)	Test Accuracy	Overall Test Accuracy
Linear Regression	Regression	Red (original variables)	0.121	N/A	0.6375	
		White (original variables)	0.098	N/A	0.5285714	
						0.5553846
		Red (significant variables)	0.101	N/A	0.634375	
		White (significant variables)	0.087	N/A	0.5285714	
						0.5546154
		Combined (original variables)	0.144	N/A	0.5692308	**0.5692308**
		Combined (significant variables)	0.112	N/A	0.5676923	**0.5676923**
Logistic Regression	Classification	Red	0.12	N/A	0.621875	
		White	0.414	N/A	0.5561224	
						0.5723077
		Combined	0.671	N/A	0.5653846	**0.5653846**
Random Forest	Regression	Red	2.483	35.992	0.634375	
		White	17.08	157.985	0.5795918	
						0.5930769
	Classification	Red	1.419	17.616	0.61875	
		White	5.917	68.724	0.5826531	
						0.5915385
XGBoost	Classification	Combined	76.53	509.4	0.5584615	**0.5584615**
SVM	Regression	Red	0.128	3.205	0.640625	
		White	1.276	27.852	0.5836735	
						0.5976923
	Classification	Red	0.15	3.955	0.628125	
		White	1.668	36.059	**0.6020408**	
						0.6084615
Naïve Bayes	Classfication	Red	0.097	N/A	0.5875	
		White	0.106	N/A	0.3938776	
						0.4415385
Ridge Regression	Regression	Red	0.18	0.021	0.640625	
		White	0.294	0.018	0.5408163	
						0.5653846
Lasso Regression	Regression	Red	0.115	0.021	**0.646875**	
		White	0.406	0.021	0.5316327	
						0.56

Model Comparison & Conclusion

We used the Wine Quality Dataset to predict the red and white wine qualities, given certain attributes about the wines, using linear and logistic regression, random forests, XGBoost, and SVMs, as well as naive Bayes, and ridge and lasso regression. Comparing all our methods from the performance summary table, we can easily argue that SVM is the superior method in terms of test accuracy for both red and white wines. The highest test accuracy for white wines, as well as the highest overall test accuracy, is obtained from SVM classification, both over 60%. The SVM classification model generalizes the best for the white wine subset (the only model over 60% for white wines), while generalizing relatively well (62%) for red wines, while other models have wider accuracy differences between red and white. Lasso regression yielded the highest test accuracy for red wines (above 64.68%). SVM regression is tied for a close second with ridge regression. Yet both ridge and lasso perform under 55% for white wines. We can also argue that XGBoost is not an apropos model for our problem, as it takes the longest time to train, only to yield a suboptimal test accuracy of under 55%, which is lower than most models. Naive Bayes is also not suitable, as it produces the lowest overall test accuracy of any model. Although producing decent accuracy, random forests in general are rarely suitable for datasets with skewed labels. Consistently, we see that for all models we tried that run on separate red and white wine subsets, the test accuracy is greater for red wines, as only one model fell short of the 60% accuracy mark for red wines, while only one model surpassed 60% for white wines. This is mostly because the inter-predictor correlation relationships are closer for red wines, as there are more pairs of predictors with higher correlations. All our methods do not yield a very high test accuracy, mainly because the correlations between predictors are mostly weak, as shown in the correlation heat maps and histograms, and because the class labels are heavily skewed towards the middle, which makes the models predict certain quality values more than others.

Appendix

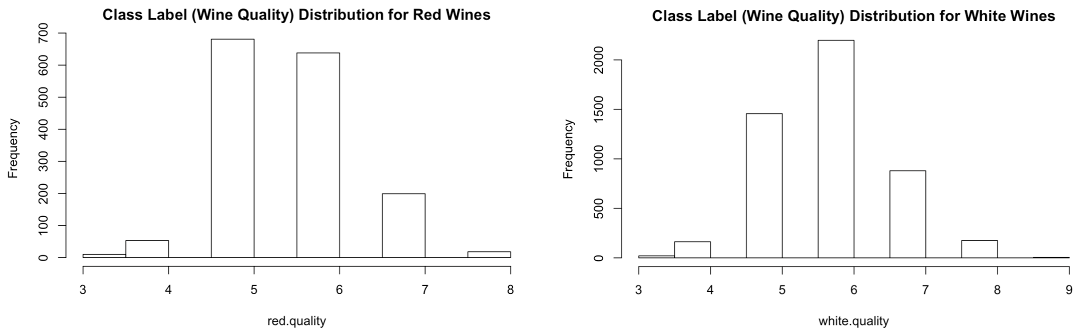


Figure 1: Class label distributions for red wines (left) and white wines (right)

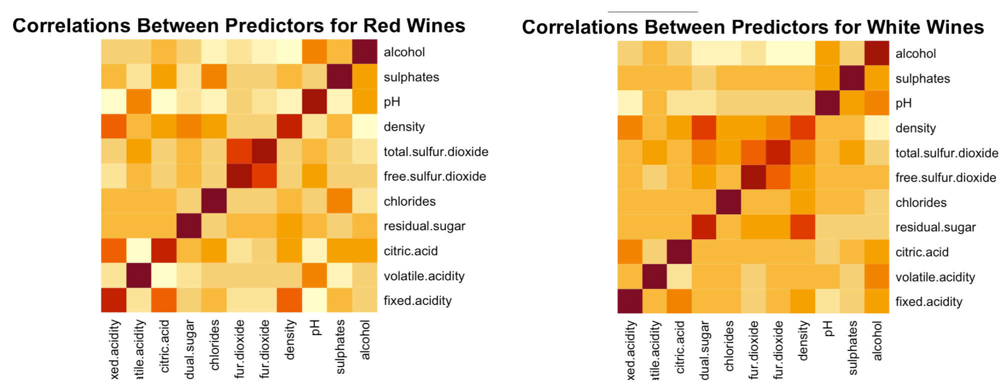


Figure 2: Heat maps showing correlations between predictors for red wines (left) and white wines (right). Lighter color denotes weaker relationship between two predictors.

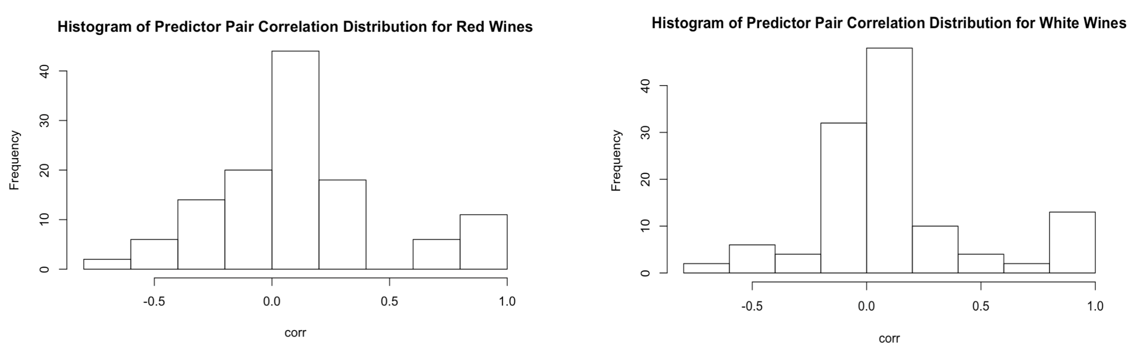


Figure 3: Histograms showing correlations between predictors for red wines (left) and white wines (right).

```
Call:
glm(formula = quality ~ ., data = data.train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.8306  -0.4856  -0.0393   0.4685   3.0600

Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
(Intercept)           5.810147   0.013564 428.362  < 2e-16 ***
type                 -0.255863   0.034286  -7.463 9.89e-14 ***
fixed.acidity         0.113441   0.025227   4.497 7.05e-06 ***
volatile.acidity     -0.287466   0.017108 -16.803  < 2e-16 ***
citric.acid          -0.007659   0.013272  -0.577   0.5639
residual.sugar        0.378264   0.034087  11.097  < 2e-16 ***
chlorides            -0.020418   0.014589  -1.400   0.1617
free.sulfur.dioxide   0.077500   0.014680   5.279 1.35e-07 ***
total.sulfur.dioxide -0.045617   0.020504  -2.225   0.0261 *
density              -0.435904   0.053968  -8.077 8.18e-16 ***
pH                    0.115117   0.016402   7.019 2.53e-12 ***
sulphates             0.089172   0.013256   6.727 1.92e-11 ***
alcohol               0.225247   0.027268   8.261  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for gaussian family taken to be 0.5575871)

    Null deviance: 4059.4  on 5196  degrees of freedom
Residual deviance: 2890.5  on 5184  degrees of freedom
AIC: 11728

Number of Fisher Scoring iterations: 2
```

Figure 4: Model summary for linear regression run on the combined dataset with all predictors.

```
Call:
glm(formula = quality ~ ., data = data.train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.8118  -0.4875  -0.0427   0.4692   3.0577

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)
(Intercept)           5.80940    0.01354 429.008  < 2e-16 ***
type                 -0.25048    0.03363  -7.447 1.11e-13 ***
fixed.acidity         0.11665    0.02474   4.716 2.47e-06 ***
volatile.acidity     -0.28666    0.01666 -17.202  < 2e-16 ***
residual.sugar        0.38735    0.03361  11.524  < 2e-16 ***
free.sulfur.dioxide   0.07663    0.01467   5.224 1.82e-07 ***
total.sulfur.dioxide -0.04626    0.02047  -2.260   0.0238 *
density              -0.44886    0.05336  -8.412  < 2e-16 ***
pH                    0.12070    0.01603   7.530 5.96e-14 ***
sulphates             0.08597    0.01311   6.560 5.91e-11 ***
alcohol               0.22391    0.02714   8.251  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for gaussian family taken to be 0.55765)

    Null deviance: 4059.4  on 5196  degrees of freedom
Residual deviance: 2892.0  on 5186  degrees of freedom
AIC: 11726

Number of Fisher Scoring iterations: 2
```

Figure 5: Model summary for linear regression run on the combined dataset with only significant predictors. Any predictor with p-value greater than 0.05 was removed in the previous model.

```
Call:
glm(formula = quality ~ ., data = data.train.white)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.4877  -0.5083  -0.0267   0.4738   3.1563

Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
(Intercept)           5.909111   0.012403 476.412  < 2e-16 ***
fixed.acidity         0.037385   0.019550   1.912   0.0559 .
volatile.acidity     -0.175857   0.013087 -13.438  < 2e-16 ***
citric.acid          -0.004634   0.012942  -0.358   0.7203
residual.sugar        0.401750   0.041591   9.660  < 2e-16 ***
chlorides            -0.001167   0.013517  -0.086   0.9312
free.sulfur.dioxide   0.080784   0.016457   4.909 9.54e-07 ***
total.sulfur.dioxide -0.009183   0.017935  -0.512   0.6087
density              -0.440531   0.061364  -7.179 8.37e-13 ***
pH                    0.112191   0.017706   6.336 2.62e-10 ***
sulphates             0.085696   0.013107   6.538 7.02e-11 ***
alcohol               0.276059   0.032370   8.528  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 0.5833359)

    Null deviance: 3251.6  on 3917  degrees of freedom
Residual deviance: 2278.5  on 3906  degrees of freedom
AIC: 9021

Number of Fisher Scoring iterations: 2
```

Figure 6: Model summary for linear regression run on the white wines subset with all predictors.

```
Call:
glm(formula = quality ~ ., data = data.train.red)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-2.67559  -0.38331  -0.07094   0.45606   1.97593

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)
(Intercept)           5.65961    0.01840 307.634  < 2e-16 ***
fixed.acidity         0.03971    0.05163   0.769 0.441931
volatile.acidity     -0.18833    0.02437  -7.729 2.20e-14 ***
citric.acid          -0.03509    0.03230  -1.087 0.277460
residual.sugar        0.01494    0.02508   0.596 0.551596
chlorides            -0.08276    0.02184  -3.789 0.000158 ***
free.sulfur.dioxide   0.03473    0.02598   1.337 0.181599
total.sulfur.dioxide -0.12127    0.02708  -4.478 8.22e-06 ***
density              -0.03665    0.04548  -0.806 0.420436
pH                   -0.05272    0.03289  -1.603 0.109171
sulphates             0.13683    0.02095   6.530 9.49e-11 ***
alcohol               0.30200    0.03083   9.795  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 0.419634)

    Null deviance: 843.43  on 1278  degrees of freedom
Residual deviance: 531.68  on 1267  degrees of freedom
AIC: 2532.9

Number of Fisher Scoring iterations: 2
```

Figure 7: Model summary for linear regression run on the red wines subset with all predictors.

```
Call:
glm(formula = quality ~ ., data = data.train.white)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.4457  -0.5129  -0.0220   0.4685   3.2217

Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
(Intercept)          5.91103    0.01234 479.013  < 2e-16 ***
volatile.acidity    -0.18026    0.01250 -14.416  < 2e-16 ***
residual.sugar       0.35495    0.03064  11.585  < 2e-16 ***
free.sulfur.dioxide  0.07436    0.01322   5.624 2.00e-08 ***
density             -0.36915    0.04227  -8.734  < 2e-16 ***
pH                   0.09073    0.01289   7.041 2.25e-12 ***
sulphates            0.08094    0.01287   6.288 3.57e-10 ***
alcohol              0.31303    0.02550  12.274  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 0.5833682)

    Null deviance: 3251.6  on 3917  degrees of freedom
Residual deviance: 2281.0  on 3910  degrees of freedom
AIC: 9017.2

Number of Fisher Scoring iterations: 2
```

Figure 8: Model summary for linear regression run on the white wines subset with only significant predictors. Any predictor with p-value greater than 0.05 was removed in the previous model.

```
Call:
glm(formula = quality ~ ., data = data.train.red)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-2.66477  -0.39657  -0.06996   0.46170   2.07097

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)
(Intercept)           5.66006    0.01819 311.082  < 2e-16 ***
volatile.acidity     -0.19480    0.01940 -10.042  < 2e-16 ***
chlorides            -0.08122    0.02020  -4.022 6.11e-05 ***
total.sulfur.dioxide -0.10098    0.01821  -5.546 3.54e-08 ***
sulphates             0.13800    0.02029   6.802 1.59e-11 ***
alcohol               0.30804    0.01892  16.284  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 0.4216015)

    Null deviance: 843.43  on 1278  degrees of freedom
Residual deviance: 536.70  on 1273  degrees of freedom
AIC: 2533

Number of Fisher Scoring iterations: 2
```

Figure 9: Model summary for linear regression run on the red wines subset with only significant predictors. Any predictor with p-value greater than 0.05 was removed in the previous model.

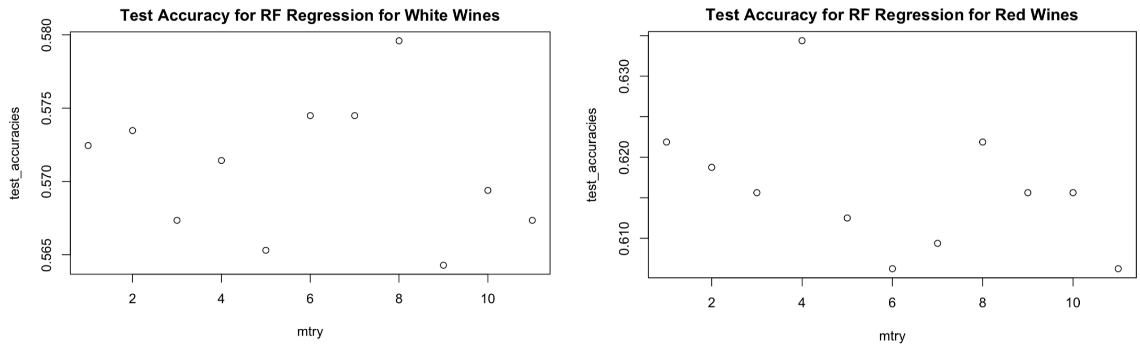


Figure 10: Test accuracies for random forest regression on red (left) and white (right) dataset for each number of predictors sampled.

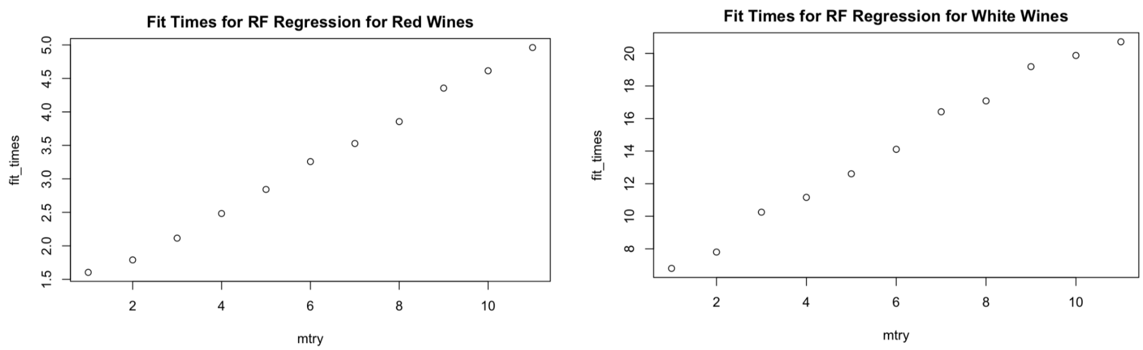


Figure 11: Fit times for random forest regression on red (left) and white (right) dataset for each number of predictors sampled.

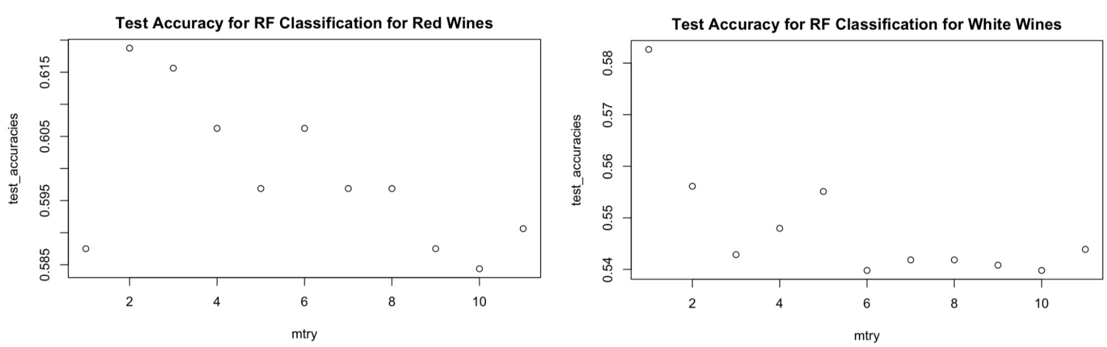
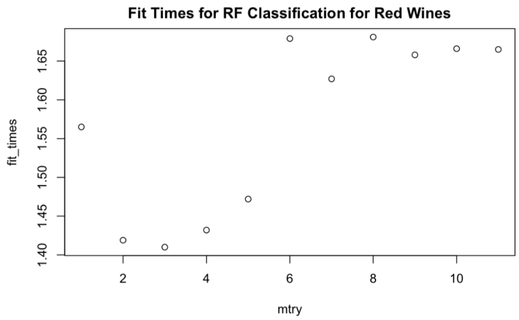


Figure 12: Test accuracies for random forest classification on red (left) and white (right) dataset for each number of predictors sampled.



