ML Block 2 - Wine Quality

Jaskirat S Marar (jasma356)

01/23/2021

Wine Quality Dataset

This data set is a collection of physicochemical variables with a sensory output as the quality of wine. The wine under consideration here is a Portuguese wine called "Vinho Verde." The data is separately available for both the white and red wine variants of this brand.(Cortez et al. 2009)

The various attributes in this dataset are as follows:

S.No.	Attribute	Details	Type
1	fixed acidity	tartaric acid g/dm^3	Input
2	volatile acidity	acetic acid g/dm^3	Input
3	citric acid	g/dm^3	Input
4	residual sugar	g/dm^3	Input
5	chlorides	Sodium Chloride g/dm^3	Input
6	free sulfur dioxide	mg/dm^3	Input
7	total sulfur dioxide	mg/dm^3	Input
8	density	g/cm^3	Input
9	pH		Input
10	sulphates	Potassium Sulphate g/dm^3	Input
11	alcohol	vol%	Input
12	Quality	Score(0-10)	Sensory Output

The dataset is available on UCI ML repository here

The data is also being attached with report as a zip file.

Research Objective

This data set gives us the opportunity to build different prediction models for determining the quality of wines using our predictor variables. So I will be attempting different techniques learned during the course to estimate/classify quality of wine using the predictor variables available to us in the data set.

Models used and experimental design

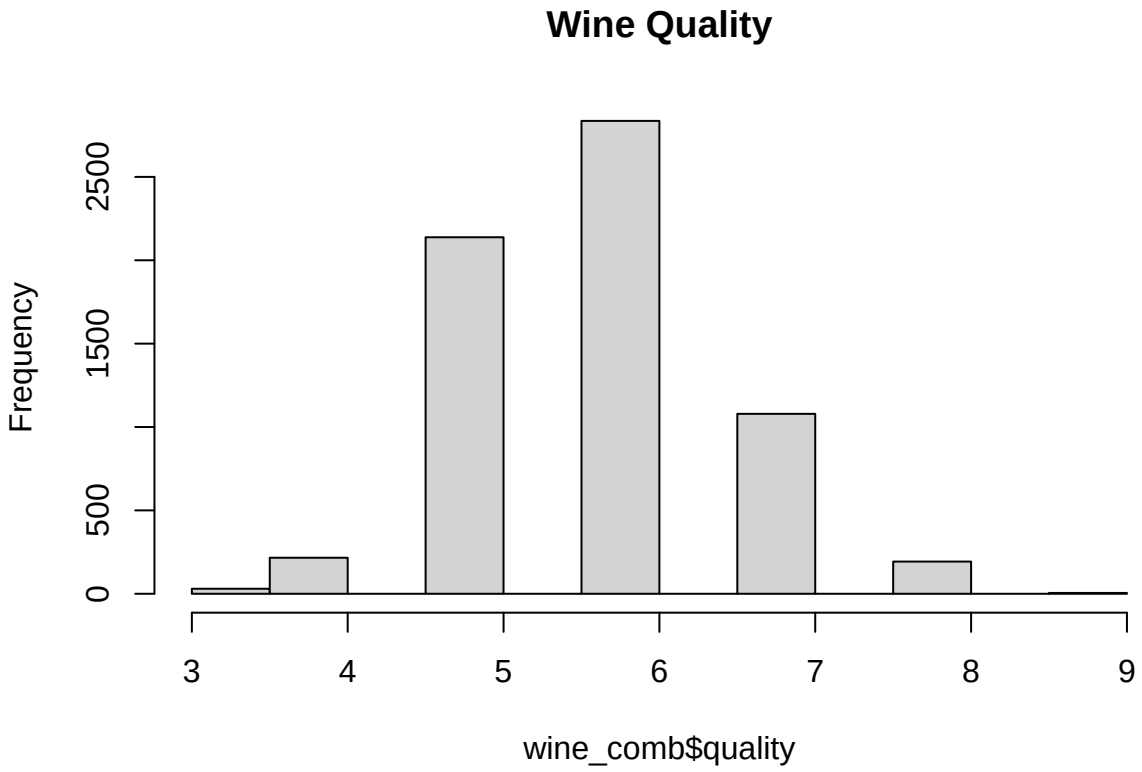
I decided to go with the following approach in this project:

1. Data Exploration to identify outliers, correlations and any other data cleanup requirement.
2. Multiclass logistic regression
3. PCA for rebasing the data and check for improved model prediction using principal components as features
4. Decision Trees for a rule based approach.
5. Random Forests for improved accuracy over Decision Trees.

Note: based on the feedback provided on the my first attempt I have decided to forego attempting this problem as a linear regression even though the class variable is numeric. Based on the feedback provided I have now completely presented this report as a solution to a classification problem.

Data Exploration

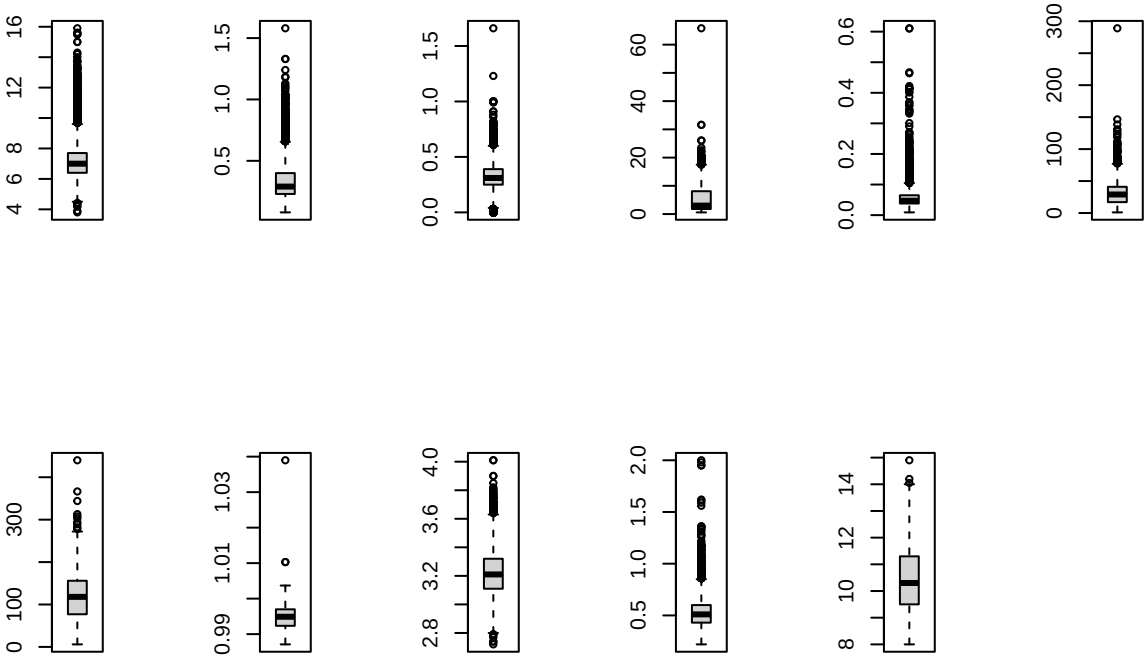
We'll start the data exploration by looking at the spread of quality of wines available in our data.