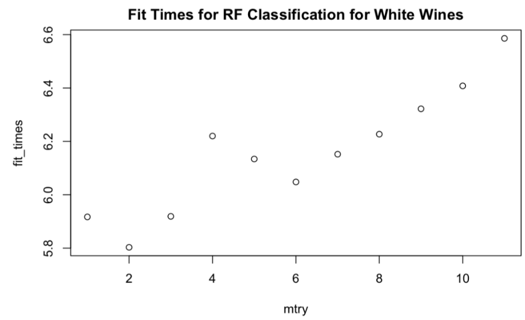
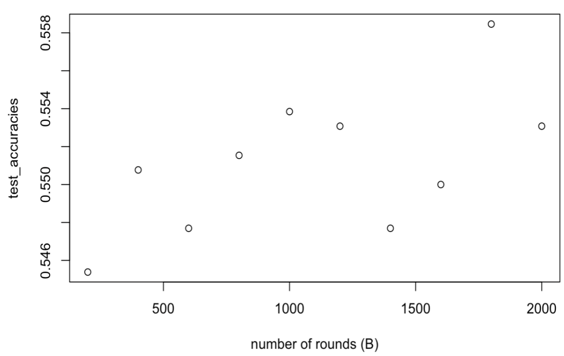


Figure 13: Fit times for random forest classification on red (left) and white (right) dataset for each number of predictors sampled (mtry).



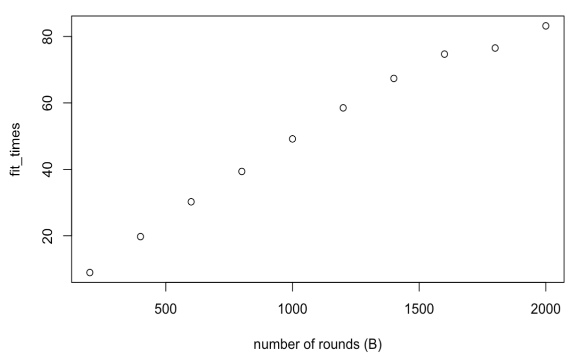
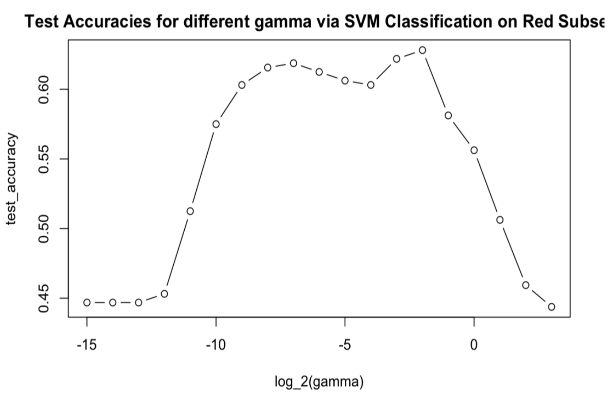


Figure 14: Test accuracy rates (left) and fit times in seconds (right) for each number of estimators for XGBoost.



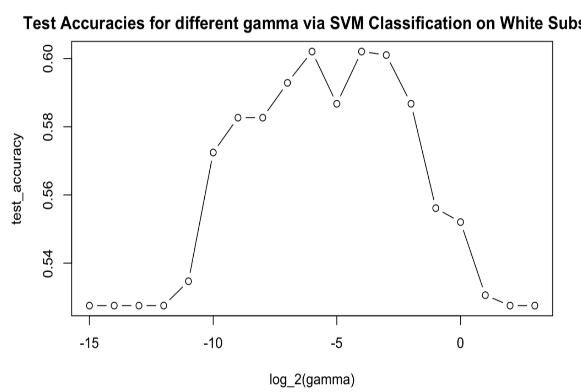


Figure 15: Test Accuracies for SVM classification on red (left) and white (right) dataset for each gamma in RBF kernel.

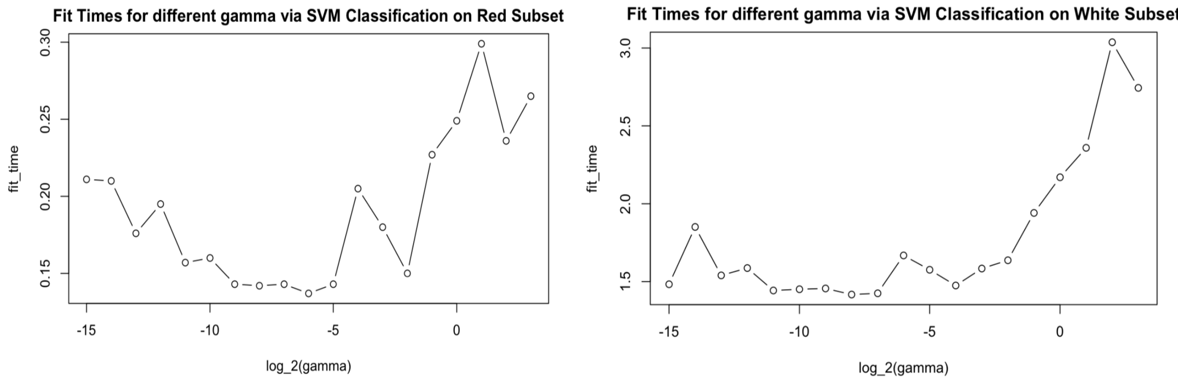


Figure 16: Fit times for SVM classification on red (left) and white (right) dataset for each gamma in RBF kernel.

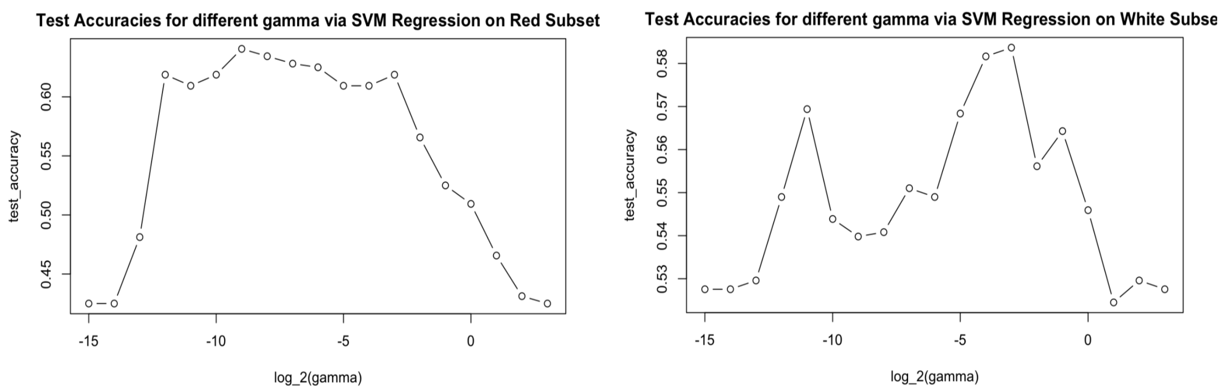


Figure 17: Test Accuracies for SVM regression on red (left) and white (right) dataset for each gamma in RBF kernel.

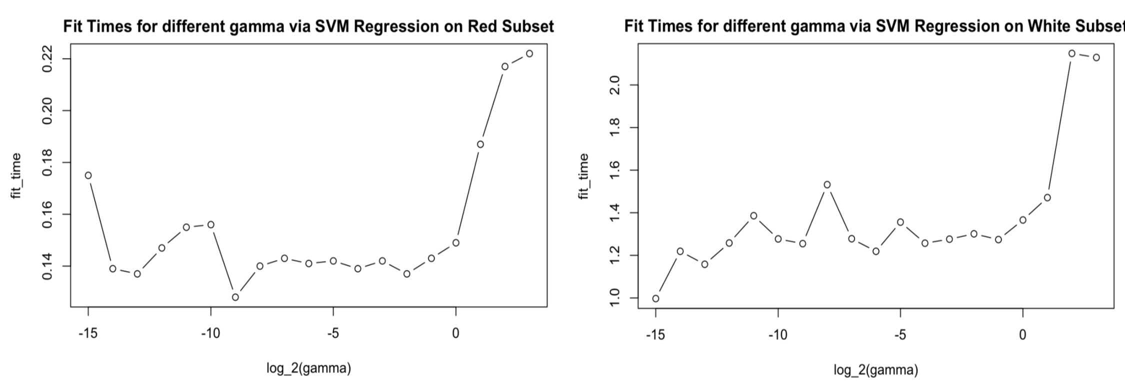


Figure 18: Fit times for SVM regression on red (left) and white (right) dataset for each gamma in RBF kernel.

References

1. Wine Quality Data Set
   https://archive.ics.uci.edu/ml/datasets/wine+quality
2. Reference Paper: P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. *Modeling Wine Preferences by Data Mining from Physicochemical Properties*. In Decision Support Systems, Elsevier, 47(4):547-553, 2009.
   https://www.sciencedirect.com/science/article/abs/pii/S0167923609001377?via%3Dihub
3. Random Forest for Imbalanced Dataset: Example with Avalanches in French Alps
   https://towardsdatascience.com/random-forest-for-imbalanced-dataset-example-with-avalanches-in-french-alps-77ffa582f68b
4. Is Random Forest a Good Option for Unbalanced Data Classification?
   https://stats.stackexchange.com/questions/242833/is-random-forest-a-good-option-for-unbalanced-data-classification
5. A Gentle Introduction to XGBoost for Applied Machine Learning
   https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/
6. RBF SVM Parameters
   https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html
7. Estimating Wine Quality with Machine Learning (AI), 72% Accuracy
   https://ai.plainenglish.io/estimating-wine-quality-with-machine-learning-ai-72-accuracy-8a5ff0bab3b2

R Notebook

Code

This is an R Markdown (http://rmarkdown.rstudio.com) Notebook. When you execute code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Cmd+Shift+Enter*.

—SVM classification—

Red dataset

Hide

```
library(kernlab)
library(e1071)
library(MASS)

set.seed(1)
red <- read.csv('winequality-red.csv', header = TRUE, sep=";")
red <- na.omit(red)
red.quality <- red$quality
red[,-12] <- scale(red[,-12])

# Red dataset
train_red_idx <- sample(nrow(red) * 0.8) # 80-20 train-test split
train.red <- red[train_red_idx,]
train.red$quality <- as.factor(train.red$quality)
train.red.quality <- train.red$quality

test.red <- red[-train_red_idx,]
test.red$quality <- as.factor(test.red$quality)
test.red.quality <- test.red$quality
# test.red <- test.red[,-12]

# # Backward model selection
# library(leaps)
# regfit.bwd=regsubsets(quality~.,data=train.red,nvmax=11,method="backward")
# summary(regfit.bwd)
#
# test.mat=model.matrix(quality~.,data=test.red) # create an X matrix of test data
# val.errors=rep(NA,19)
# for(i in 1:19){
#    coefi=coef(regfit.best,id=i)
#    pred=test.mat[,names(coefi)]%*%coefi
#    val.errors[i]=mean((Hitters$Salary[test]-pred)^2)
# }
# val.errors
# which.min(val.errors)
# coef(regfit.best,10)

set.seed(1)

gamma <- 2^seq(-15, 3, 1)
test_accuracy <- rep(0, length(gamma))
fit_time <- rep(0, length(gamma))
i = 1

start <- proc.time()

for(g in gamma){
  start_it <- proc.time()
  model = svm(quality~., data = train.red, gamma=g)
  fit_time[i] <- proc.time() - start_it
  test_accuracy[i] <- mean(test.red.quality == predict(model, test.red))
  i = i + 1
}
```

```
number of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement length
```

Hide

```
print(cv_time_red <- proc.time() - start) # total CV time
```

```
   user  system elapsed 
  3.955   0.095   4.214 
```

Hide

```
print(fit_time[which.max(test_accuracy)]) # fit time of best model
```

```
[1] 0.15
```

Hide

```
print(best_gamma_red <- 2^(which.max(test_accuracy) - 16)) # gamma yielding highest test accuracy
```

```
[1] 0.25
```

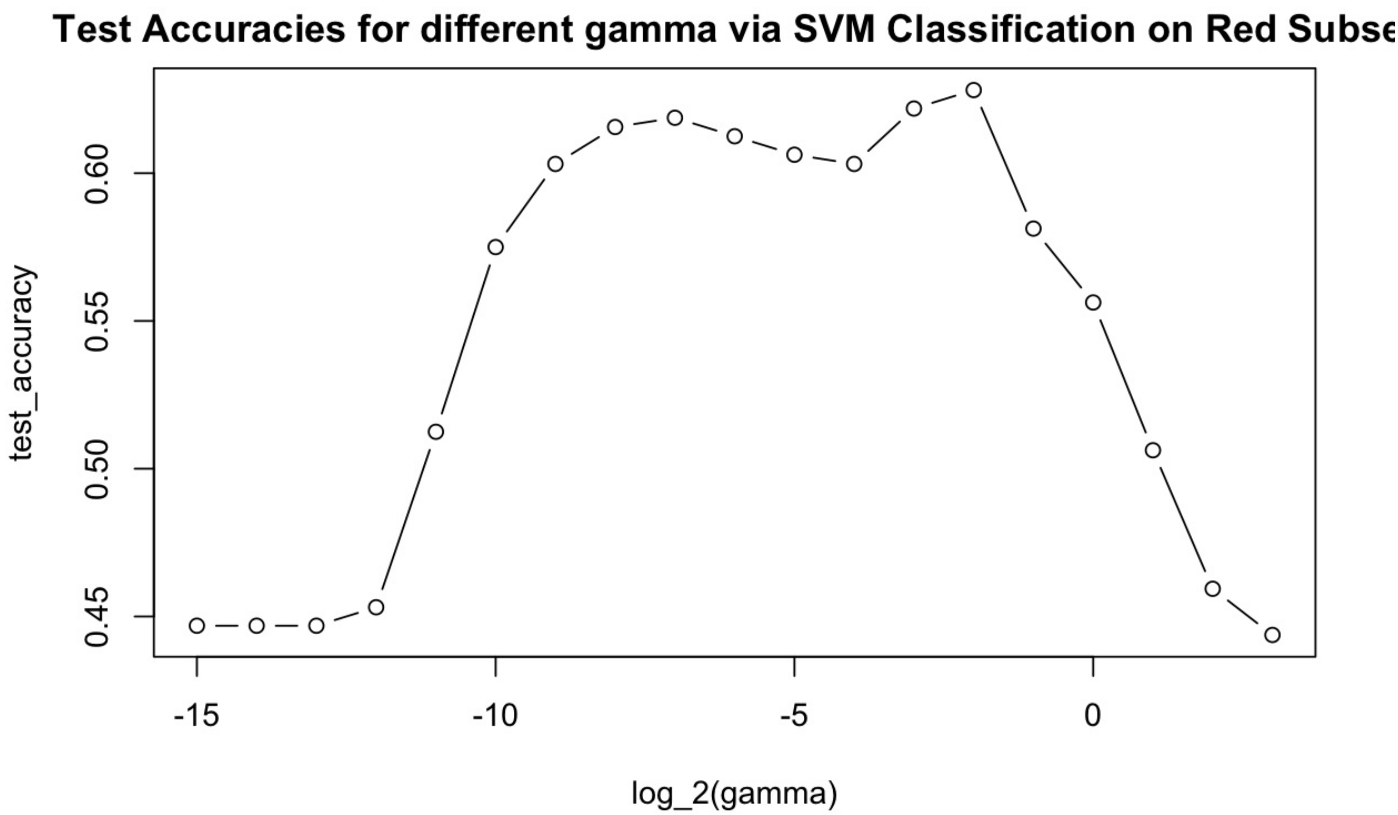
Hide

```
print(test_acc_red <- max(test_accuracy)) # test accuracy
```

```
[1] 0.628125
```

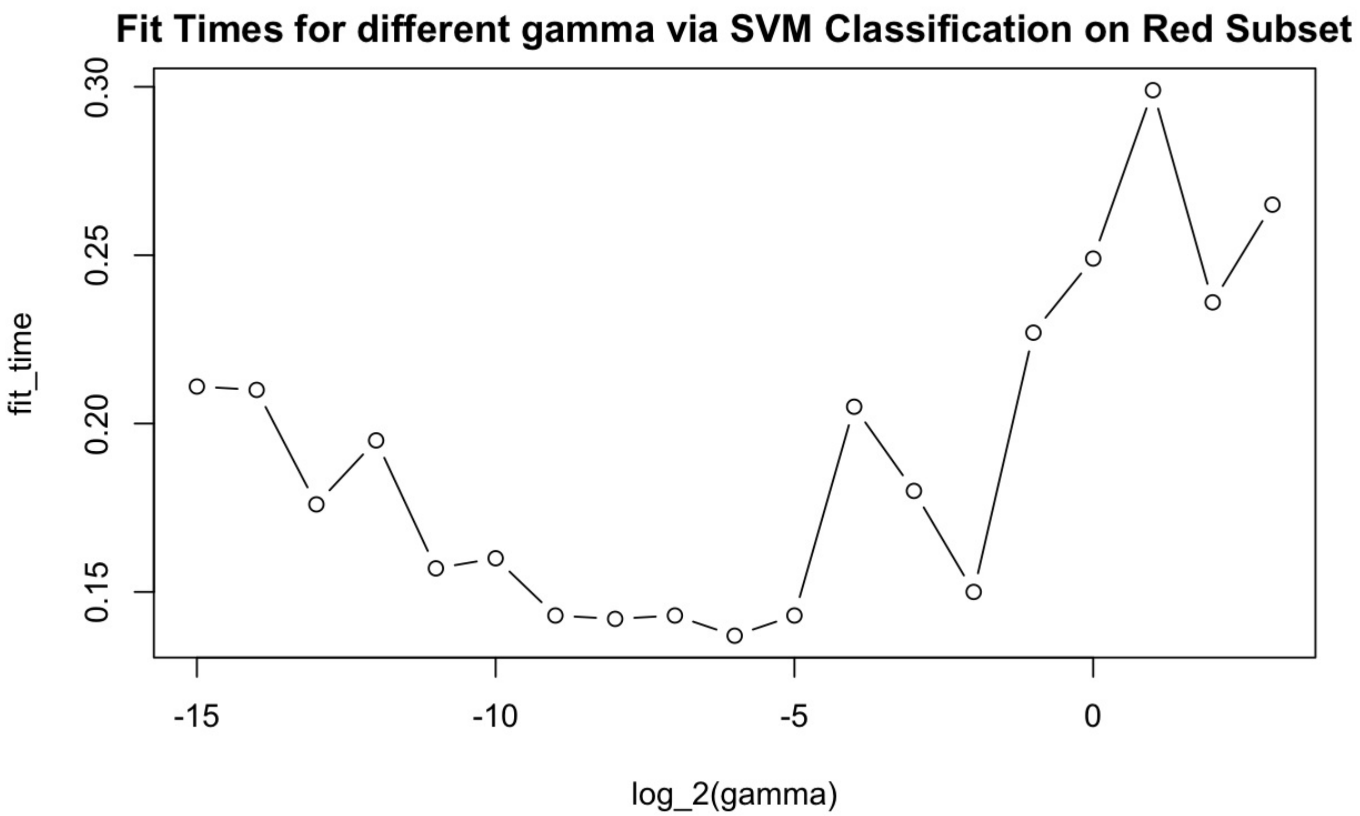
Hide

```
plot(log2(gamma), test_accuracy, type='b', xlab = 'log_2(gamma)', main='Test Accuracies for different gamma via SVM Classification on Red Subset')
```



Hide

```
plot(log2(gamma), fit_time, type='b', xlab = 'log_2(gamma)', main='Fit Times for different gamma via SVM Classifi
cation on Red Subset')
```



Hide

```
NA
NA
```

White dataset

Hide

```
library(kernlab)
library(e1071)
```

```
package ‘e1071’ was built under R version 3.6.2
```

Hide

```
library(MASS)

white <- read.csv('winequality-white.csv', header = TRUE, sep=";")
white <- na.omit(white)
white.quality <- white$quality
white[,-12] <- scale(white[,-12])

set.seed(1)
train_white_idx <- sample(nrow(white) * 0.8) # 80-20 train-test split
train.white <- white[train_white_idx,]
train.white$quality <- as.factor(train.white$quality)
train.white.quality <- train.white$quality

test.white <- white[-train_white_idx,]
test.white$quality <- as.factor(test.white$quality)
test.white.quality <- test.white$quality
# test.white <- test.white[,-12]

# # Backward model selection
# library(leaps)
# regfit.bwd=regsubsets(quality~.,data=train.white,nvmax=11,method="backward")
# summary(regfit.bwd)
#
# test.mat=model.matrix(quality~.,data=test.white) # create an X matrix of test data
# val.errors=rep(NA,19)
# for(i in 1:19){
#    coefi=coef(regfit.best,id=i)
#    pred=test.mat[,names(coefi)]%*%coefi
#    val.errors[i]=mean((Hitters$Salary[test]-pred)^2)
# }
# val.errors
# which.min(val.errors)
# coef(regfit.best,10)

set.seed(1)

gamma <- 2^seq(-15, 3, 1)
test_accuracy <- rep(0, length(gamma))
fit_time <- rep(0, length(gamma))
i = 1

start <- proc.time()

for(g in gamma){
  start_it <- proc.time()
  model = svm(quality~., data = train.white, gamma=g)
  fit_time[i] <- proc.time() - start_it
  test_accuracy[i] <- mean(as.character(test.white.quality) == as.character(predict(model, test.white)))
  i = i + 1
}
```

```
number of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of
replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is
not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of i
tems to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacemen
t lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a mul
tiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to r
eplace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthn
umber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of
replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is
not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of i
tems to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacemen
t length
```

Hide

```
print(cv_time_white <- proc.time() - start) # total CV time
```

```
   user  system elapsed
 36.059   0.505  37.336
```

Hide

```
print(fit_time[which.max(test_accuracy)]) # fit time of best model
```

```
[1] 1.668
```

Hide

```
print(best_gamma_white <- 2^(which.max(test_accuracy) - 16)) # gamma yielding highest test accuracy
```

```
[1] 0.015625
```

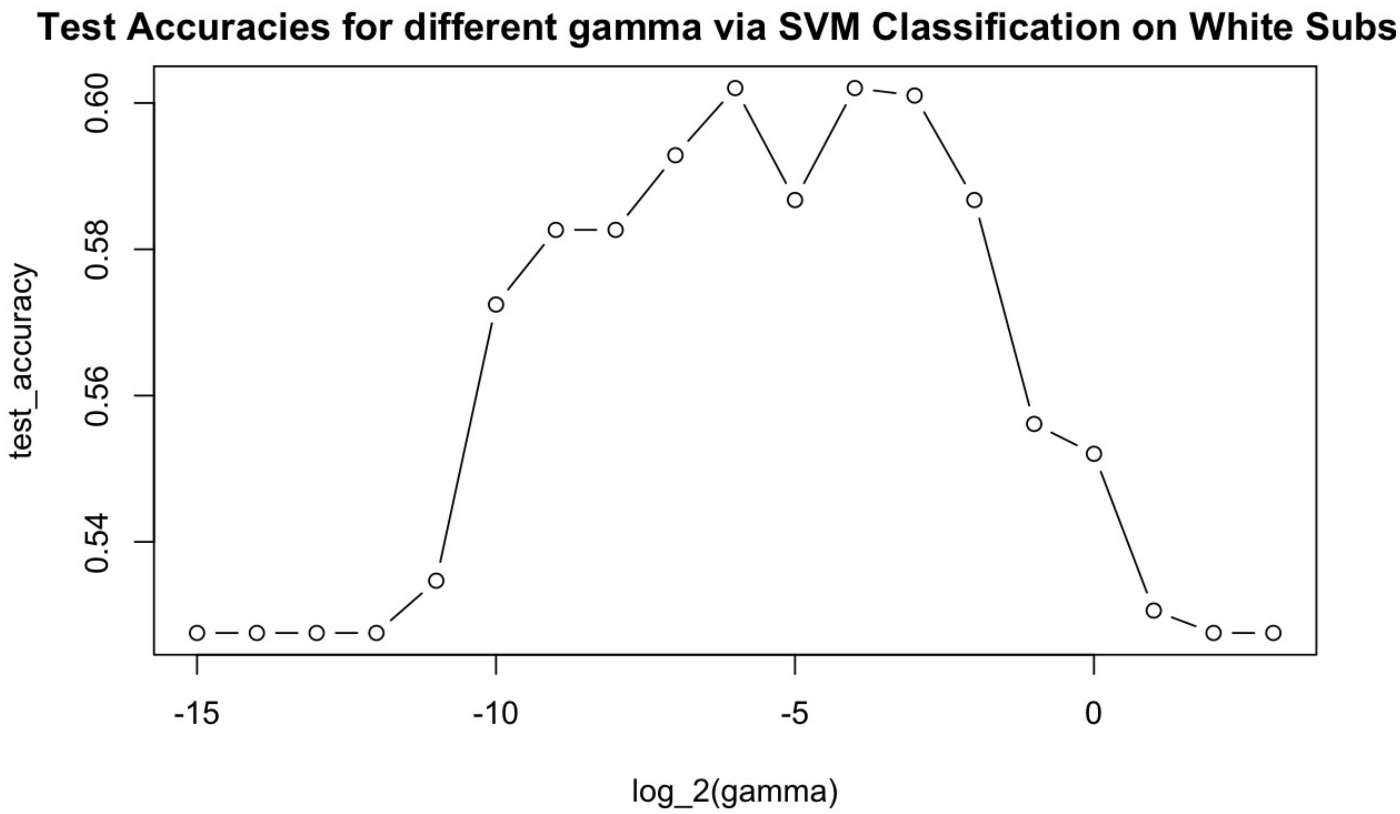
Hide

```
print(test_acc_white <- max(test_accuracy)) # test accuracy
```

```
[1] 0.6020408
```

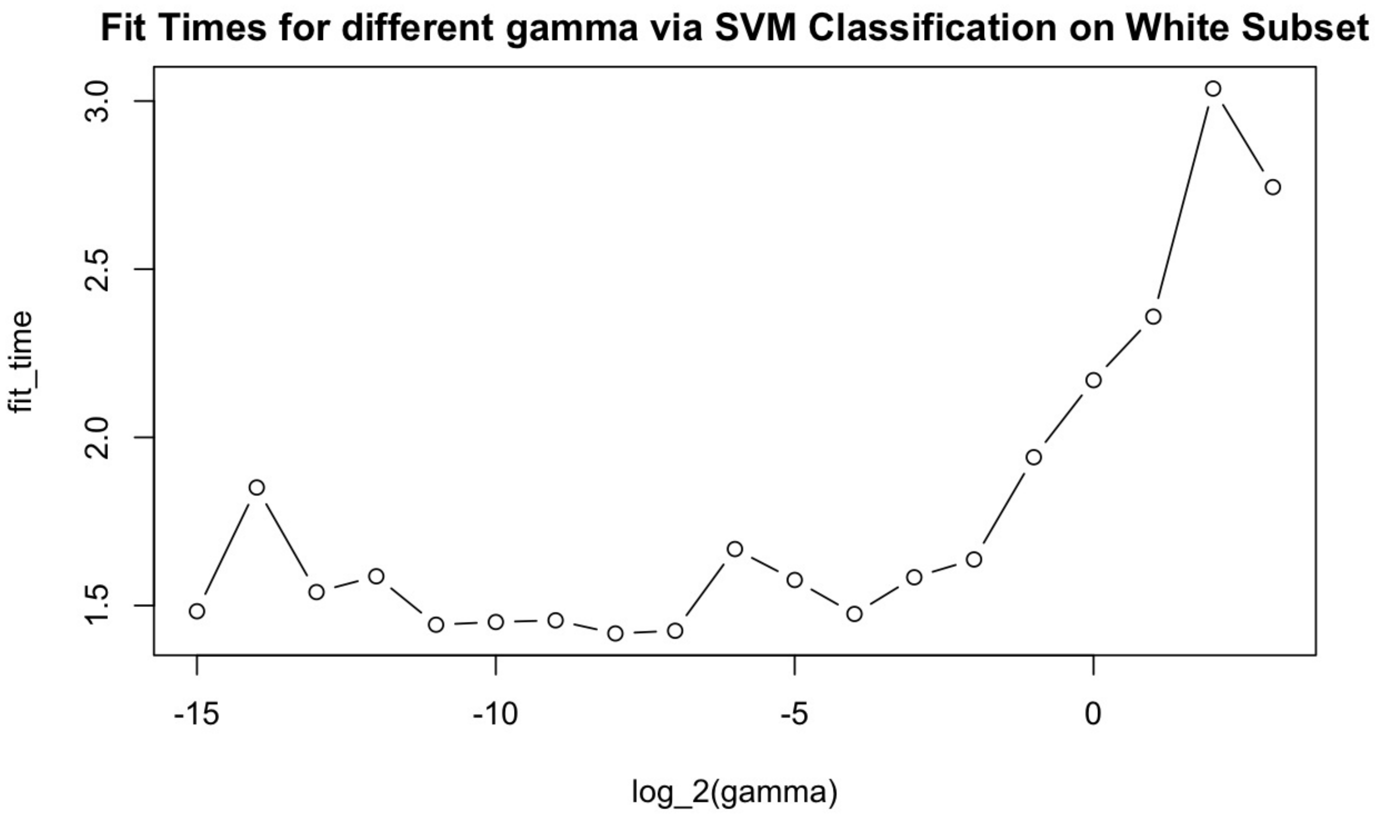
Hide

```
plot(log2(gamma), test_accuracy, type='b', xlab = 'log_2(gamma)', main='Test Accuracies for different gamma via S
VM Classification on White Subset')
```



Hide

```
plot(log2(gamma), fit_time, type='b', xlab = 'log_2(gamma)', main='Fit Times for different gamma via SVM Classifi
cation on White Subset')
```



Weighted Test Accuracy

Hide

```
(nrow(test.red) * test_acc_red + nrow(test.white) * test_acc_white) / (nrow(test.red) + nrow(test.white))
```

```
[1] 0.6084615
```

— SVM regression (epsilon-insensitive loss) —

Red dataset

Hide

```
set.seed(1)
red <- read.csv('winequality-red.csv', header = TRUE, sep=";")
red <- na.omit(red)
red.quality <- red$quality
red[,-12] <- scale(red[,-12])

# Red dataset
train_red_idx <- sample(nrow(red) * 0.8) # 80-20 train-test split
train.red <- red[train_red_idx,]
# train.red$quality <- as.factor(train.red$quality)
train.red.quality <- train.red$quality

test.red <- red[-train_red_idx,]
# test.red$quality <- as.factor(test.red$quality)
test.red.quality <- test.red$quality
# test.red <- test.red[,-12]

# # Backward model selection
# library(leaps)
# regfit.bwd=regsubsets(quality~.,data=train.red,nvmax=11,method="backward")
# summary(regfit.bwd)
#
# test.mat=model.matrix(quality~.,data=test.red) # create an X matrix of test data
# val.errors=rep(NA,19)
# for(i in 1:19){
#    coefi=coef(regfit.best,id=i)
#    pred=test.mat[,names(coefi)]%*%coefi
#    val.errors[i]=mean((Hitters$Salary[test]-pred)^2)
# }
# val.errors
# which.min(val.errors)
# coef(regfit.best,10)

set.seed(1)

gamma <- 2^seq(-15, 3, 1)
test_accuracy <- rep(0, length(gamma))
fit_time <- rep(0, length(gamma))
i = 1

start <- proc.time()

for(g in gamma){
  start_it <- proc.time()
  model = svm(quality~., data = train.red, gamma=g, type='eps-regression')
  fit_time[i] <- proc.time() - start_it
  test_accuracy[i] <- mean(abs(test.red.quality - predict(model, test.red)) <= 0.5)
  i = i + 1
}
```

```
number of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of
replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is
not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of i
tems to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacemen
t lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a mul
tiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to r
eplace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthn
umber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of
replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is
not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of i
tems to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacemen
t length
```

Hide

```
print(cv_time_red <- proc.time() - start) # total CV time
```

```
   user  system elapsed
  3.205   0.057   3.267
```

Hide

```
print(fit_time[which.max(test_accuracy)]) # fit time of best model
```

```
[1] 0.128
```

Hide

```r
print(best_gamma_red <- 2^(which.max(test_accuracy) - 16)) # gamma yielding highest test accuracy
```

```
[1] 0.001953125
```

Hide

```r
print(test_acc_red <- max(test_accuracy)) # test accuracy
```

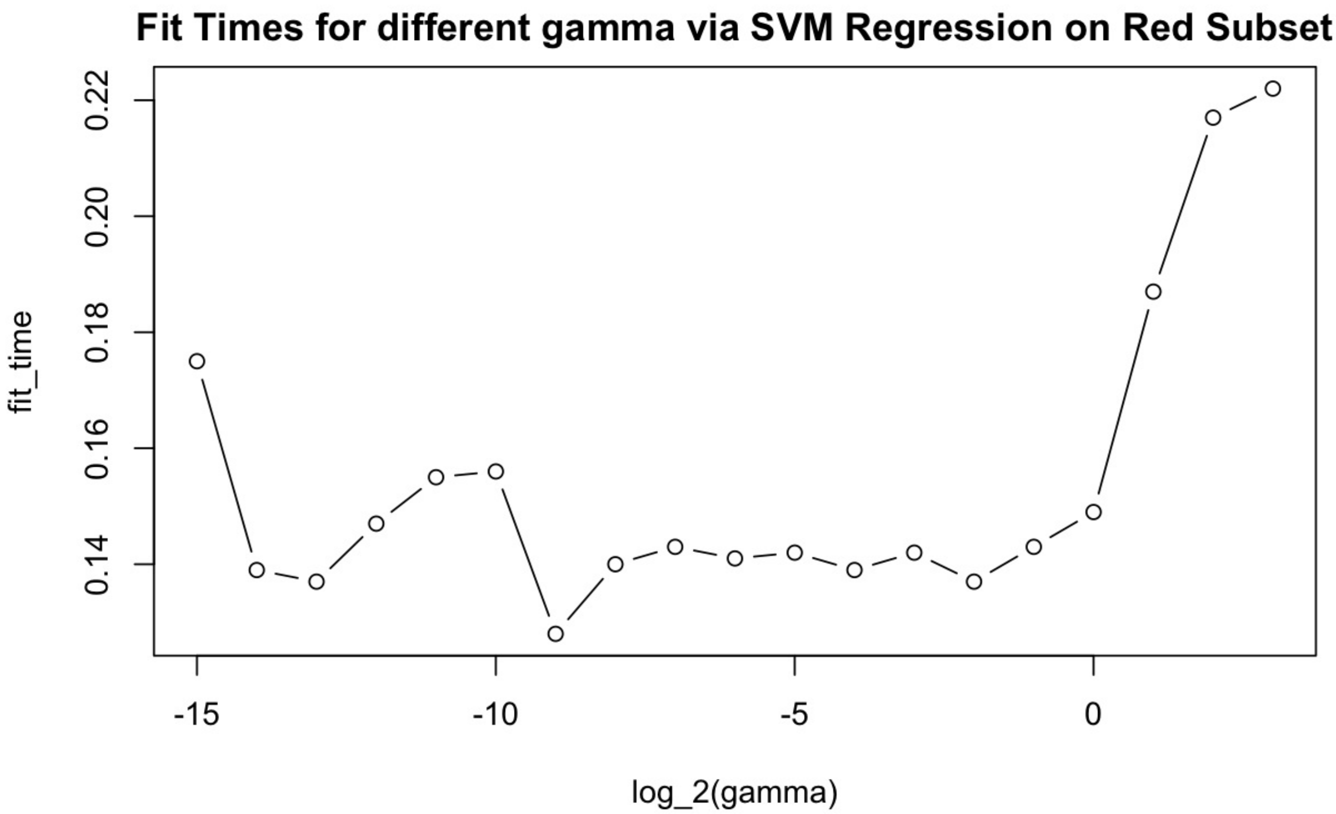
```
[1] 0.640625
```

Hide

```r
plot(log2(gamma), test_accuracy, type='b', xlab = 'log_2(gamma)', main='Test Accuracies for different gamma via S
VM Regression on Red Subset')
```

Hide

```r
plot(log2(gamma), fit_time, type='b', xlab = 'log_2(gamma)', main='Fit Times for different gamma via SVM Regressi
on on Red Subset')
```