From the initial assessment of the spread in quality of different wines available in the dataset, we find that majority of the data is from wines that measure between 5-7 in terms of quality. There are very less wines of very high or very low quality in the dataset.

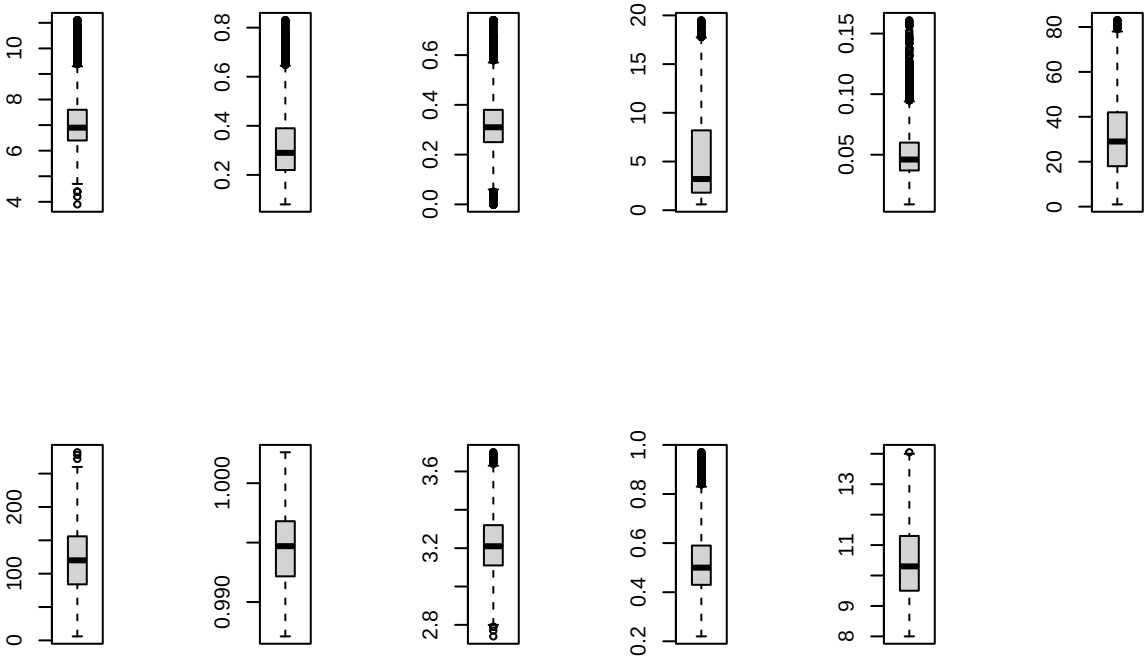
We'll now check for outliers that might impact modelling