White dataset

Hide

```
white <- read.csv('winequality-white.csv', header = TRUE, sep=";")
white <- na.omit(white)
white.quality <- white$quality
white[,-12] <- scale(white[,-12])

set.seed(1)
train_white_idx <- sample(nrow(white) * 0.8) # 80-20 train-test split
train.white <- white[train_white_idx,]
# train.white$quality <- as.factor(train.white$quality)
train.white.quality <- train.white$quality

test.white <- white[-train_white_idx,]
# test.white$quality <- as.factor(test.white$quality)
test.white.quality <- test.white$quality
# test.white <- test.white[,-12]

# # Backward model selection
# library(leaps)
# regfit.bwd=regsubsets(quality~.,data=train.white,nvmax=11,method="backward")
# summary(regfit.bwd)
#
# test.mat=model.matrix(quality~.,data=test.white) # create an X matrix of test data
# val.errors=rep(NA,19)
# for(i in 1:19){
#    coefi=coef(regfit.best,id=i)
#    pred=test.mat[,names(coefi)]%*%coefi
#    val.errors[i]=mean((Hitters$Salary[test]-pred)^2)
# }
# val.errors
# which.min(val.errors)
# coef(regfit.best,10)

set.seed(1)

gamma <- 2^seq(-15, 3, 1)
test_accuracy <- rep(0, length(gamma))
fit_time <- rep(0, length(gamma))
i = 1

start <- proc.time()

for(g in gamma){
  start_it <- proc.time()
  model = svm(quality~., data = train.white, gamma=g, type='eps-regression')
  fit_time[i] <- proc.time() - start_it
  test_accuracy[i] <- mean(abs(test.white.quality - predict(model, test.white)) <= 0.5)
  i = i + 1
}
```

```
number of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of
replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is
not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of i
tems to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacemen
t lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a mul
tiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to r
eplace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthn
umber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of
replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is
not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of i
tems to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacemen
t length
```

Hide

```
print(cv_time_white <- proc.time() - start) # total CV time
```

```
   user  system elapsed
 27.852   0.369  28.544
```

Hide

```
print(fit_time[which.max(test_accuracy)]) # fit time of best model
```

```
[1] 1.276
```

Hide

```
print(best_gamma_white <- 2^(which.max(test_accuracy) - 16)) # gamma yielding highest test accuracy
```

```
[1] 0.125
```

Hide

```
print(test_acc_white <- max(test_accuracy)) # test accuracy
```

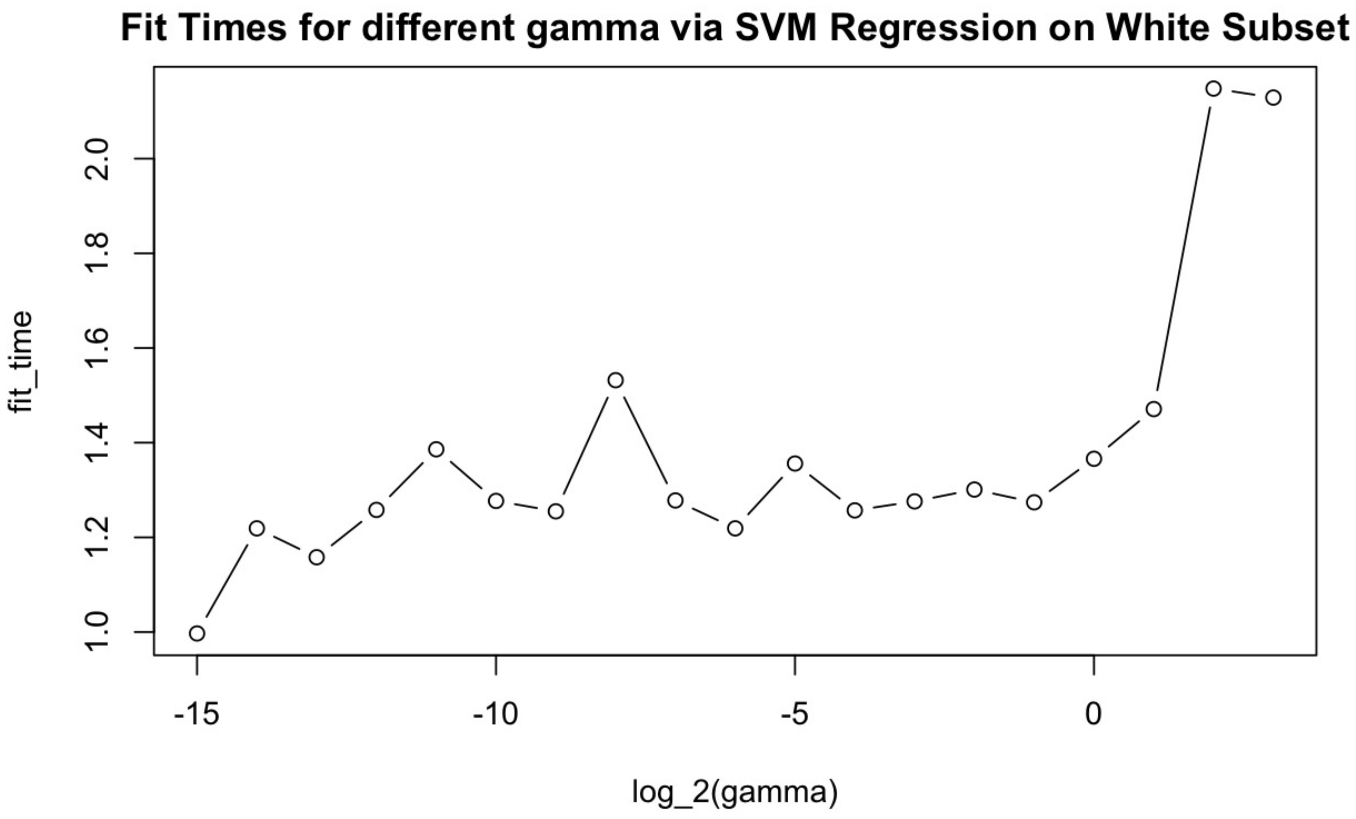
```
[1] 0.5836735
```

Hide

```
plot(log2(gamma), test_accuracy, type='b', xlab = 'log_2(gamma)', main='Test Accuracies for different gamma via S
VM Regression on White Subset')
```

Hide

```
plot(log2(gamma), fit_time, type='b', xlab = 'log_2(gamma)', main='Fit Times for different gamma via SVM Regressi
on on White Subset')
```



Weighted Test Accuracy

Hide

```
(nrow(test.red) * test_acc_red + nrow(test.white) * test_acc_white) / (nrow(test.red) + nrow(test.white))
```

```
[1] 0.5976923
```

When you save the notebook, an HTML file containing the code and output will be saved alongside it (click the *Preview* button or press *Cmd+Shift+K* to preview the HTML file).

The preview shows you a rendered HTML copy of the contents of the editor. Consequently, unlike *Knit*, *Preview* does not run any R code chunks. Instead, the output of the chunk when it was last run in the editor is displayed.

When you save the notebook, an HTML file containing the code and output will be saved alongside it (click the *Preview* button or press *Cmd+Shift+K* to preview the HTML file).

The preview shows you a rendered HTML copy of the contents of the editor. Consequently, unlike *Knit*, *Preview* does not run any R code chunks. Instead, the output of the chunk when it was last run in the editor is displayed.

R Notebook

Code

This is an R Markdown (http://rmarkdown.rstudio.com) Notebook. When you execute code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Cmd+Shift+Enter*.

Lasso Regression

1. Use Split Dataset

Red Dataset:

Hide

```
library(glmnet)

# Red Datasets
red <- read.csv('winequality-red.csv', header = TRUE, sep=";")
red <- na.omit(red)
red.quality <- red$quality
red[,-12] <- scale(red[,-12]) # scale/standardization

train_red_idx <- sample(nrow(red) * 0.8) # 80-20 train-test split
train.red <- red[train_red_idx,]
train.red.quality <- train.red$quality
train.x.red <- model.matrix(quality~., train.red)

test.red <- red[-train_red_idx,]
test.red.quality <- test.red$quality
test.x.red <- model.matrix(quality~., test.red)

# Ridge Regression
start = proc.time()
cv.ridge = cv.glmnet(train.x.red, train.red.quality, alpha=0)
proc.time() - start
```

```
   user  system elapsed
  0.180   0.020   0.203
```

Hide

```
par(mfrow=c(1,1))
cv.ridge
```

```


Call:  cv.glmnet(x = train.x.red, y = train.red.quality, alpha = 0)

Measure: Mean-Squared Error

    Lambda Index Measure      SE Nonzero
min 0.0406   100  0.4225 0.01700      11
1se 0.4556    74  0.4391 0.01472      11
```

Hide

```
plot(cv.ridge, main='Test MSE vs Lambda (Ridge, Red)')

bestlam.red = cv.ridge$lambda.min
bestlam.red
```

```
[1] 0.04055818
```

Hide

```
start = proc.time()
ridge.best.red = glmnet(train.x.red, train.red.quality, alpha=0, lambda=bestlam.red)
proc.time() - start
```

```
   user  system elapsed
  0.021   0.002   0.024
```

Hide

```
ridge.pred.red = predict(ridge.best.red, test.x.red)
test_acc.red.ridge <- mean(abs(ridge.pred.red - test.red.quality) <= 0.5) # test accuracy (prediction within 0.5
of ground truth)
test_acc.red.ridge
```

```
[1] 0.640625
```

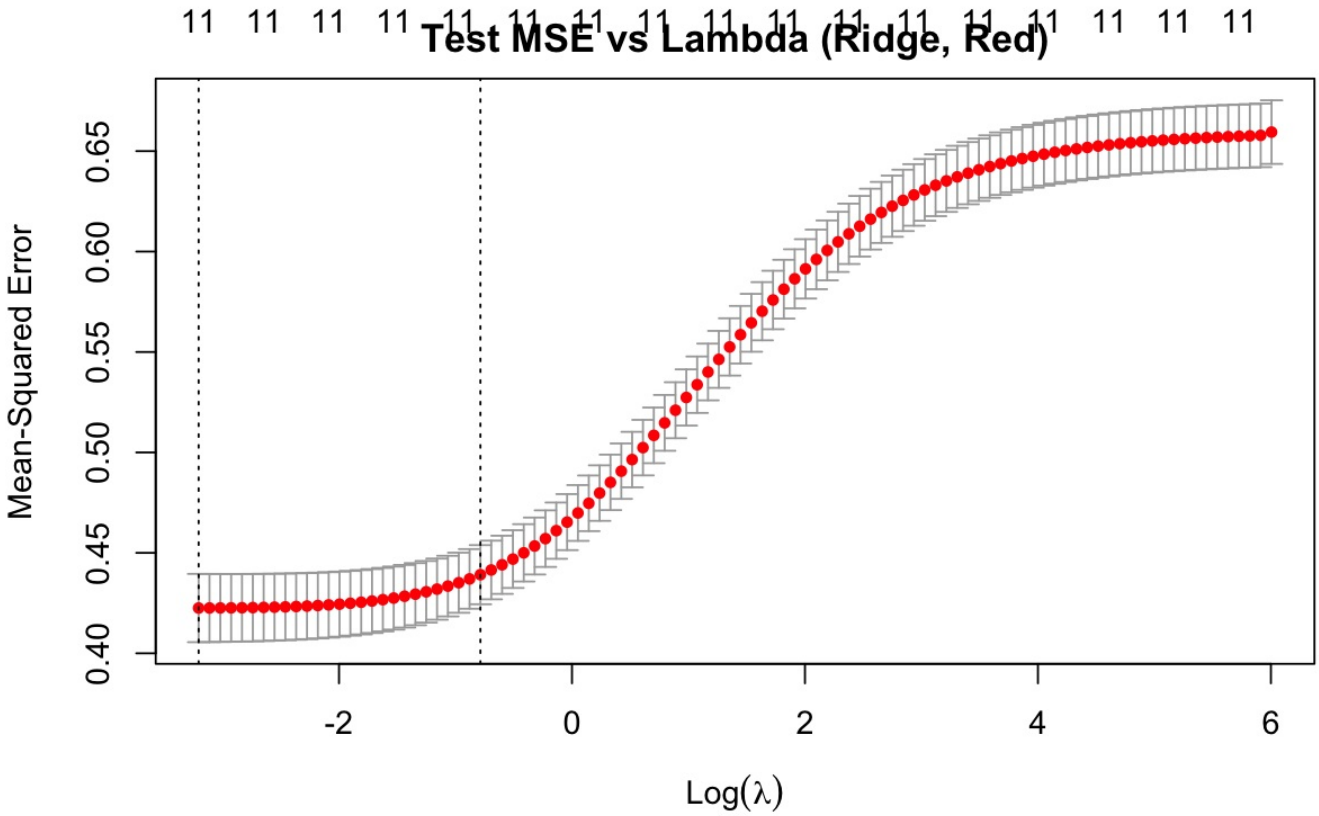
Hide

```
# Lasso Regression
start = proc.time()
cv.lasso = cv.glmnet(train.x.red, train.red.quality, alpha=1)
proc.time() - start
```

```
   user  system elapsed
  0.115   0.010   0.131
```

Hide

```
par(mfrow=c(1,1))
```



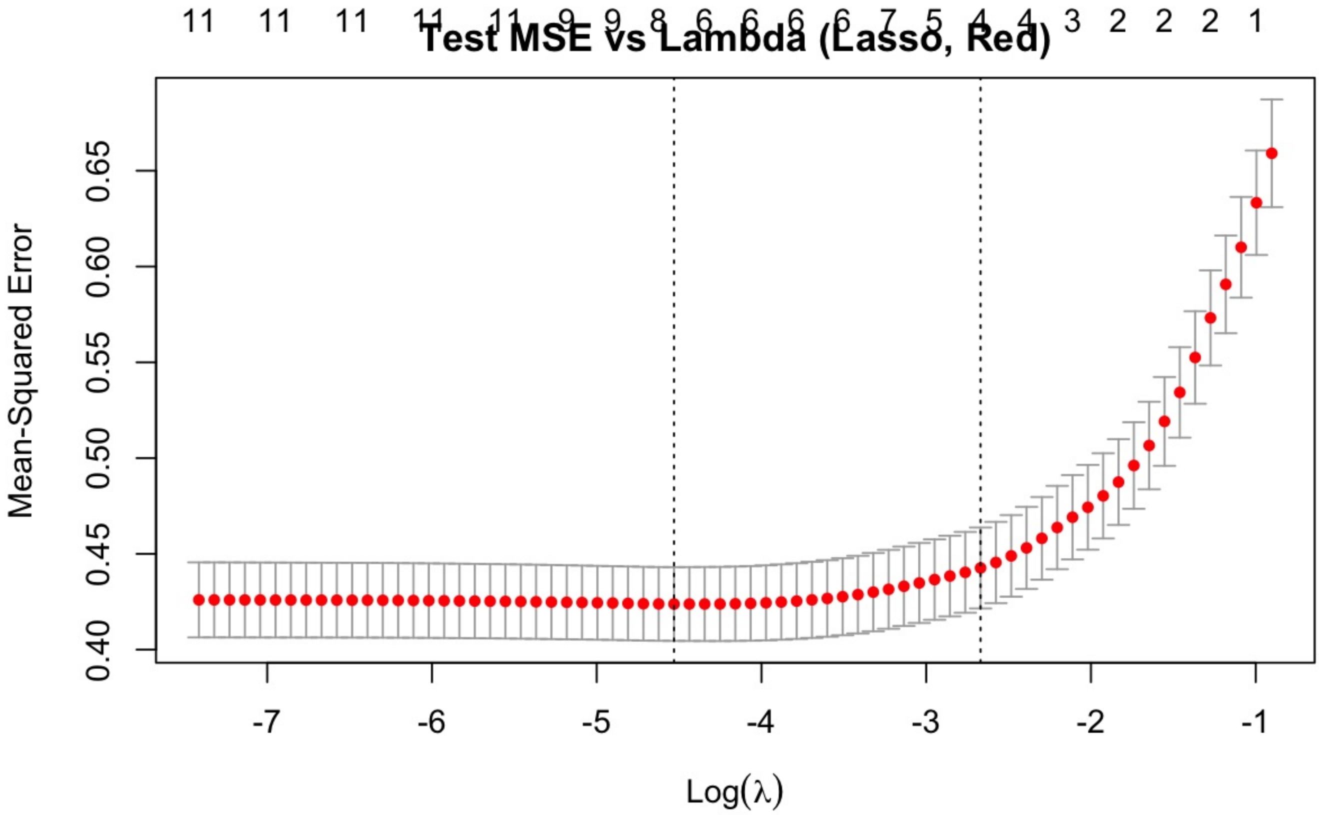
Hide

```
cv.lasso
```

```

Call:  cv.glmnet(x = train.x.red, y = train.red.quality, alpha = 1)

Measure: Mean-Squared Error

     Lambda Index Measure      SE Nonzero
min 0.01077    40  0.4238 0.01925       8
1se 0.06925    20  0.4426 0.02113       4
```

Hide

```
plot(cv.lasso, main='Test MSE vs Lambda (Lasso, Red)')
```



Hide

```
bestlam.red = cv.lasso$lambda.min
bestlam.red
```

```
[1] 0.01077261
```

Hide

```
start = proc.time()
lasso.best.red = glmnet(train.x.red, train.red.quality, alpha=1, lambda=bestlam.red)
proc.time() - start
```

```
   user  system elapsed 
  0.021   0.003   0.023 
```

Hide

```
lasso.pred.red = predict(lasso.best.red, test.x.red)
test_acc.red.lasso <-  mean(abs(lasso.pred.red - test.red.quality) <= 0.5) # test accuracy (prediction within 0.5 of ground truth)
test_acc.red.lasso
```

```
[1] 0.646875
```

White Dataset:

Hide

```
white <- read.csv('winequality-white.csv', header = TRUE, sep=";")
white <- na.omit(white)
white.quality <- white$quality
white[,-12] <- scale(white[,-12]) # scale/standardization

set.seed(1)
train_white_idx <- sample(nrow(white) * 0.8) # 80-20 train-test split
train.white <- white[train_white_idx,]
# train.white$quality <- as.factor(train.white$quality)
train.white.quality <- train.white$quality
train.x.white <- model.matrix(quality~., train.white)

test.white <- white[-train_white_idx,]
# test.white$quality <- as.factor(test.white$quality)
test.white.quality <- test.white$quality
test.x.white <- model.matrix(quality~., test.white)

# Ridge Regression
start = proc.time()
cv.ridge = cv.glmnet(train.x.white, train.white.quality, alpha=0)
proc.time() - start
```

```
   user  system elapsed 
  0.294   0.032   0.383 
```

Hide

```
par(mfrow=c(1,1))
cv.ridge
```

```


Call:  cv.glmnet(x = train.x.white, y = train.white.quality, alpha = 0) 

Measure: Mean-Squared Error 

     Lambda Index Measure      SE Nonzero
min 0.04141   100  0.5888 0.01888      11
1se 0.26619    80  0.6058 0.01844      11
```

Hide

```
plot(cv.ridge, main='Test MSE vs Lambda (Ridge, White)')

bestlam.white = cv.ridge$lambda.min
bestlam.white
```

```
[1] 0.04141091
```

Hide

```
start = proc.time()
ridge.best.white = glmnet(train.x.white, train.white.quality, alpha=0, lambda=bestlam.white)
proc.time() - start
```

```
   user  system elapsed 
  0.018   0.003   0.022 
```

Hide

```
ridge.pred.white = predict(ridge.best.white, test.x.white)
test_acc.white.ridge <- mean(abs(ridge.pred.white - test.white.quality) <= 0.5) # test accuracy (prediction withi
n 0.5 of ground truth)
test_acc.white.ridge
```

```
[1] 0.5408163
```

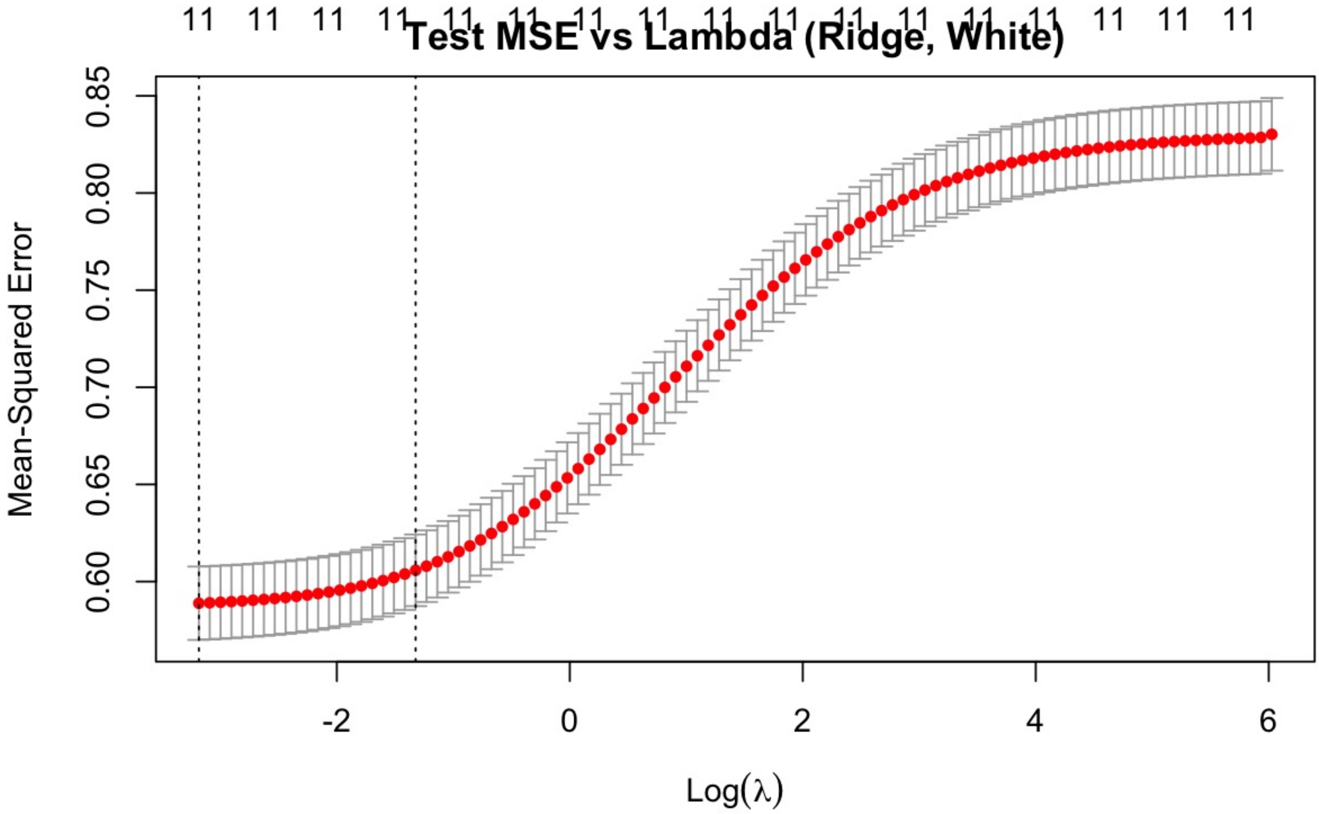
Hide

```
# Lasso Regression
start = proc.time()
cv.lasso = cv.glmnet(train.x.white, train.white.quality, alpha=1)
proc.time() - start
```

```
   user  system elapsed 
  0.406   0.032   0.517 
```

Hide

```
par(mfrow=c(1,1))
```



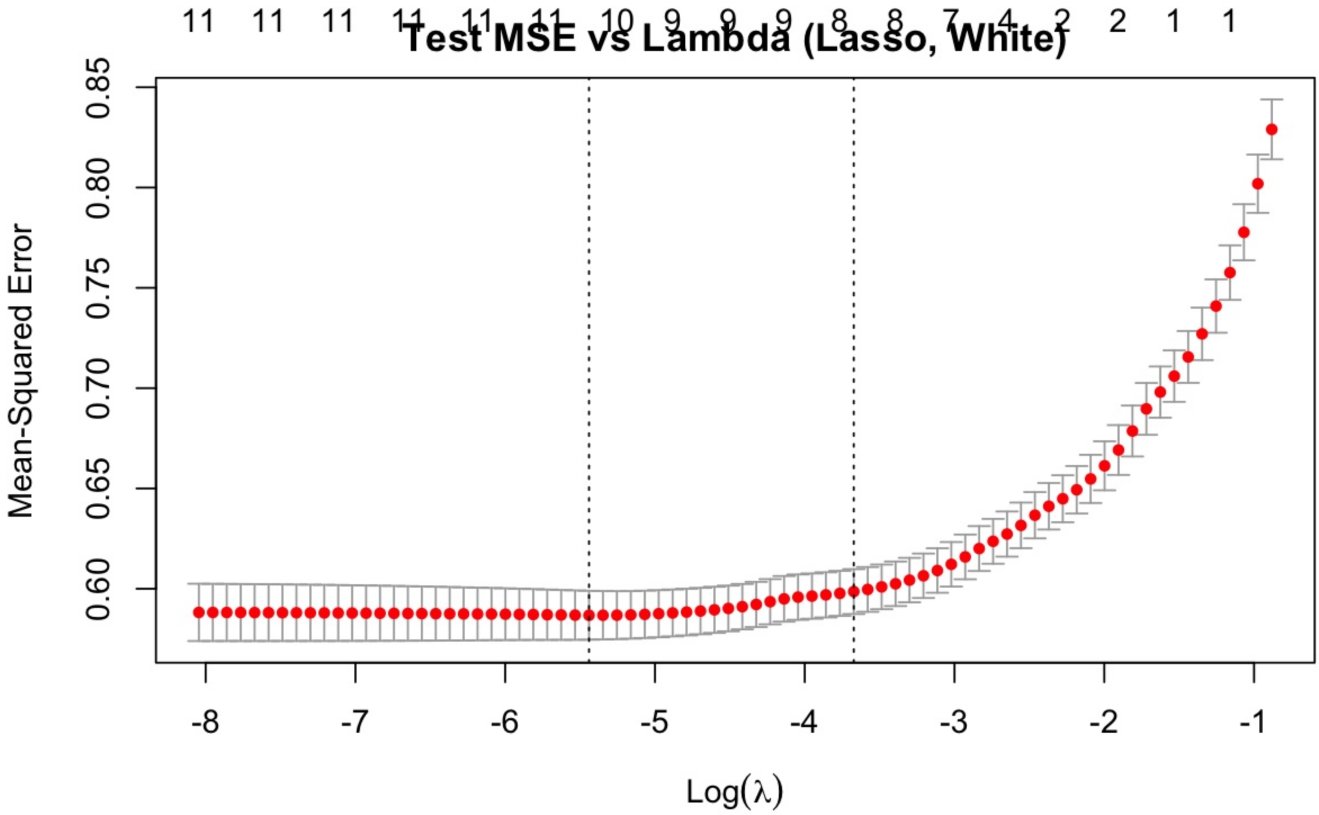
Hide

```
cv.lasso
```

```

Call:  cv.glmnet(x = train.x.white, y = train.white.quality, alpha = 1) 

Measure: Mean-Squared Error 

      Lambda Index Measure      SE Nonzero
min 0.004338    50  0.5868 0.01220      10
1se 0.025409    31  0.5986 0.01119       8
```

Hide

```
plot(cv.lasso, main='Test MSE vs Lambda (Lasso, White)')
```



Hide

```
bestlam.white = cv.lasso$lambda.min
bestlam.white
```

```
[1] 0.004338272
```

Hide

```
start = proc.time()
lasso.best.white = glmnet(train.x.white, train.white.quality, alpha=1, lambda=bestlam.white)
proc.time() - start
```

```
   user  system elapsed 
  0.021   0.004   0.034 
```

Hide

```
lasso.pred.white = predict(lasso.best.white, test.x.white)
test_acc.white.lasso <- mean(abs(lasso.pred.white - test.white.quality) <= 0.5) # test accuracy (prediction withi
n 0.5 of ground truth)
test_acc.white.lasso
```

```
[1] 0.5316327
```

Weighted Accuracy:

Hide

```
(test_acc.white.ridge * nrow(test.white) + test_acc.red.ridge * nrow(test.red)) / ( nrow(test.white) +  nrow(test
.red))
```

```
[1] 0.5653846
```

Hide

```
(test_acc.white.lasso * nrow(test.white) + test_acc.red.lasso * nrow(test.red)) / ( nrow(test.white) +  nrow(test
.red))
```

```
[1] 0.56
```

Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Cmd+Option+I*.

When you save the notebook, an HTML file containing the code and output will be saved alongside it (click the *Preview* button or press *Cmd+Shift+K* to preview the HTML file).

The preview shows you a rendered HTML copy of the contents of the editor. Consequently, unlike *Knit*, *Preview* does not run any R code chunks. Instead, the output of the chunk when it was last run in the editor is displayed.

R Notebook

Code

This is an R Markdown (http://rmarkdown.rstudio.com) Notebook. When you execute code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Cmd+Shift+Enter*.

— Correlations Between Variables —

To see why we get very low accuracies accross the board for this Wines dataset, let’s look at the correlation relationships between the predictors. Let’s also take a look at the class label distributions of the dataset (how balanced is it?).

Red Wines Dataset:

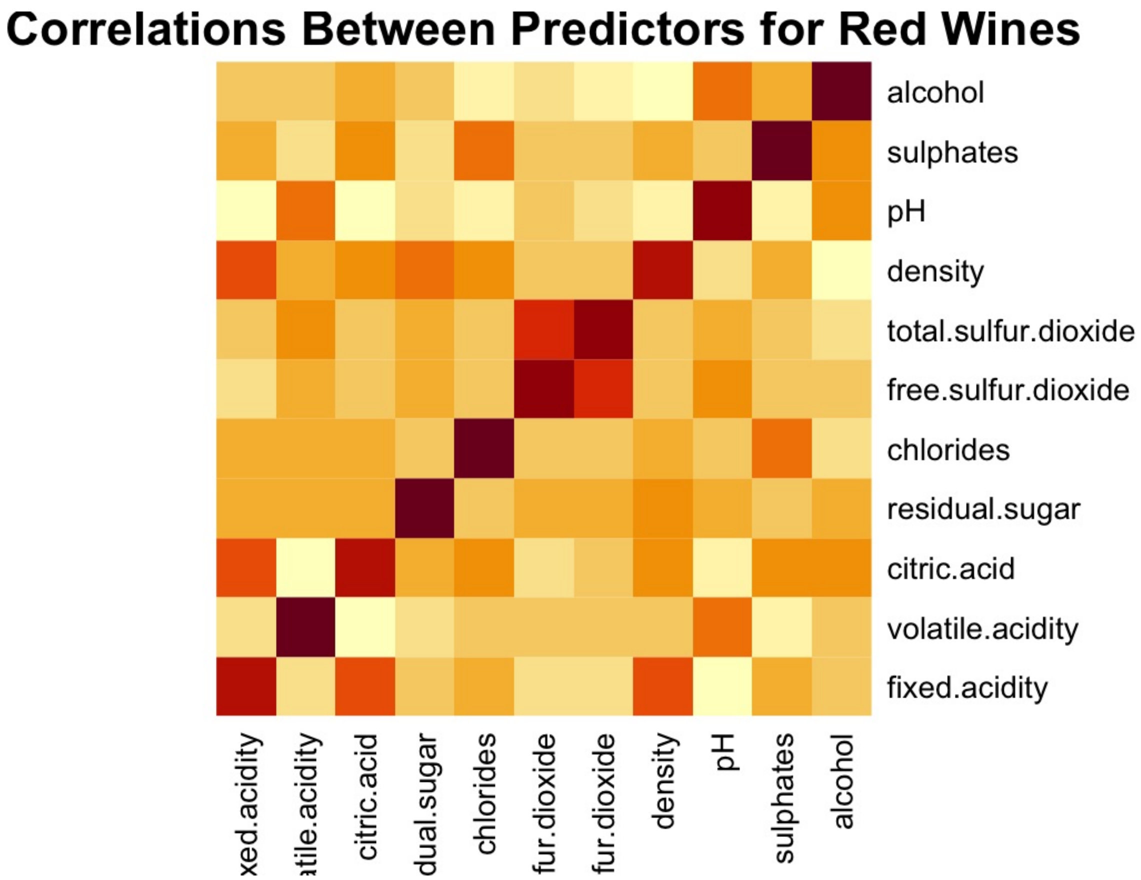
Hide

```
red <- read.csv('winequality-red.csv', header = TRUE, sep=";")
red <- na.omit(red)
red.quality <- red$quality
red[,-12] <- scale(red[,-12])

# Correlation matrix between variables
corr <- cor(red[,-12])
corr
```

```
                     fixed.acidity volatile.acidity citric.acid residual.sugar    chlorides free.sulfur.dioxide t
otal.sulfur.dioxide     density          pH
fixed.acidity           1.00000000     -0.256130895  0.67170343    0.114776724  0.093705186        -0.153794193
-0.11318144  0.66804729 -0.68297819
volatile.acidity       -0.25613089      1.000000000 -0.55249568    0.001917882  0.061297772        -0.010503827
0.07647000  0.02202623  0.23493729
citric.acid             0.67170343     -0.552495685  1.00000000    0.143577162  0.203822914        -0.060978129
0.03553302  0.36494718 -0.54190414
residual.sugar          0.11477672      0.001917882  0.14357716    1.000000000  0.055609535         0.187048995
0.20302788  0.35528337 -0.08565242
chlorides               0.09370519      0.061297772  0.20382291    0.055609535  1.000000000         0.005562147
0.04740047  0.20063233 -0.26502613
free.sulfur.dioxide    -0.15379419     -0.010503827 -0.06097813    0.187048995  0.005562147         1.000000000
0.66766645 -0.02194583  0.07037750
total.sulfur.dioxide   -0.11318144      0.076470005  0.03553302    0.203027882  0.047400468         0.667666450
1.00000000  0.07126948 -0.06649456
density                 0.66804729      0.022026232  0.36494718    0.355283371  0.200632327        -0.021945831
0.07126948  1.00000000 -0.34169933
pH                     -0.68297819      0.234937294 -0.54190414   -0.085652422 -0.265026131         0.070377499
-0.06649456 -0.34169933  1.00000000
sulphates               0.18300566     -0.260986685  0.31277004    0.005527121  0.371260481         0.051657572
0.04294684  0.14850641 -0.19664760
alcohol                -0.06166827     -0.202288027  0.10990325    0.042075437 -0.221140545        -0.069408354
-0.20565394 -0.49617977  0.20563251
                        sulphates     alcohol
fixed.acidity         0.183005664 -0.06166827
volatile.acidity     -0.260986685 -0.20228803
citric.acid           0.312770044  0.10990325
residual.sugar        0.005527121  0.04207544
chlorides             0.371260481 -0.22114054
free.sulfur.dioxide   0.051657572 -0.06940835
total.sulfur.dioxide  0.042946836 -0.20565394
density               0.148506412 -0.49617977
pH                   -0.196647602  0.20563251
sulphates             1.000000000  0.09359475
alcohol               0.093594750  1.00000000
```

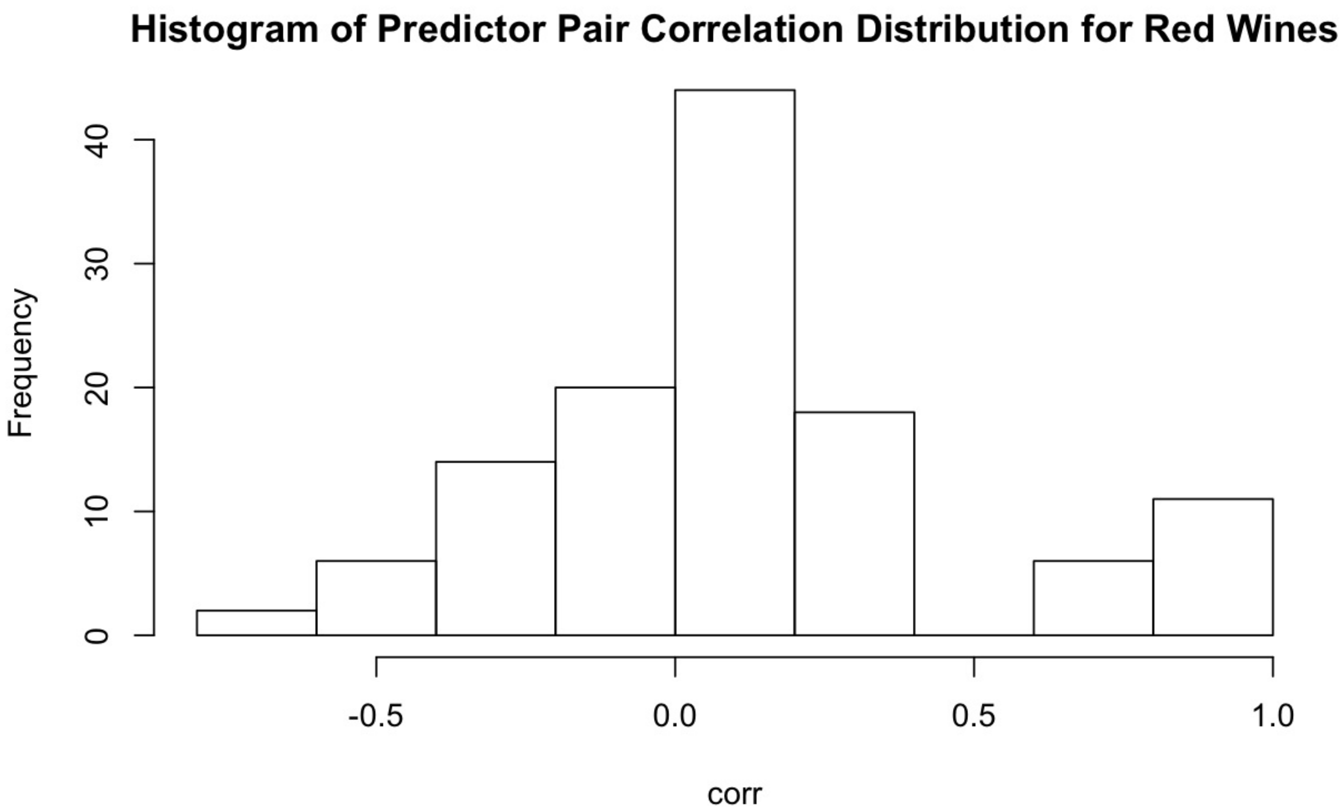
Hide

```
# Plot Heatmap
heatmap(corr, main="Correlations Between Predictors for Red Wines", Colv = NA, Rowv = NA, scale="column")
```