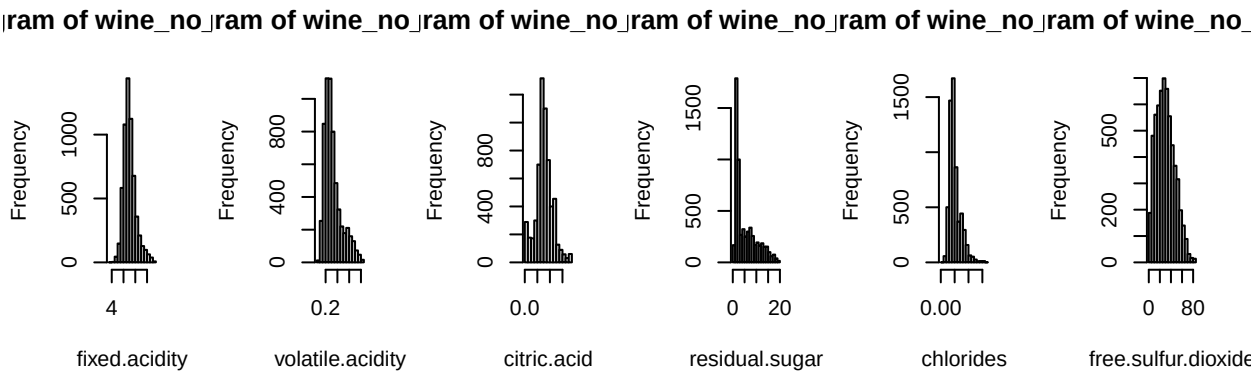
```
##
## FALSE
## 84461
```

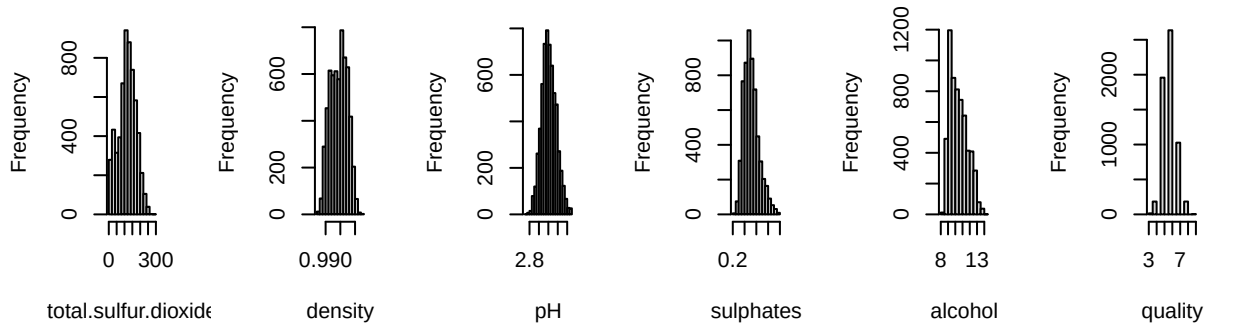


We can see that almost all data attributes have outliers associated with them. Removing outliers can be a debatable topic since, offhandedly we cant know if the outlier that we remove may or may not be responsible for explaining significant phenomenon in the data. Hence, by removing outliers we are increasing the bias in our model because we want a better fit. In this, since this data is a classification problem, we also run the risk of making the problem imbalanced if it turns out that the outliers belong more to a particular class over other classes.

But, since our dataset is relatively small I am choosing to eliminate these outliers for better a better fitted model and higher classification accuracy. I will use the Z-score method to remove outliers, where I will remove those outliers which have $|Z_{score}| > 3\sigma$ from mean.


4 6 8 10
0.2 0.4 0.6 0.8
0.0 0.2 0.4 0.6
0 5 10 15 20
0.05 0.10 0.15
0 20 40 60 80
0 100 200
0.990 1.000
2.8 3.2 3.6
0.2 0.4 0.6 0.8 1.0
8 9 11 13





As we observe, due to removal of outliers, most of the features are normally distributed, also given the fact that we are only dealing with 11 features we will choose not to scale the data unless required for modelling.

Now we do a cursory check for collinearity.

```
##                       fixed.acidity volatile.acidity citric.acid residual.sugar
## fixed.acidity                  1.00             0.20        0.24          -0.09
## volatile.acidity               0.20             1.00       -0.43          -0.20
## citric.acid                    0.24            -0.43        1.00           0.16
## residual.sugar                -0.09            -0.20        0.16           1.00
## chlorides                      0.36             0.50       -0.14          -0.14
## free.sulfur.dioxide           -0.25            -0.35        0.19           0.44
## total.sulfur.dioxide          -0.27            -0.40        0.26           0.50
## density                        0.41             0.25        0.04           0.58
## pH                            -0.22             0.26       -0.29          -0.28
## sulphates                      0.23             0.23       -0.02          -0.17
## alcohol                       -0.10            -0.04        0.00          -0.39
##                       chlorides free.sulfur.dioxide total.sulfur.dioxide density
## fixed.acidity              0.36               -0.25                -0.27    0.41
## volatile.acidity           0.50               -0.35                -0.40    0.25
## citric.acid               -0.14                0.19                 0.26    0.04
## residual.sugar            -0.14                0.44                 0.50    0.58
## chlorides                  1.00               -0.26                -0.35    0.49
## free.sulfur.dioxide       -0.26                1.00                 0.72    0.09
## total.sulfur.dioxide      -0.35                0.72                 1.00    0.10
## density                    0.49                0.09                 0.10    1.00
## pH                         0.20               -0.16                -0.26    0.05
```

```
## sulphates                0.33              -0.17              -0.26    0.24
## alcohol                 -0.32              -0.19              -0.29   -0.74
##                            pH sulphates alcohol
## fixed.acidity          -0.22      0.23   -0.10
## volatile.acidity        0.26      0.23   -0.04
## citric.acid            -0.29     -0.02    0.00
## residual.sugar         -0.28     -0.17   -0.39
## chlorides               0.20      0.33   -0.32
## free.sulfur.dioxide    -0.16     -0.17   -0.19
## total.sulfur.dioxide   -0.26     -0.26   -0.29
## density                 0.05      0.24   -0.74
## pH                      1.00      0.29    0.09
## sulphates               0.29      1.00    0.01
## alcohol                 0.09      0.01    1.00
```

There doesn't seem to be any strong correlations that we may have to address before pushing forward with our analysis. The strongest correlation that we were able to observe is between the pair of density and alcohol and also in total.SO_2 vs free.SO_2

We'll now scale the data because we will be attempting multinomial logistic regression and need the data to be scaled. We also split our data into training and test data sets in a 70-30 ratio.

Logistic Regression

As mentioned before, this data is best suited as a classification problem. So I will first start with a basic multiclass logistic regression model. The multiclass problem we have to extend the scalar valued function representing $P(y = 1|x)$ into a vector valued function (M-dimensional for M classes) with each individual class probabilities as its elements. Here a softmax function is used instead of the logistic function from the binary classification problem. The same can be written as follows:

$$\mathbf{g}(\mathbf{x}) = softmax(z) \ ; \ where \ \mathbf{z} = [\theta_1^T \ \theta_2^T \ \theta_3^T \ ...\theta_M^T]^T$$

Each individual class probability or element of the M-dimensional vector can be written as:

$$g_m(\mathbf{X}) = \frac{e^{\theta_m^T X}}{\sum_{j=1}^{M} e^{\theta_j^T X}} \ ; \ where \ m = 1, ..., M$$

I used the multinom() from the nnet package to train the multiclass logistic model for this dataset.

```
## # weights:  91 (72 variable)
## initial  value 8184.498087
## iter  10 value 5278.103792
## iter  20 value 4972.875339
## iter  30 value 4580.070485
## iter  40 value 4448.548066
## iter  50 value 4421.294612
## iter  60 value 4410.060877
## iter  70 value 4405.136136
## iter  80 value 4402.680431
## iter  90 value 4401.778767
## iter 100 value 4401.736510
## final  value 4401.736510
## stopped after 100 iterations
```

```
## Call:
## multinom(formula = fit, data = train)
##
## Coefficients:
##   (Intercept) fixed.acidity volatile.acidity citric.acid residual.sugar
## 4    2.382315    -1.3095473       0.16181259  -0.1396834      -2.677768
## 5    5.101443    -1.4574360      -0.06538864  -0.1536102      -2.813748
## 6    5.704146    -1.3881717      -0.65498117  -0.2904304      -2.439923
## 7    4.217342    -0.9061617      -0.93224842  -0.2771723      -1.641336
## 8    2.185507    -0.8121356      -0.79605680  -0.2502472      -1.011741
## 9  -13.064956     5.6840117      -2.44244987   0.1018439       3.617485
##    chlorides free.sulfur.dioxide total.sulfur.dioxide   density        pH
## 4 -0.6295217        -0.553927222           0.87724646  3.331343 -1.564350
## 5 -0.4371057         0.600615085           0.47291603  3.791911 -1.643152
## 6 -0.4281659         0.844556483           0.14753074  3.618404 -1.562457
## 7 -0.5351138         0.929447011          -0.02543718  2.612452 -1.195526
## 8 -0.3350448         1.178746500           0.01325972  1.946334 -1.024132
## 9 -4.8240652        -0.009014784           1.60136746 -4.304879  3.625551
##     sulphates   alcohol
## 4 -0.35008297 0.9267891
## 5 -0.08721462 0.8614903
## 6  0.20931822 1.7924616
## 7  0.49146179 2.1572667
## 8  0.40194697 2.1813763
## 9  0.83547394 1.2396685
##
## Std. Errors:
##   (Intercept) fixed.acidity volatile.acidity citric.acid residual.sugar
## 4   0.4715361     0.5924133        0.3824962   0.3722921      0.9076791
## 5   0.4518706     0.5678110        0.3689491   0.3587692      0.8715972
## 6   0.4509851     0.5667563        0.3688143   0.3585575      0.8700444
## 7   0.4541778     0.5713067        0.3736964   0.3621448      0.8766136
## 8   0.4781762     0.6026751        0.3975678   0.3815793      0.9198700
## 9   8.4053127     3.7025056        1.9697122   1.0070360      4.4205619
##   chlorides free.sulfur.dioxide total.sulfur.dioxide  density        pH
## 4 0.3624339           0.5462383            0.5598214 1.418429 0.4974826
## 5 0.3363634           0.5241395            0.5449174 1.364292 0.4755365
## 6 0.3360800           0.5235003            0.5442674 1.362127 0.4743924
## 7 0.3428569           0.5259372            0.5478890 1.372040 0.4774384
## 8 0.3697246           0.5393759            0.5715024 1.440027 0.4985297
## 9 3.8324006           1.5828742            1.6709485 6.914051 2.8311303
##   sulphates   alcohol
## 4 0.4275235 0.7818923
## 5 0.4083256 0.7536828
## 6 0.4074179 0.7523748
## 7 0.4090603 0.7577025
## 8 0.4193524 0.7924192
## 9 1.5214373 3.2263604
##
## Residual Deviance: 8803.473
## AIC: 8947.473
```

I will now check for the accuracy of the model by reporting the confusion matrices for both train and test data. For this I will utilize the confusionMatrix() from the caret package. This reports the confusion matrix,

accuracy and other terms related to the confusion matrix that can help us interpret the results of the model.

```
##
##  Confusion matrix for Training data:
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    3    4    5    6    7    8    9
##          3    0    0    0    0    0    0    0
##          4    0    1    4    0    0    0    0
##          5    4   80  815  402   44   12    0
##          6    6   50  556 1332  493   73    1
##          7    2    2    6  117  166   39    1
##          8    0    0    0    0    0    0    0
##          9    0    0    0    0    0    0    0
##
## Overall Statistics
##
##                Accuracy : 0.5502
##                  95% CI : (0.535, 0.5653)
##     No Information Rate : 0.4401
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.2722
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3  Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity          0.000000 0.0075188   0.5902   0.7196  0.23613  0.00000
## Specificity          1.000000 0.9990179   0.8081   0.4994  0.95233  1.00000
## Pos Pred Value            NaN 0.2000000   0.6006   0.5305  0.49850      NaN
## Neg Pred Value       0.997147 0.9685789   0.8013   0.6938  0.86135  0.97052
## Prevalence           0.002853 0.0316215   0.3283   0.4401  0.16714  0.02948
## Detection Rate       0.000000 0.0002378   0.1938   0.3167  0.03947  0.00000
## Detection Prevalence 0.000000 0.0011888   0.3226   0.5970  0.07917  0.00000
## Balanced Accuracy    0.500000 0.5032684   0.6991   0.6095  0.59423  0.50000
##                       Class: 9
## Sensitivity          0.0000000
## Specificity          1.0000000
## Pos Pred Value             NaN
## Neg Pred Value       0.9995245
## Prevalence           0.0004755
## Detection Rate       0.0000000
## Detection Prevalence 0.0000000
## Balanced Accuracy    0.5000000
```

```
##
##  Confusion matrix for Test data:
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8   9
```

```
##          3   0   0   0   0   0   0   0
##          4   0   0   1   0   0   0   0
##          5   2  28 349 170  25   4   0
##          6   1  23 220 564 231  40   0
##          7   0   0   7  51  68  16   3
##          8   0   0   0   0   0   0   0
##          9   0   0   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.5441
##                  95% CI : (0.5208, 0.5673)
##     No Information Rate : 0.4354
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.2675
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3  Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity          0.000000 0.0000000   0.6049   0.7185  0.20988  0.00000
## Specificity          1.000000 0.9994292   0.8132   0.4941  0.94794  1.00000
## Pos Pred Value            NaN 0.0000000   0.6038   0.5227  0.46897      NaN
## Neg Pred Value       0.998336 0.9716981   0.8139   0.6948  0.84560  0.96672
## Prevalence           0.001664 0.0282862   0.3200   0.4354  0.17970  0.03328
## Detection Rate       0.000000 0.0000000   0.1936   0.3128  0.03771  0.00000
## Detection Prevalence 0.000000 0.0005546   0.3206   0.5984  0.08042  0.00000
## Balanced Accuracy    0.500000 0.4997146   0.7090   0.6063  0.57891  0.50000
##                      Class: 9
## Sensitivity          0.000000
## Specificity          1.000000
## Pos Pred Value            NaN
## Neg Pred Value       0.998336
## Prevalence           0.001664
## Detection Rate       0.000000
## Detection Prevalence 0.000000
## Balanced Accuracy    0.500000
```