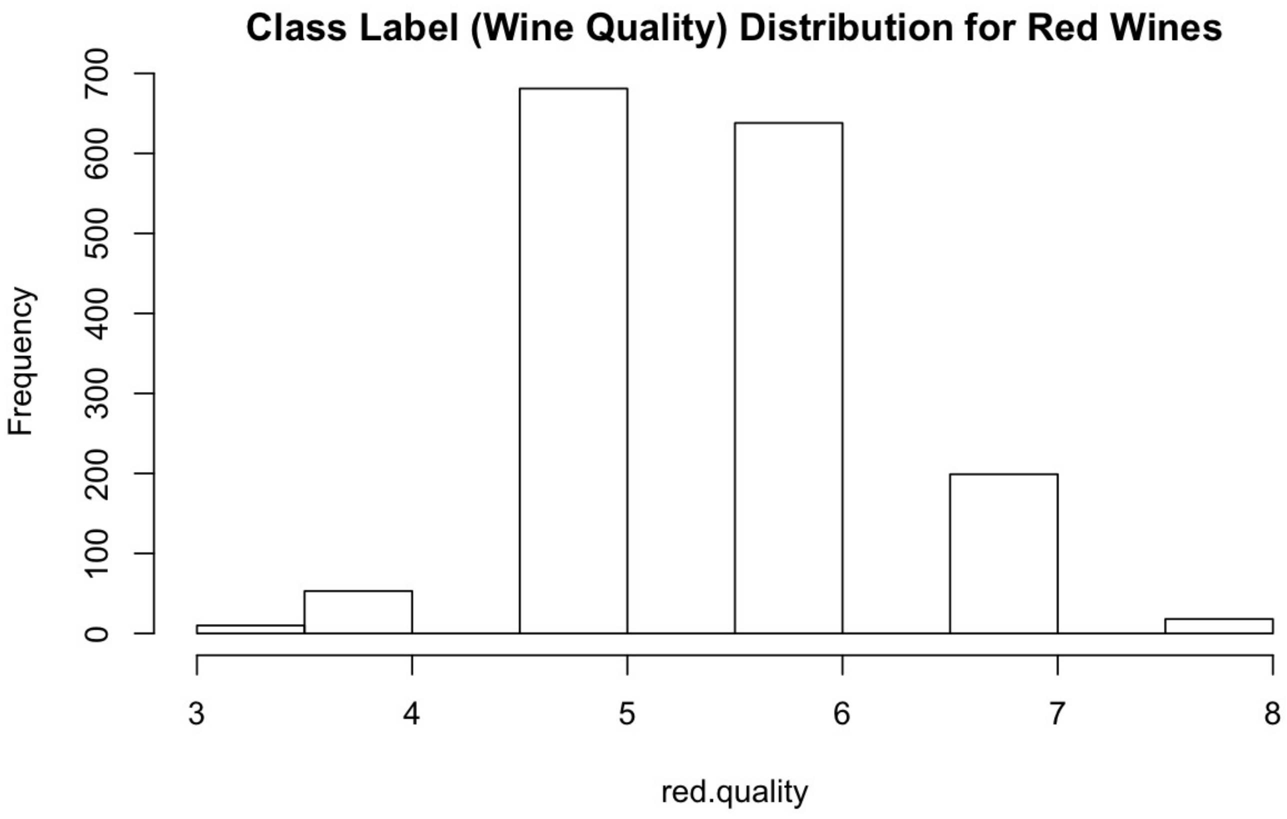
Hide

```
# Plot histogram
hist(corr, main='Histogram of Predictor Pair Correlation Distribution for Red Wines')
```



Hide

```
# Plot histogram between class labels
class_dist <- hist(red.quality, main='Class Label (Wine Quality) Distribution for Red Wines')
```



Hide

```
class_dist$counts
```

```
 [1]  10  53   0 681   0 638   0 199   0  18
```

White Wines Dataset:

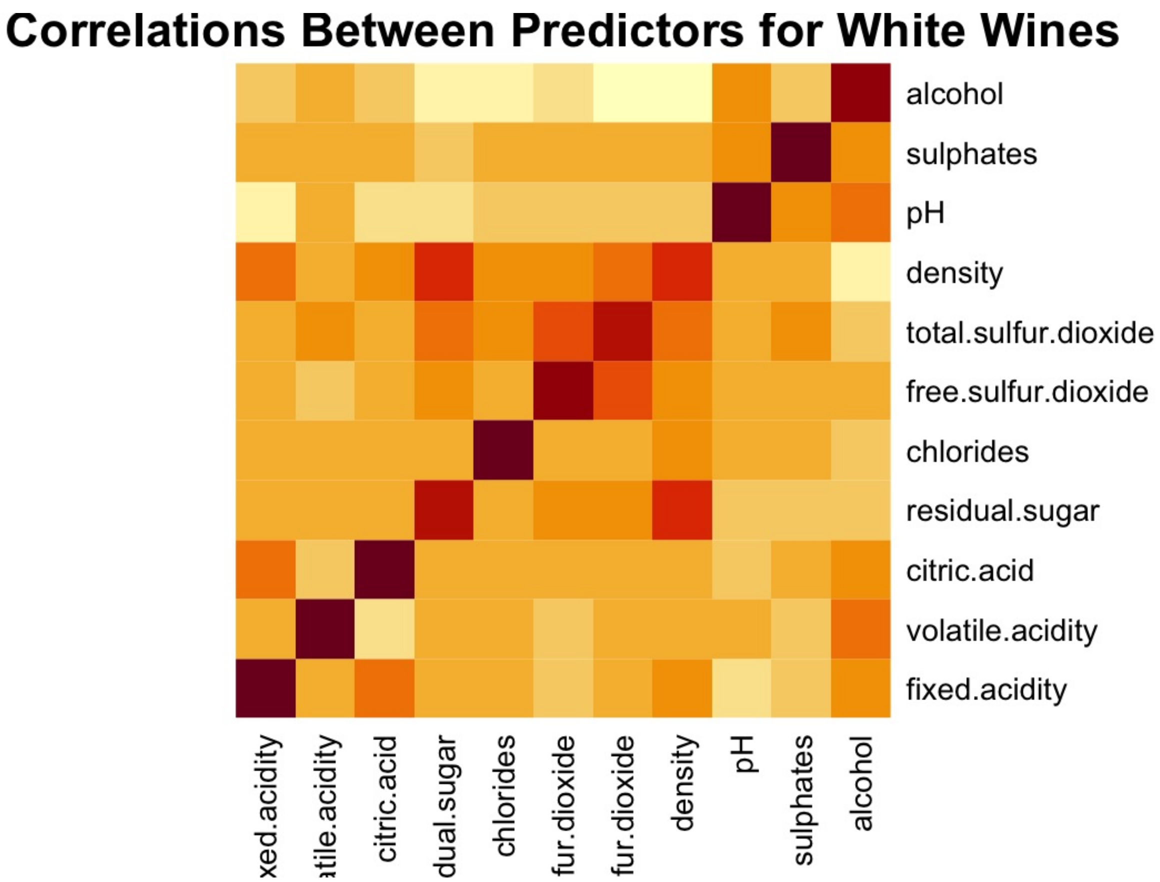
Hide

```
white <- read.csv('winequality-white.csv', header = TRUE, sep=";")
white <- na.omit(white)
white.quality <- white$quality
white[,-12] <- scale(white[,-12])

# Correlation matrix between variables
corr <- cor(white[,-12])
corr
```

```
                     fixed.acidity volatile.acidity citric.acid residual.sugar   chlorides free.sulfur.dioxide to
tal.sulfur.dioxide     density
fixed.acidity           1.00000000      -0.02269729  0.28918070     0.08902070  0.02308564       -0.0493958591
0.091069756  0.26533101
volatile.acidity       -0.02269729       1.00000000 -0.14947181     0.06428606  0.07051157       -0.0970119393
0.089260504  0.02711385
citric.acid             0.28918070      -0.14947181  1.00000000     0.09421162  0.11436445        0.0940772210
0.121130798  0.14950257
residual.sugar          0.08902070       0.06428606  0.09421162     1.00000000  0.08868454        0.2990983537
0.401439311  0.83896645
chlorides               0.02308564       0.07051157  0.11436445     0.08868454  1.00000000        0.1013923521
0.198910300  0.25721132
free.sulfur.dioxide    -0.04939586      -0.09701194  0.09407722     0.29909835  0.10139235        1.0000000000
0.615500965  0.29421041
total.sulfur.dioxide    0.09106976       0.08926050  0.12113080     0.40143931  0.19891030        0.6155009650
1.000000000  0.52988132
density                 0.26533101       0.02711385  0.14950257     0.83896645  0.25721132        0.2942104109
0.529881324  1.00000000
pH                     -0.42585829      -0.03191537 -0.16374821    -0.19413345 -0.09043946       -0.0006177961
0.002320972 -0.09359149
sulphates              -0.01714299      -0.03572815  0.06233094    -0.02666437  0.01676288        0.0592172458
0.134562367  0.07449315
alcohol                -0.12088112       0.06771794 -0.07572873    -0.45063122 -0.36018871       -0.2501039415
-0.448892102 -0.78013762
                                pH   sulphates     alcohol
fixed.acidity        -0.4258582910 -0.01714299 -0.12088112
volatile.acidity     -0.0319153683 -0.03572815  0.06771794
citric.acid          -0.1637482114  0.06233094 -0.07572873
residual.sugar       -0.1941334540 -0.02666437 -0.45063122
chlorides            -0.0904394560  0.01676288 -0.36018871
free.sulfur.dioxide  -0.0006177961  0.05921725 -0.25010394
total.sulfur.dioxide  0.0023209718  0.13456237 -0.44889210
density              -0.0935914935  0.07449315 -0.78013762
pH                    1.0000000000  0.15595150  0.12143210
sulphates             0.1559514973  1.00000000 -0.01743277
alcohol               0.1214320987 -0.01743277  1.00000000
```

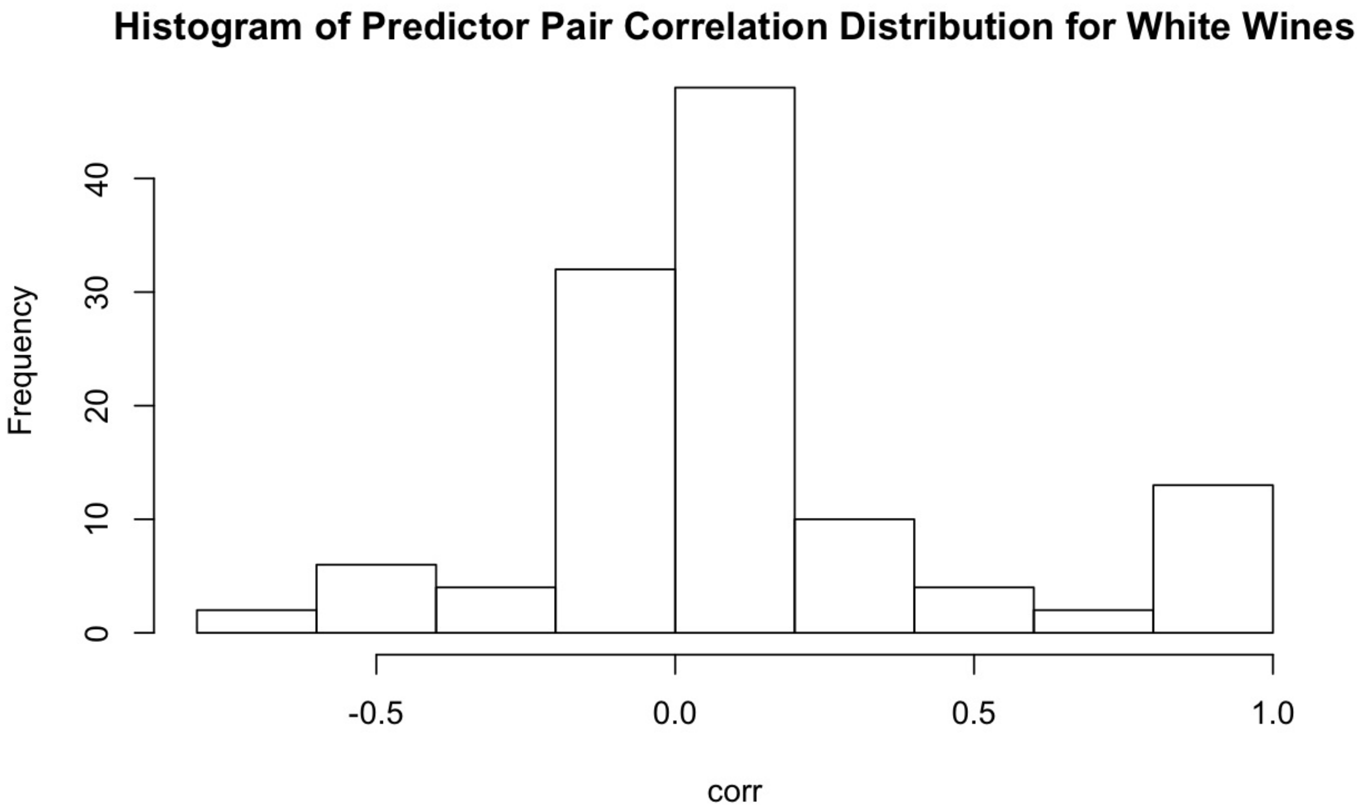
Hide

```
# Plot Heatmap
heatmap(corr, main="Correlations Between Predictors for White Wines", Colv = NA, Rowv = NA, scale="column")
```



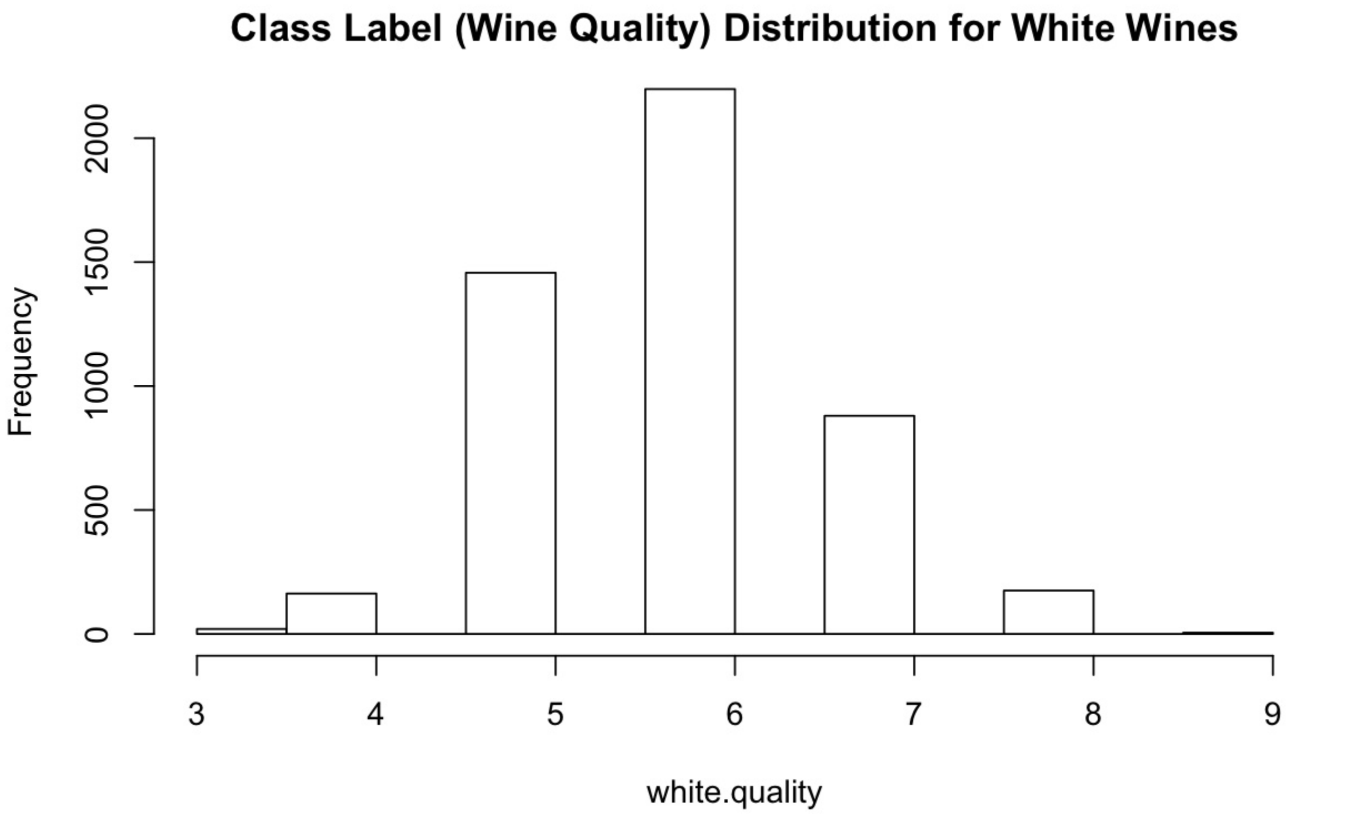
Hide

```
# Plot histogram
hist(corr, main='Histogram of Predictor Pair Correlation Distribution for White Wines')
```



Hide

```
# Plot histogram between class labels
class_dist <- hist(white.quality, main='Class Label (Wine Quality) Distribution for White Wines')
```



Hide

```
class_dist$counts
```

```
[1]   20  163    0 1457    0 2198    0  880    0  175    0    5
```

We can see that in the Wines dataset, there is very little correlation between the different predictors, so it is very difficult to find patterns among the variables, because there is not that many predictors (only 12 of them) and they are not very strong indicators. From the heat maps, we can see that most of the non-diagonal entries have a lighter color, which shows a weak relationship between the variables. Looking at the histograms, most of these predictor pairs have a correlation closer to zero, and very few of them greater than 0.3. This contributes to low accuracies accross different models, since we can't utilize any inter-predictor patterns to better learn about the data.

Moreover, we can also see that the class labels (wine qualities) are extremely unbalanced, as the bulk of the labels are 5, 6, or 7, yet there are very few examples that are 4 or lower or 8 or above. The models, in turn, will also spit out class labels that are equally unbalanced, as the predictors will also predict mostly 5s, 6s, or 7s, while very rarely predicting any other quality value.