From the results we can see that the overall accuracy of the model is 55% for the training data and 54% for the test data. The overall accuracy only tells us that the model is predicting unseen data almost as well as the training data. If I look at the statistics by each class, we can see that the model is giving a decent true positive rate for the 2 largest classes in the data i.e. wines of quality 5 and 6, but the for all the other classes, the true positivity rate is 21% for wine quality '7' and zero for others. So the misclassification is very high for all the wines of either high or low quality. On a hunch, I want to check if my earlier decision of removing outliers might have had an impact on this model.

```
## # weights:  91 (72 variable)
## initial  value 8848.053448
## iter  10 value 5749.250013
## iter  20 value 5444.778812
## iter  30 value 5058.121461
## iter  40 value 4942.569753
## iter  50 value 4896.734161
```

```
## iter  60 value 4871.327919
## iter  70 value 4866.736747
## iter  80 value 4864.298679
## iter  90 value 4862.881473
## final  value 4862.691851
## converged
```

```
##
##  Confusion matrix for Training data:
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    3    4    5    6    7    8    9
##          3    1    0    0    0    0    0    0
##          4    0    1    2    0    0    0    0
##          5    6   95  885  459   50   13    0
##          6   10   64  587 1426  541   93    1
##          7    0    1   10  105  155   36    3
##          8    0    0    1    1    0    0    0
##          9    0    0    0    1    0    0    0
##
## Overall Statistics
##
##                Accuracy : 0.5428
##                  95% CI : (0.5282, 0.5573)
##     No Information Rate : 0.4381
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.2604
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3  Class: 4 Class: 5 Class: 6 Class: 7  Class: 8
## Sensitivity          0.0588235 0.0062112   0.5960   0.7159  0.20777 0.0000000
## Specificity          1.0000000 0.9995440   0.7965   0.4928  0.95922 0.9995460
## Pos Pred Value       1.0000000 0.3333333   0.5869   0.5239  0.50000 0.0000000
## Neg Pred Value       0.9964804 0.9647887   0.8026   0.6899  0.86051 0.9687569
## Prevalence           0.0037387 0.0354080   0.3266   0.4381  0.16406 0.0312294
## Detection Rate       0.0002199 0.0002199   0.1946   0.3136  0.03409 0.0000000
## Detection Prevalence 0.0002199 0.0006598   0.3316   0.5986  0.06818 0.0004399
## Balanced Accuracy    0.5294118 0.5028776   0.6962   0.6043  0.58350 0.4997730
##                      Class: 9
## Sensitivity          0.0000000
## Specificity          0.9997799
## Pos Pred Value       0.0000000
## Neg Pred Value       0.9991201
## Prevalence           0.0008797
## Detection Rate       0.0000000
## Detection Prevalence 0.0002199
## Balanced Accuracy    0.4998899
```

```
##
```

```
##  Confusion matrix for Test data:

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8   9
##          3   1   0   0   0   0   0   0
##          4   2   1   0   0   0   0   0
##          5   8  35 379 171  22   3   0
##          6   2  18 272 629 249  35   0
##          7   0   1   2  44  62  13   1
##          8   0   0   0   0   0   0   0
##          9   0   0   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.5497
##                  95% CI : (0.5273, 0.572)
##     No Information Rate : 0.4328
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.2686
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3  Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity          0.0769231 0.0181818   0.5804   0.7453  0.18619  0.00000
## Specificity          1.0000000 0.9989446   0.8157   0.4792  0.96228  1.00000
## Pos Pred Value       1.0000000 0.3333333   0.6133   0.5220  0.50407      NaN
## Neg Pred Value       0.9938430 0.9722650   0.7943   0.7114  0.85167  0.97385
## Prevalence           0.0066667 0.0282051   0.3349   0.4328  0.17077  0.02615
## Detection Rate       0.0005128 0.0005128   0.1944   0.3226  0.03179  0.00000
## Detection Prevalence 0.0005128 0.0015385   0.3169   0.6179  0.06308  0.00000
## Balanced Accuracy    0.5384615 0.5085632   0.6981   0.6122  0.57423  0.50000
##                      Class: 9
## Sensitivity          0.0000000
## Specificity          1.0000000
## Pos Pred Value             NaN
## Neg Pred Value       0.9994872
## Prevalence           0.0005128
## Detection Rate       0.0000000
## Detection Prevalence 0.0000000
## Balanced Accuracy    0.5000000
```

The misclassification rates for the lower and higher wine quality samples have improved slightly. Still the model is not giving a high sensitivity for these classes though.

Conclusions:

Removing outliers has improved the prediction sensitivity for the median classes i.e. 5, 6 & 7. But this came at the cost of misclassification for the higher and lower quality classes. In this problem, we are evaluating wine qualities and the quality of a wine will determine if it is a regular commodity or a luxury commodity with a very high price tag. Thus, it is important to have a model that can classify well across all classes and not just the median classes. We should not remove outliers since they are obviously essential to classification

of the higher and lower wine qualities. Especially because our current model is classifying all the higher quality wines as mid-range wines, this can lead to a potential revenue loss to the wine maker.

Now before I move onto a different classification technique altogether, I'll attempt to improve the classification model by using PCA and consequently applying the multiclass logistic regression on the Principal Components.