Reference: https://ai.plainenglish.io/estimating-wine-quality-with-machine-learning-ai-72-accuracy-8a5ff0bab3b2 (https://ai.plainenglish.io/estimating-wine-quality-with-machine-learning-ai-72-accuracy-8a5ff0bab3b2)

R Notebook

This is an R Markdown Notebook. When you execute code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Cmd+Shift+Enter*.

— Random Forests —

— Data Preparation —

Let's first try random forest regression.

```
# Import random forest library
library(randomForest)

# Preprocessing Steps:

# Read red and white wine datasets
red <- read.csv('winequality-red.csv', header = TRUE, sep=";")
red <- na.omit(red)
red.quality <- red$quality
stopifnot(nrow(red) == 1599)

white <- read.csv('winequality-white.csv', header = TRUE, sep=";")
white <- na.omit(white)
white.quality <- white$quality
stopifnot(nrow(white) == 4898)

# Standardize predictors to normal distribution (mean 0 and std dev 1)
red[,-12] <- scale(red[,-12])
white[,-12] <- scale(white[,-12])

# Split into training and test sets (80-20 split)
set.seed(1)

# White dataset
train_white_idx <- sample(nrow(white) * 0.8) # 80-20 train-test split
train.white <- white[train_white_idx,]
train.white.quality <- train.white$quality

test.white <- white[-train_white_idx,]
test.white.quality <- test.white$quality
test.white <- test.white[,-13]

# Red dataset
train_red_idx <- sample(nrow(red) * 0.8) # 80-20 train-test split
train.red <- red[train_red_idx,]
train.red.quality <- train.red$quality

test.red <- red[-train_red_idx,]
test.red.quality <- test.red$quality
test.red <- test.red[,-13]
```

```
nrow(white)
```

```
[1] 4898
```

```
nrow(red)
```

```
[1] 1599
```

— Method 1: Random Forest Regressor —

White Dataset:

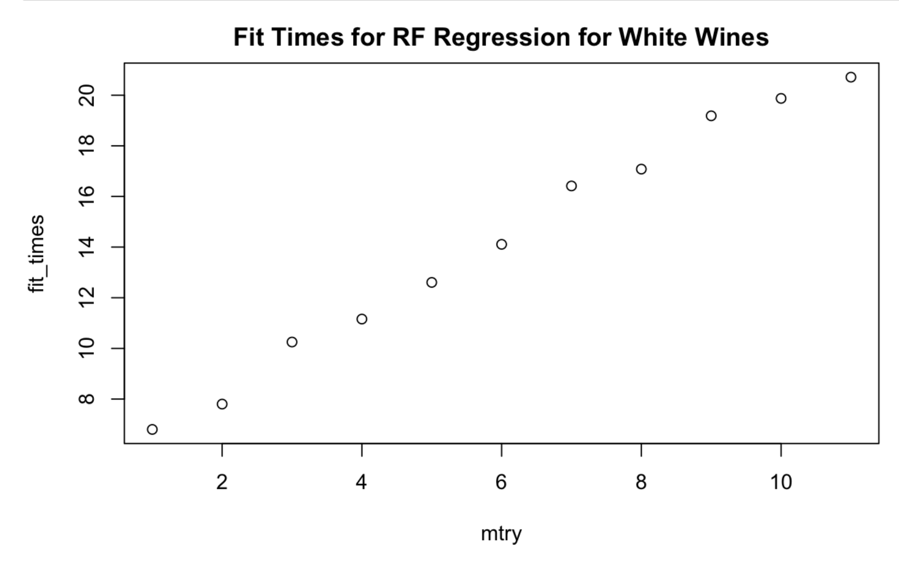
```
test_accuracies = rep(0, 11)
test_accuracies_1 = rep(0, 11)
fit_times = rep(0, 11)
ptm <- proc.time()

for(i in 1:11){
  set.seed(1)
  start =  proc.time()
  bag1 = randomForest(quality~., data=train.white, mtry=i, importance=TRUE) # default: ntree=500
  fit_times[i] <- proc.time() - start
  test_accuracies[i] = mean(abs(predict(bag1, test.white) - test.white.quality) <= 0.5)
  test_accuracies_1[i] = mean(abs(predict(bag1, test.white) - test.white.quality) <= 1)
}
```

```
number of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement length
```

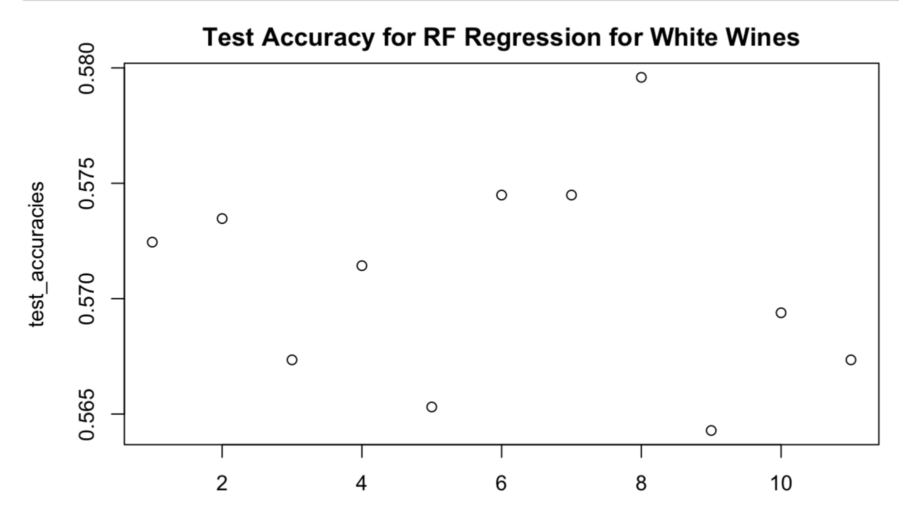
```
cv_time <- proc.time() - ptm
cv_time # record CV time
```

```
   user  system elapsed 
157.985   2.169 163.081 
```

```
plot(fit_times, xlab='mtry', main='Fit Times for RF Regression for White Wines')
```



```
plot(test_accuracies, xlab='mtry',  main='Test Accuracy for RF Regression for White Wines')
```



```
mtry = which.max(test_accuracies)
rf_acc_0.5_white <- test_accuracies[mtry]
rf_acc_1_white <- test_accuracies_1[mtry]
fit_time_white <- fit_times[mtry]

mtry
```

```
[1] 8
```

```
rf_acc_0.5_white
```

```
[1] 0.5795918
```

```
rf_acc_1_white
```

```
[1] 0.9030612
```

```
fit_time_white
```

```
[1] 17.08
```

Red Dataset:

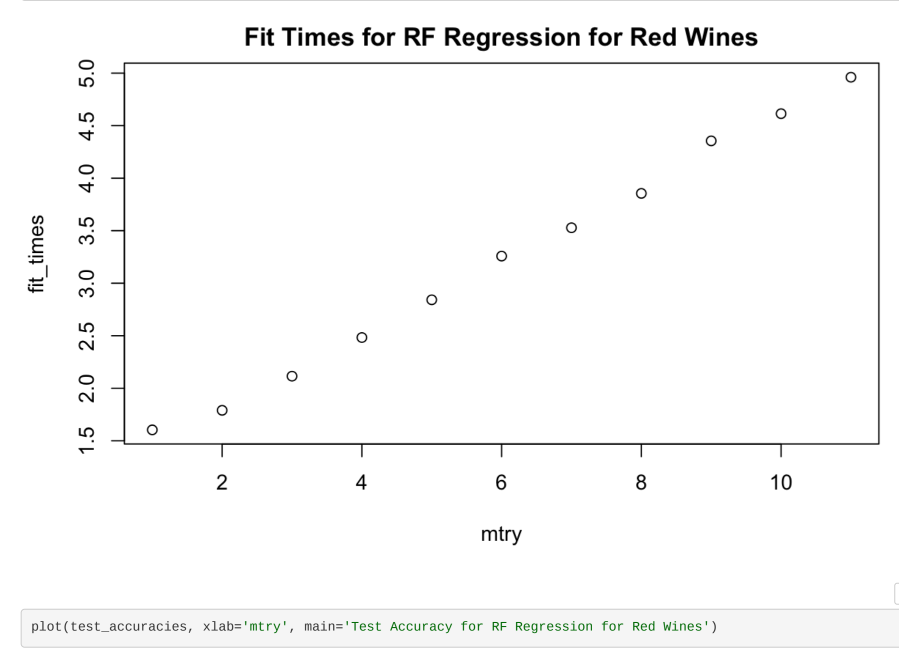
```
test_accuracies = rep(0, 11)
test_accuracies_1 = rep(0, 11)
fit_times = rep(0, 11)
ptm <- proc.time()

for(i in 1:11){
  set.seed(1)
  start =  proc.time()
  bag1 = randomForest(quality~., data=train.red, mtry=i, importance=TRUE) # default: ntree=500
  fit_times[i] <- proc.time() - start
  test_accuracies[i] = mean(abs(predict(bag1, test.red) - test.red.quality) <= 0.5)
  test_accuracies_1[i] = mean(abs(predict(bag1, test.red) - test.red.quality) <= 1)
}
```

```
number of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement length
```

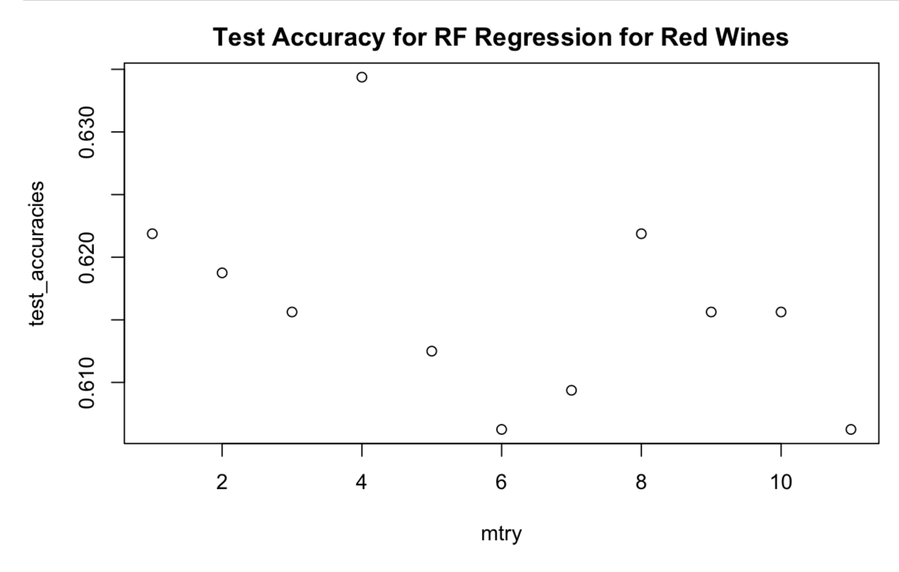
```
cv_time <- proc.time() - ptm
cv_time # record CV time
```

```
  user  system elapsed 
35.992   0.660  37.861 
```

```
plot(fit_times, xlab='mtry', main='Fit Times for RF Regression for Red Wines')
```



```
plot(test_accuracies, xlab='mtry', main='Test Accuracy for RF Regression for Red Wines')
```



```
mtry = which.max(test_accuracies)
rf_acc_0.5_red <- test_accuracies[mtry]
rf_acc_1_red <- test_accuracies_1[mtry]
fit_time_red <- fit_times[mtry]

mtry
```

```
[1] 4
```

```
rf_acc_0.5_red
```

```
[1] [illegible]
```

```
rf_acc_1_red
```

```
[1] [illegible]
```

```
fit_time_red
```

```
[1] [illegible]
```

Weighted Average Test performances for both red and white dataset:

Within absolute margin 0.5

```
(rf_acc_0.5_red * nrow(test.red) + rf_acc_0.5_white * nrow(test.white)) / (nrow(test.red) + nrow(test.white))
```

```
[1] [illegible]
```

Within absolute margin 1

```
(rf_acc_1_red * nrow(test.red) + rf_acc_1_white * nrow(test.white)) / (nrow(test.red) + nrow(test.white))
```

```
[1] [illegible]
```

— Data Preparation —

We will now try the classification task. The key difference in this code below is that the response variable quality is converted into a factor, which is expected for random forest classification.

```
[illegible]
```

— Method 2: Random Forest Classifier —

White Dataset:

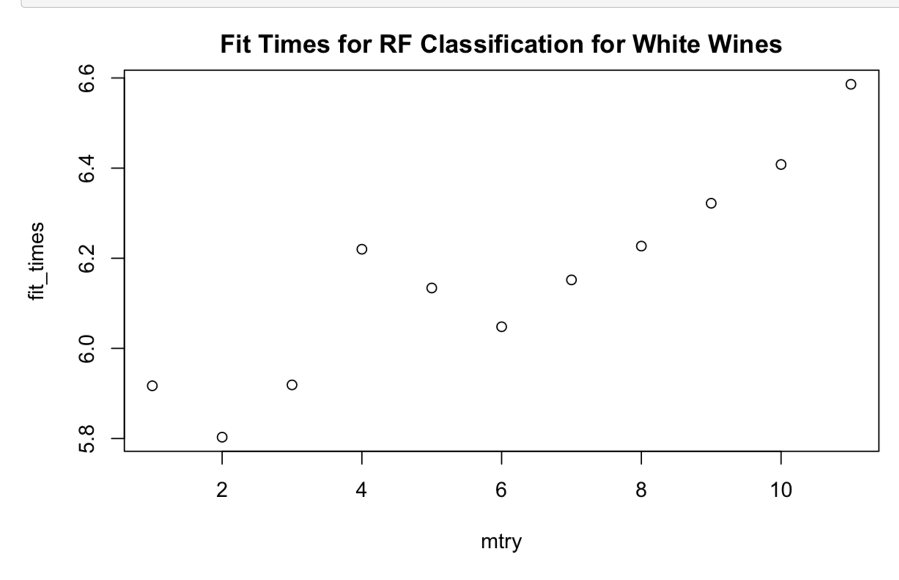
```
test_accuracies = rep(0, 11)
fit_times = rep(0, 11)
ptm <- proc.time()

for(i in 1:11){
  set.seed(1)
  start =  proc.time()
  bag1 = randomForest(quality~., data=train.white, mtry=i, importance=TRUE)
  fit_times[i] <- proc.time() - start
  test_accuracies[i] = mean(predict(bag1, test.white) == test.white.quality)
}
```

```
number of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement length
```

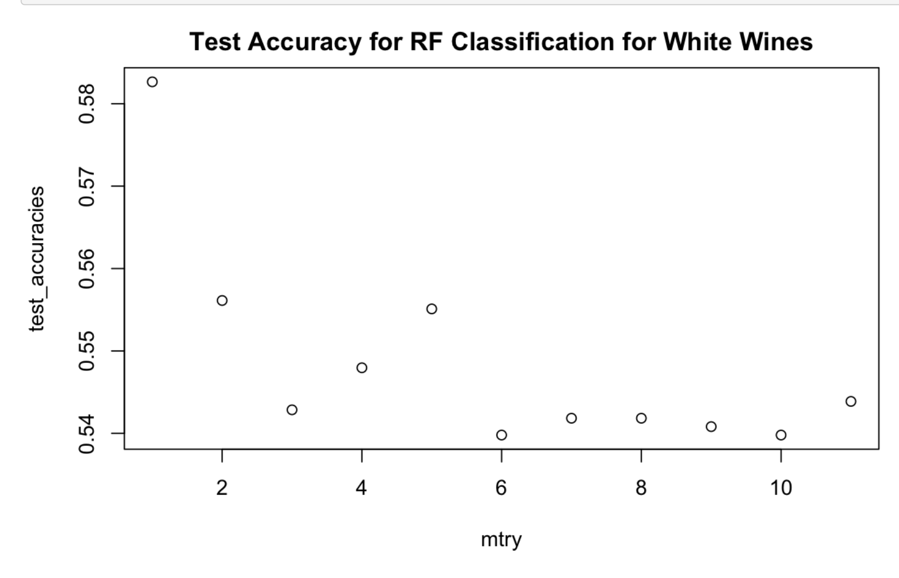
```
cv_time <- proc.time() - ptm
cv_time # record CV time
```

```
   user  system elapsed 
[illegible]
```

```
plot(fit_times, xlab='mtry', main='Fit Times for RF Classification for White Wines')
```



```
plot(test_accuracies, xlab='mtry',  main='Test Accuracy for RF Classification for White Wines')
```



```
mtry = which.max(test_accuracies)
rf_acc_white <- test_accuracies[mtry]
fit_time_white <- fit_times[mtry]

mtry
```

```
[1] [illegible]
```

```
rf_acc_white
```

```
[1] [illegible]
```

```
fit_time_white
```

```
[1] [illegible]
```

Red Dataset:

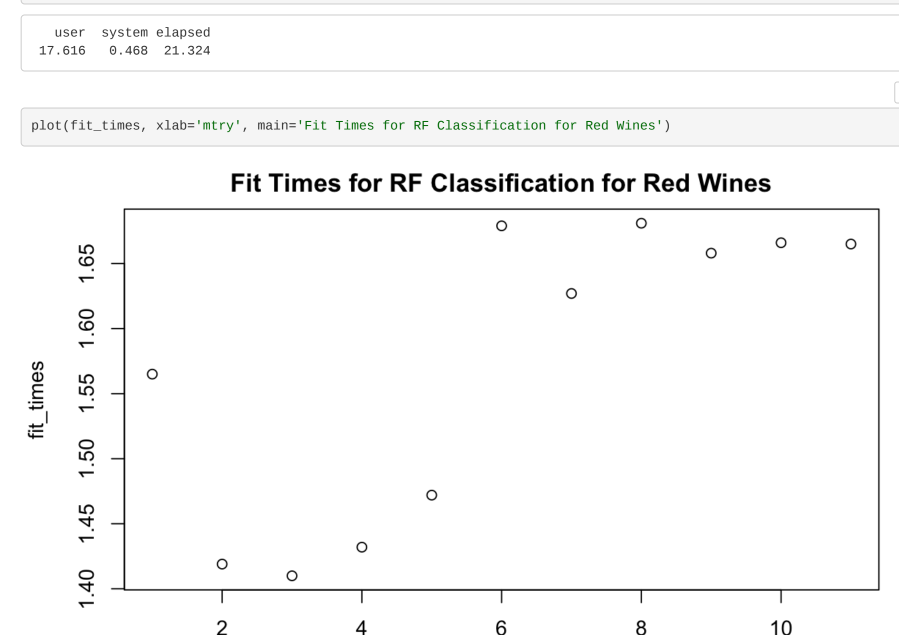
```
test_accuracies = rep(0, 11)
fit_times = rep(0, 11)
ptm <- proc.time()

for(i in 1:11){
  set.seed(1)
  start =  proc.time()
  bag1 = randomForest(quality~., data=train.red, mtry=i, importance=TRUE)
  fit_times[i] <- proc.time() - start
  test_accuracies[i] = mean(predict(bag1, test.red) == test.red.quality)
}
```

```
number of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement lengthnumber of items to replace is not a multiple of replacement length
```

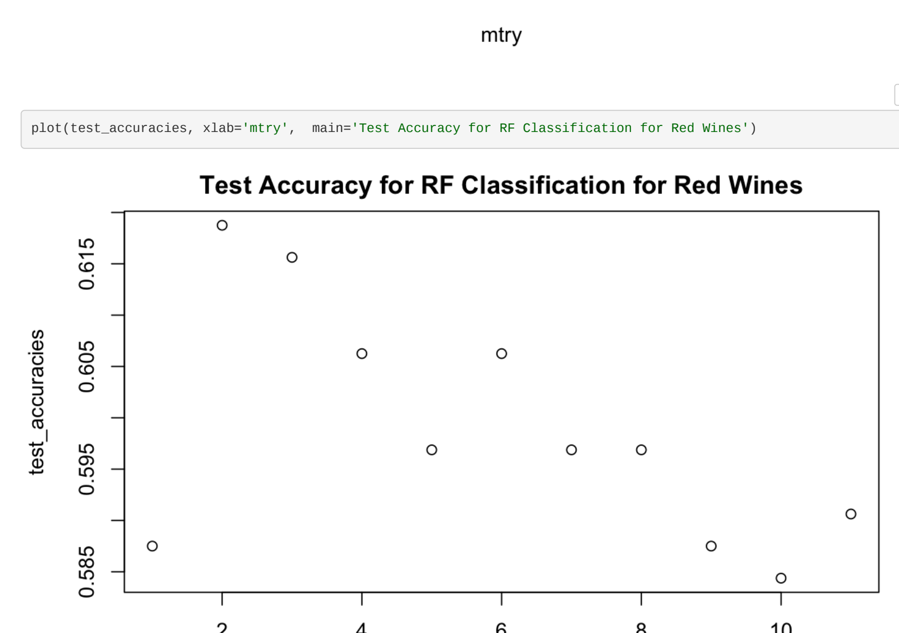
```
cv_time <- proc.time() - ptm
cv_time # record CV time
```

```
   user  system elapsed 
[illegible]
```

```
plot(fit_times, xlab='mtry', main='Fit Times for RF Classification for Red Wines')
```



```
plot(test_accuracies, xlab='mtry',  main='Test Accuracy for RF Classification for Red Wines')
```



```
mtry = which.max(test_accuracies)
rf_acc_red <- test_accuracies[mtry]
fit_time_red <- fit_times[mtry]

mtry
```

```
[1] 2
```

```
rf_acc_red
```

```
[1] 0.61875
```

```
fit_time_red
```

```
[1] 1.419
```

Weighted Average Test performances for both red and white dataset:

```
(rf_acc_red * nrow(test.red) + rf_acc_white * nrow(test.white)) / (nrow(test.red) + nrow(test.white))
```

```
[1] 0.5915385
```

— Method 1: Linear regression —

```r
# Now let's get rid of the least significant variables - citric.acid and chlorides
data.train <- data.train[, -c(4, 6)]
data.test <- data.test[, -c(4, 6)]

# Try the same method again
ptm <- proc.time()
lr <- glm(quality~., data=data.train)
print(lr.time <- proc.time() - ptm) # record training time in seconds (use the THIRD value!)
```

```
   user  system elapsed 
  0.112   0.007   0.147 
```

```r
mean(abs(predict(lr, data.test) - test.quality) <= 0.5) # Test Accuracy, within absolute class deviation 0.5
```

```
[1] 0.5676923
```

```r
mean(abs(predict(lr, data.test) - test.quality) <= 1) # Test Accuracy, within absolute class deviation 1
```

```
[1] 0.86
```

```r
summary(lr)
```

```

Call:
glm(formula = quality ~ ., data = data.train)

Deviance Residuals: 
    Min       1Q   Median       3Q      Max  
-3.8118  -0.4875  -0.0427   0.4692   3.0577  

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)    
(Intercept)           5.80940    0.01354 429.008  < 2e-16 ***
type                 -0.25048    0.03363  -7.447 1.11e-13 ***
fixed.acidity         0.11665    0.02474   4.716 2.47e-06 ***
volatile.acidity     -0.28666    0.01666 -17.202  < 2e-16 ***
residual.sugar        0.38735    0.03361  11.524  < 2e-16 ***
free.sulfur.dioxide   0.07663    0.01467   5.224 1.82e-07 ***
total.sulfur.dioxide -0.04626    0.02047  -2.260   0.0238 *  
density              -0.44886    0.05336  -8.412  < 2e-16 ***
pH                    0.12070    0.01603   7.530 5.96e-14 ***
sulphates             0.08597    0.01311   6.560 5.91e-11 ***
alcohol               0.22391    0.02714   8.251  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 0.55765)

    Null deviance: 4059.4  on 5196  degrees of freedom
Residual deviance: 2892.0  on 5186  degrees of freedom
AIC: 11726

Number of Fisher Scoring iterations: 2
```

Back to Original Dataset

— Data Preparation —

```r
# Read Dataset and into split into 80-20 train-test
data <- read.csv('wine-quality-white-and-red.csv', header = TRUE)
set.seed(1) # set seed
data$type <- as.numeric(data$type)
data[,-13] <- scale(data[,-13])

train <- sample(nrow(data) * 0.8) # 80-20 train-test split
data.train <- data[train,]
train.quality <- data.train$quality

data.test <- data[-train,]
test.quality <- data.test$quality
data.test <- data.test[,-13]
```

— Method 2: Multinomial Logistic Regression —

```r
# For multinomial logistic regression
library(nnet)
library(tidyverse)

# For unfactoring categorical integer levels
library(varhandle)

ptm <- proc.time()
model <- nnet::multinom(quality~., data = data.train) # Fit the model
```

```
# weights:  98 (78 variable)
initial  value 10112.895045 
iter  10 value 6685.233041
iter  20 value 6473.101052
iter  30 value 6132.728486
iter  40 value 5816.074738
iter  50 value 5687.466580
iter  60 value 5632.052973
iter  70 value 5604.246940
iter  80 value 5595.588812
iter  90 value 5590.638204
iter 100 value 5588.848070
final  value 5588.848070 
stopped after 100 iterations
```

```r
print(lr.time <- proc.time() - ptm) # record training time in seconds (use the THIRD value!)
```

```
   user  system elapsed 
  0.671   0.033   0.773 
```

```r
predicted.classes <- model %>% predict(data.test) # Make predictions

mean(predicted.classes == test.quality) # Test error rate
```

```
[1] 0.5653846
```

```r
summary(model) # Summarize the model
```

```
Call:
nnet::multinom(formula = quality ~ ., data = data.train)

Coefficients:
  (Intercept)      type fixed.acidity volatile.acidity citric.acid residual.sugar  chlorides free.sulfur.dioxide
total.sulfur.dioxide    density
4    4.525292 -3.714001    -1.3074928        0.3197784   0.1848244    -0.57471707 -0.2917812          -1.2093406
-0.28813123  0.4077224
5    7.250568 -5.330750    -1.4981488       -0.3154275   0.2543914     0.05254728 -0.3502184          -0.6957289
0.06929728 -0.1012274
6    7.732686 -5.584994    -1.4959268       -1.1464000   0.2102523     0.47629362 -0.3676428          -0.5076768
-0.15562209 -0.3757985
7    6.284338 -6.260060    -0.7213397       -1.4905531   0.1105291     1.67373122 -0.7148359          -0.4265093
-0.15041222 -2.2109031
8    4.276528 -6.057959    -0.7697263       -1.4058139   0.1810914     1.93927971 -0.4356059          -0.1601924
-0.27477121 -2.2381847
9   -4.716229 -5.591829     0.8193350       -1.2819008   0.2993214     0.67967440 -6.9760566          -0.1177948
-0.44645578 -0.4016252
           pH  sulphates     alcohol
4 -0.45039791 0.36626289 -0.59807128
5 -0.49445762 0.28509594 -0.85232949
6 -0.39669237 0.46489687  0.01720690
7  0.09557737 0.76482789 -0.08091591
8  0.16914775 0.64577493  0.30776579
9  1.87204493 0.09856306  1.74699190

Std. Errors:
  (Intercept)      type fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide t
otal.sulfur.dioxide   density        pH
4   0.3824428 0.4560665     0.3853931        0.2935832   0.3219232      0.5494426 0.2242741           0.2388298
0.4330206 0.6654583 0.3288060
5   0.3653902 0.4252647     0.3614121        0.2875256   0.3093925      0.5215460 0.2088354           0.1982051
0.4115882 0.6498365 0.3107740
6   0.3647652 0.4241518     0.3600657        0.2902580   0.3095108      0.5224842 0.2100650           0.1959243
0.4115096 0.6542994 0.3101582
7   0.3694562 0.4399698     0.3622288        0.2985921   0.3135338      0.5379478 0.2439895           0.1995656
0.4177159 0.6916491 0.3129395
8   0.4102287 0.5506795     0.4219741        0.3291750   0.3282108      0.6221835 0.3042027           0.2101420
0.4483918 0.8684350 0.3376430
9   2.5742973 1.8652259     0.5645353        0.9386465   0.4672072      1.6190126 2.8073098           0.7539679
1.1948898 2.2932496 0.7371580
  sulphates   alcohol
4 0.3712716 0.4081659
5 0.3587569 0.3950005
6 0.3580724 0.3965718
7 0.3598117 0.4117637
8 0.3703290 0.4789914
9 0.7959933 1.4509339

Residual Deviance: 11177.7 
AIC: 11333.7 
```

```r
# Reference:
# http://www.sthda.com/english/articles/36-classification-methods-essentials/147-multinomial-logistic-regression-essentials-in-r/
```

2. Split Dataset (Red and White Wines in separate datasets)

Let's try splitting this dataset by red and white wine, since red and white wine may have different measures in terms of quality.

— Data Preparation —

```r
# Read Dataset and into split into 80-20 train-test
data <- read.csv('wine-quality-white-and-red.csv', header = TRUE)

set.seed(1) # set seed

data.red <- data[data$type=='red',-1]
data.red[, -12] <- scale(data.red[, -12])
data.white <- data[data$type=='white',-1]
data.white[, -12] <- scale(data.white[, -12])

# White wine dataset
train.white <- sample(nrow(data.white) * 0.8) # 80-20 train-test split
data.train.white <- data.white[train.white,]
train.white.quality <- data.train.white$quality

data.test.white <- data.white[-train.white,]
test.quality.white <- data.test.white$quality
data.test.white <- data.test.white[,-13]

# Red wine dataset
train.red <- sample(nrow(data.red) * 0.8) # 80-20 train-test split
data.train.red <- data.red[train.red,]
train.red.quality <- data.train.red$quality

data.test.red <- data.red[-train.red,]
test.quality.red <- data.test.red$quality
data.test.red <- data.test.red[,-13]
```

— Linear regression —

White Dataset:

```r
# White dataset
ptm <- proc.time()
lr.white <- glm(quality~., data=data.train.white)
print(lr.time.white <- proc.time() - ptm) # record training time in seconds (use the THIRD value!)
```

```
   user  system elapsed 
  0.098   0.009   0.110 
```

```r
test_acc_0.5_white <- mean(abs(predict(lr.white, data.test.white) - test.quality.white) <= 0.5)
test_acc_1_white <- mean(abs(predict(lr.white, data.test.white) - test.quality.white) <= 1)

# Test Accuracy, within absolute class deviation 0.5
test_acc_0.5_white
```

```
[1] 0.5285714
```

```r
# Test Accuracy, within absolute class deviation 1
test_acc_1_white
```

```
[1] 0.8826531
```

```r
# Inspect variables in the linear regression
summary(lr.white)
```

```

Call:
glm(formula = quality ~ ., data = data.train.white)

Deviance Residuals: 
    Min       1Q   Median       3Q      Max  
-3.4877  -0.5083  -0.0267   0.4738   3.1563  

Coefficients:
                      Estimate Std. Error t value Pr(>|t|)    
(Intercept)           5.909111   0.012403 476.412  < 2e-16 ***
fixed.acidity         0.037385   0.019550   1.912   0.0559 .  
volatile.acidity     -0.175857   0.013087 -13.438  < 2e-16 ***
citric.acid          -0.004634   0.012942  -0.358   0.7203    
residual.sugar        0.401750   0.041591   9.660  < 2e-16 ***
chlorides            -0.001167   0.013517  -0.086   0.9312    
free.sulfur.dioxide   0.080784   0.016457   4.909 9.54e-07 ***
total.sulfur.dioxide -0.009183   0.017935  -0.512   0.6087    
density              -0.440531   0.061364  -7.179 8.37e-13 ***
pH                    0.112191   0.017706   6.336 2.62e-10 ***
sulphates             0.085696   0.013107   6.538 7.02e-11 ***
alcohol               0.276059   0.032370   8.528  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 0.5833359)

    Null deviance: 3251.6  on 3917  degrees of freedom
Residual deviance: 2278.5  on 3906  degrees of freedom
AIC: 9021

Number of Fisher Scoring iterations: 2
```

Red Dataset:

```r
# White dataset
ptm <- proc.time()
lr.red <- glm(quality~., data=data.train.red)
print(lr.time.red <- proc.time() - ptm) # record training time in seconds (use the THIRD value!)
```

```
   user  system elapsed 
  0.121   0.007   0.136 
```

```r
test_acc_0.5_red <- mean(abs(predict(lr.red, data.test.red) - test.quality.red) <= 0.5)
test_acc_1_red <- mean(abs(predict(lr.red, data.test.red) - test.quality.red) <= 1)

# Test Accuracy, within absolute class deviation 0.5
test_acc_0.5_red
```

```
[1] 0.6375
```

```r
# Test Accuracy, within absolute class deviation 1
test_acc_1_red
```

```
[1] 0.878125
```

```r
# Inspect variables in the linear regression
summary(lr.red)
```

```

Call:
glm(formula = quality ~ ., data = data.train.red)

Deviance Residuals: 
     Min        1Q    Median        3Q       Max  
-2.67559  -0.38331  -0.07094   0.45606   1.97593  

Coefficients:
                     Estimate Std. Error t value Pr(>|t|)    
(Intercept)           5.65961    0.01840 307.634  < 2e-16 ***
fixed.acidity         0.03971    0.05163   0.769 0.441931    
volatile.acidity     -0.18833    0.02437  -7.729 2.20e-14 ***
citric.acid          -0.03509    0.03230  -1.087 0.277460    
residual.sugar        0.01494    0.02508   0.596 0.551596    
chlorides            -0.08276    0.02184  -3.789 0.000158 ***
free.sulfur.dioxide   0.03473    0.02598   1.337 0.181599    
total.sulfur.dioxide -0.12127    0.02708  -4.478 8.22e-06 ***
density              -0.03665    0.04548  -0.806 0.420436    
pH                   -0.05272    0.03289  -1.603 0.109171    
sulphates             0.13683    0.02095   6.530 9.49e-11 ***
alcohol               0.30200    0.03083   9.795  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 0.419634)

    Null deviance: 843.43  on 1278  degrees of freedom
Residual deviance: 531.68  on 1267  degrees of freedom
AIC: 2532.9

Number of Fisher Scoring iterations: 2
```

Weighted Average Test performances for both red and white dataset:

```r
(test_acc_0.5_red * nrow(data.test.red) + test_acc_0.5_white * nrow(data.test.white)) / (nrow(data.test.red) + nrow(data.test.white)) # within absolute deviation 0.5
```

```
[1] 0.5553846
```

```r
(test_acc_1_red * nrow(data.test.red) + test_acc_1_white * nrow(data.test.white)) / (nrow(data.test.red) + nrow(data.test.white)) # within absolute deviation 1
```

```
[1] 0.8815385
```

R Notebook

Code

This is an R Markdown Notebook. When you execute code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Cmd+Shift+Enter*.

— Naive Bayes —

Let's use Naive Bayes. This method makes sense, becuase in our correlation matrix plots, we showed that there is a very weak relationship between (most of the) predictors, and Naive Bayes assumes that each predictor is independent from each other.

Hide

```
library(naivebayes)
```

```
naivebayes 0.9.7 loaded
```

Hide

```
red <- read.csv('winequality-red.csv', header = TRUE, sep=";")
red <- na.omit(red)
red.quality <- red$quality
red[,-12] <- scale(red[,-12])

# Red dataset
set.seed(1)
train_red_idx <- sample(nrow(red) * 0.8) # 80-20 train-test split
train.red <- red[train_red_idx,]
train.red$quality <- as.factor(train.red$quality)
train.red.quality <- train.red$quality

test.red <- red[-train_red_idx,]
test.red$quality <- as.factor(test.red$quality)
test.red.quality <- test.red$quality
test.red <- test.red[,-12]

# Fit model:
start <- proc.time()
model <- naive_bayes(quality~., data=train.red)
red_fit_time <- proc.time() - start
red_fit_time
```

```
   user  system elapsed 
  0.097   0.005   0.221 
```

Hide

```
red_acc <- mean(as.character(test.red.quality) == as.character(predict(model, test.red)))
red_acc
```

```
[1] 0.5875
```

Hide

```
white <- read.csv('winequality-white.csv', header = TRUE, sep=";")
white <- na.omit(white)
white.quality <- white$quality
white[,-12] <- scale(white[,-12])

# White dataset
set.seed(1)
train_white_idx <- sample(nrow(white) * 0.8) # 80-20 train-test split
train.white <- white[train_white_idx,]
train.white$quality <- as.factor(train.white$quality)
train.white.quality <- train.white$quality

test.white <- white[-train_white_idx,]
test.white$quality <- as.factor(test.white$quality)
test.white.quality <- test.white$quality
test.white <- test.white[,-12]

# Fit model:
start <- proc.time()
model <- naive_bayes(quality~., data=train.white)
white_fit_time <- proc.time() - start
white_fit_time
```

```
   user  system elapsed 
  0.106   0.004   0.230 
```

Hide

```
white_acc <- mean(as.character(test.white.quality) == as.character(predict(model, test.white)))
white_acc
```

```
[1] 0.3938776
```

Weighted average test accuracy:

Hide

```
# Weighted accuracies:
(red_acc * nrow(test.red) + white_acc * nrow(test.white)) / (nrow(test.red) + nrow(test.white))
```

```
[1] 0.4415385
```

Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Cmd+Option+I*.

When you save the notebook, an HTML file containing the code and output will be saved alongside it (click the *Preview* button or press *Cmd+Shift+K* to preview the HTML file).

The preview shows you a rendered HTML copy of the contents of the editor. Consequently, unlike *Knit*, *Preview* does not run any R code chunks. Instead, the output of the chunk when it was last run in the editor is displayed.