PCA

By running our data through this analysis, we will be learning a lower dimension representation $\mathbf{z} \in \mathbb{R}^q$ of the data $\mathbf{x} \in \mathbb{R}^{11}$ where $q < 11$ projecting the data observations onto a q-dimensional linear subspace of the original data. To do this, we have already scaled and centered our data, which is important because we will be encoding a linear combination of the original data to form our new representation of the data and if the data is not centered, then features with different variability will overwhelm the 'smoother' features in our resulting PCs. Since, we don't need to decide the lower dimension *'q'* before hand, we can obtain the solution for all values of q and later pick our components that best represent the variance of the data. Therefore, a single SVD (singular value decomposition) applied on $\mathbf{X}_0$ i.e. our centered and scaled data gives us the latent representation for all values of q as follows:

$$\mathbf{Z_0} = X_0 V$$

where, V corresponds to the eigenvectors of the new basis representation.

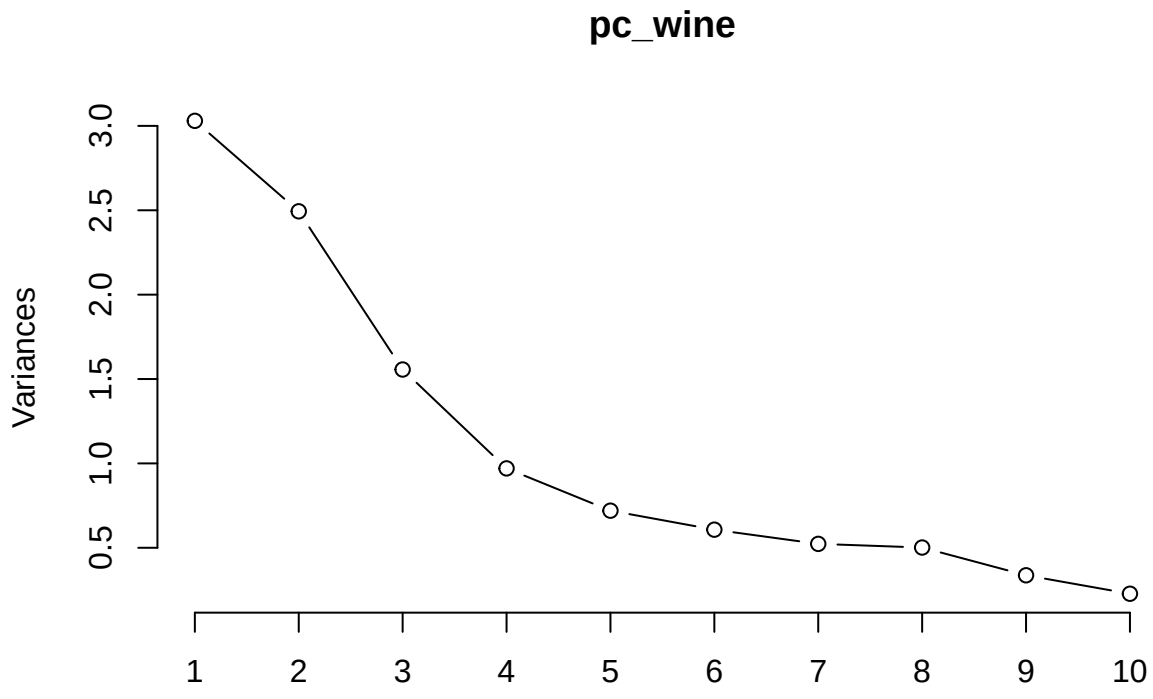
The actual implementation of the above is done using the prcomp() function as follows:

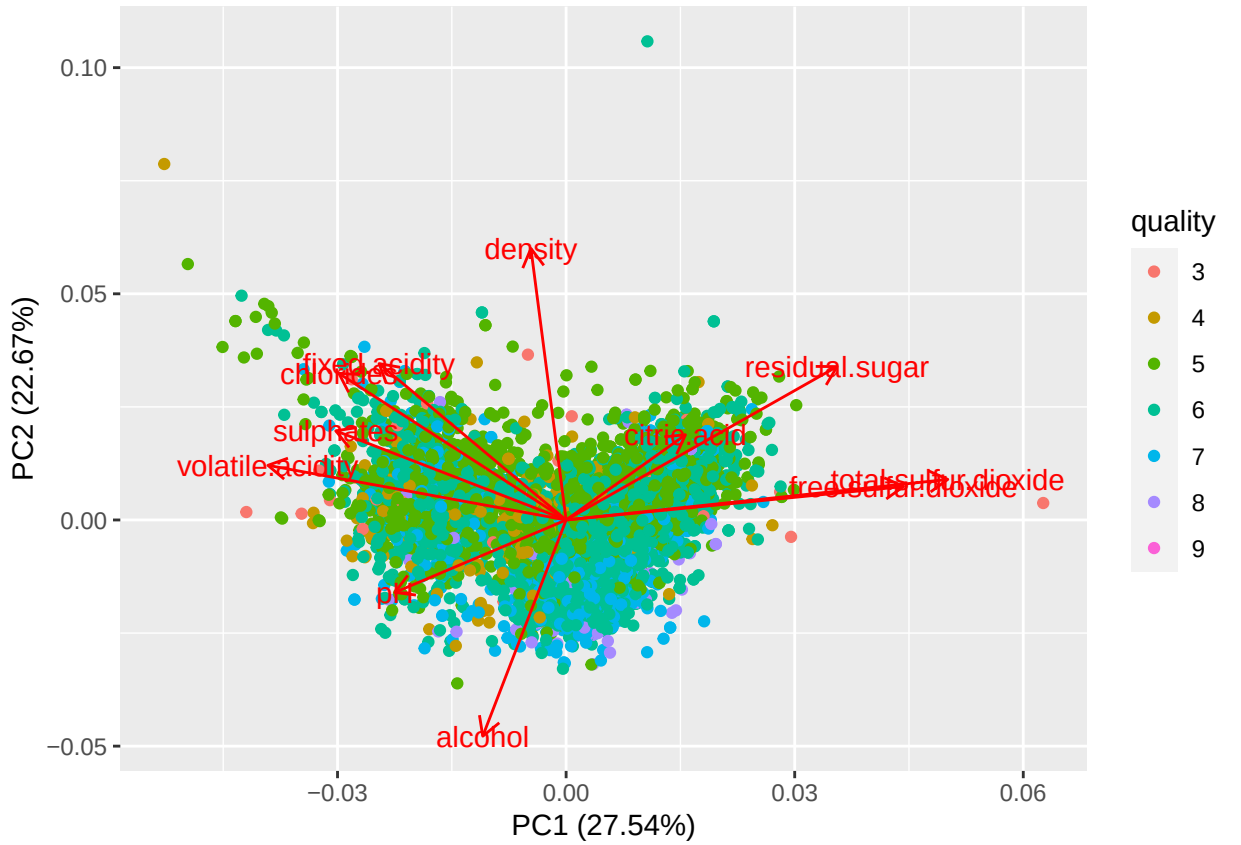
```
## Warning: In prcomp.default(scaled_wine1[, 1:11], cor = TRUE) :
##  extra argument 'cor' will be disregarded
```

```
## Standard deviations (1, .., p=11):
##  [1] 1.7406518 1.5791852 1.2475364 0.9851660 0.8484544 0.7793021 0.7232971
##  [8] 0.7081739 0.5805377 0.4771748 0.1811927
##
## Rotation (n x k) = (11 x 11):
##                              PC1         PC2         PC3         PC4        PC5
## fixed.acidity        -0.23879890  0.33635454 -0.43430130  0.16434621 -0.1474804
## volatile.acidity     -0.38075750  0.11754972  0.30725942  0.21278489  0.1514560
## citric.acid           0.15238844  0.18329940 -0.59056967 -0.26430031 -0.1553487
## residual.sugar        0.34591993  0.32991418  0.16468843  0.16744301 -0.3533619
## chlorides            -0.29011259  0.31525799  0.01667910 -0.24474386  0.6143911
## free.sulfur.dioxide   0.43091401  0.07193260  0.13422395 -0.35727894  0.2235323
## total.sulfur.dioxide  0.48741806  0.08726628  0.10746230 -0.20842014  0.1581336
## density              -0.04493664  0.58403734  0.17560555  0.07272496 -0.3065613
## pH                   -0.21868644 -0.15586900  0.45532412 -0.41455110 -0.4533764
## sulphates            -0.29413517  0.19171577 -0.07004248 -0.64053571 -0.1365769
## alcohol              -0.10643712 -0.46505769 -0.26110053 -0.10680270 -0.1888920
##                              PC6         PC7          PC8        PC9
## fixed.acidity        -0.20455371 -0.28307944  0.401235645  0.3440567
## volatile.acidity     -0.49214307 -0.38915976 -0.087435088 -0.4969327
## citric.acid           0.22763380 -0.38128504 -0.293412336 -0.4026887
## residual.sugar       -0.23347775  0.21797554 -0.524872935  0.1080032
## chlorides             0.16097639 -0.04606816 -0.471516850  0.2964437
## free.sulfur.dioxide  -0.34005140 -0.29936325  0.207807585  0.3666563
## total.sulfur.dioxide -0.15127722 -0.13891032  0.128621319 -0.3206955
## density               0.01874307 -0.04675897  0.004831136  0.1128800
## pH                    0.29657890 -0.41890702 -0.028643277  0.1278367
## sulphates            -0.29692579  0.52534311  0.165818022 -0.2077642
```

```
## alcohol              -0.51837780 -0.10410343 -0.399233887  0.2518903
##                             PC10          PC11
## fixed.acidity        -0.281267685 -0.3346792663
## volatile.acidity      0.152176731 -0.0847718098
## citric.acid           0.234463340  0.0011089514
## residual.sugar       -0.001372773 -0.4497650778
## chlorides            -0.196630217 -0.0434375867
## free.sulfur.dioxide   0.480243340  0.0002125351
## total.sulfur.dioxide -0.713663486  0.0626848131
## density              -0.003908289  0.7151620723
## pH                   -0.141310977 -0.2063605036
## sulphates             0.045959499 -0.0772024671
## alcohol              -0.205053085  0.3357018784
```

```
## Importance of components:
##                            PC1    PC2    PC3     PC4     PC5     PC6     PC7
## Standard deviation     1.7407 1.5792 1.2475 0.98517 0.84845 0.77930 0.72330
## Proportion of Variance 0.2754 0.2267 0.1415 0.08823 0.06544 0.05521 0.04756
## Cumulative Proportion  0.2754 0.5021 0.6436 0.73187 0.79732 0.85253 0.90009
##                            PC8     PC9   PC10    PC11
## Standard deviation     0.70817 0.58054 0.4772 0.18119
## Proportion of Variance 0.04559 0.03064 0.0207 0.00298
## Cumulative Proportion  0.94568 0.97632 0.9970 1.00000
```





The first 2 components together explain $\sim 50\%$ of the variance. We get 11 PCs corresponding to the 11 features and in the plot, we can see 2 groupings (which I suspect might be due to the data being a combination of red and white wines). By observing the direction of the loading vectors, we can also observe a correlation between the following features:

1. free and total sulfur dioxide
2. citric acid & residual sugar

Using these PCs, we can now create a new regression model with our PCs as the new predictor variables for predicting quality of wines.

```
## # weights:  91 (72 variable)
## initial  value 12642.578238
## iter  10 value 8656.140845
## iter  20 value 7918.394115
## iter  30 value 7495.075845
## iter  40 value 7200.450917
## iter  50 value 6981.169717
## iter  60 value 6924.798545
## iter  70 value 6918.056731
## iter  80 value 6917.505287
## iter  90 value 6917.450495
## final  value 6917.449508
## converged
```

```
## Call:
## multinom(formula = fit_pc, data = scaled_wine1)
##
```

```
## Coefficients:
##   (Intercept) pc_wine$x[, 1] pc_wine$x[, 2] pc_wine$x[, 3] pc_wine$x[, 4]
## 4    2.280696    -0.20748360     -0.2099349    -0.12034162     0.31250046
## 5    4.815474     0.01662872      0.1356827    -0.02604461    -0.08390934
## 6    5.388416     0.08313516     -0.3055935    -0.39206315    -0.42573532
## 7    3.910994     0.15381856     -0.6943890    -0.72975977    -0.68202552
## 8    1.839149     0.35166819     -0.8199411    -0.72417275    -0.73438638
## 9   -7.714732     1.71694381     -2.9417144    -1.06042177     0.67846160
##   pc_wine$x[, 5] pc_wine$x[, 6] pc_wine$x[, 7] pc_wine$x[, 8] pc_wine$x[, 9]
## 4    -0.19625698      0.6293745      1.1153665    -0.06597173   -1.350479056
## 5    -0.09537669      0.5955233      1.0962726     0.13155829   -0.773450893
## 6    -0.52759292      0.2435836      1.4198461    -0.26337091    0.003963637
## 7    -1.04207667     -0.1223549      1.5844999    -0.39521564    0.483931190
## 8    -1.04016223     -0.3797039      1.4602925    -0.73475621    0.588081551
## 9    -6.68373679     -1.5195176      0.1316126     2.96506815   -0.182174414
##   pc_wine$x[, 10] pc_wine$x[, 11]
## 4      -0.2537257        1.464674
## 5       0.3716649        2.593090
## 6       0.4299891        2.660998
## 7       0.3407620        1.632342
## 8       0.4424733        1.169708
## 9       0.9689755        1.696557
##
## Std. Errors:
##   (Intercept) pc_wine$x[, 1] pc_wine$x[, 2] pc_wine$x[, 3] pc_wine$x[, 4]
## 4   0.3026571     0.10781863      0.1395727      0.1475602      0.2160285
## 5   0.2877106     0.09764343      0.1305389      0.1378527      0.2014519
## 6   0.2869363     0.09780559      0.1302584      0.1380025      0.2013599
## 7   0.2901358     0.10072684      0.1323133      0.1408482      0.2043010
## 8   0.3154639     0.11913195      0.1432167      0.1565497      0.2177735
## 9   2.7001502     0.86363346      0.8450721      0.4715923      0.7812806
##   pc_wine$x[, 5] pc_wine$x[, 6] pc_wine$x[, 7] pc_wine$x[, 8] pc_wine$x[, 9]
## 4      0.2082605      0.2560043      0.2817811      0.2523701      0.2942418
## 5      0.1873580      0.2400598      0.2650785      0.2302368      0.2721469
## 6      0.1878729      0.2395578      0.2656928      0.2300500      0.2729811
## 7      0.1981869      0.2425129      0.2693134      0.2367413      0.2801304
## 8      0.2428854      0.2564647      0.2854363      0.2672408      0.3092425
## 9      2.1167369      0.8887369      0.8725231      1.5750128      1.2295977
##   pc_wine$x[, 10] pc_wine$x[, 11]
## 4       0.3959154       0.9382815
## 5       0.3689853       0.8573052
## 6       0.3690956       0.8569144
## 7       0.3749945       0.8689621
## 8       0.4049930       0.9824846
## 9       1.1141940       2.5906858
##
## Residual Deviance: 13834.9
## AIC: 13978.9

##
##  Confusion matrix using PCA:

## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction    3    4    5    6    7    8    9
##          3    3    0    0    0    0    0    0
##          4    0    1    2    0    0    0    0
##          5   15  133 1281  651   75   16    0
##          6   12   80  842 2013  754  127    1
##          7    0    2   12  171  250   50    4
##          8    0    0    1    0    0    0    0
##          9    0    0    0    1    0    0    0
##
## Overall Statistics
##
##                Accuracy : 0.5461
##                  95% CI : (0.5339, 0.5583)
##     No Information Rate : 0.4365
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.2682
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                       Class: 3  Class: 4 Class: 5 Class: 6 Class: 7  Class: 8
## Sensitivity          0.1000000 0.0046296   0.5992   0.7098  0.23170 0.0000000
## Specificity          1.0000000 0.9996816   0.7958   0.5040  0.95589 0.9998414
## Pos Pred Value       1.0000000 0.3333333   0.5901   0.5257  0.51125 0.0000000
## Neg Pred Value       0.9958423 0.9668925   0.8019   0.6915  0.86202 0.9702894
## Prevalence           0.0046175 0.0332461   0.3291   0.4365  0.16608 0.0297060
## Detection Rate       0.0004618 0.0001539   0.1972   0.3098  0.03848 0.0000000
## Detection Prevalence 0.0004618 0.0004618   0.3342   0.5893  0.07527 0.0001539
## Balanced Accuracy    0.5500000 0.5021556   0.6975   0.6069  0.59379 0.4999207
##                       Class: 9
## Sensitivity          0.0000000
## Specificity          0.9998460
## Pos Pred Value       0.0000000
## Neg Pred Value       0.9992303
## Prevalence           0.0007696
## Detection Rate       0.0000000
## Detection Prevalence 0.0001539
## Balanced Accuracy    0.4999230
```

I wasn't able to improve the accuracy of the model much by using Principal components as features. Only sensitivity for class '3' & '7' showed any improvement. Misclassification for the higher classes i.e. 8 & 9 still remains very high. The overall accuracy of the model remained ~ 55%

I'll try running employing the random forest approach to see if we are able to achieve better results.

Decision Trees

To continue our deep dive into the wine quality data we will employ Decision Trees to classify our predictions and compare them to the results we have achieved so far. I use the rpart library to create the decision tree and use the classification method to get results.

For learning a classification tree we compute the prediciton associated with each region by taking a majority vote and the split at the internal node is calculated by solving the following optimization problem:

$$min_{j,s}\ n_1Q_1\ +\ n_2Q_2$$

where n1 & n2 are the number of training points on both nodes being considered. Q1 & Q2 are the costs or the prediction errors with these nodes. The splitting criteria used here will be the Gini index, defined as :

$$Q_l = \sum_{m=1}^{M} \hat{\pi}_{lm}(1 - \hat{\pi}_{lm})$$

where $\hat{\pi}_{lm}$ is the proportion of the training data points in the l^{th} region belonging to class *'m'*

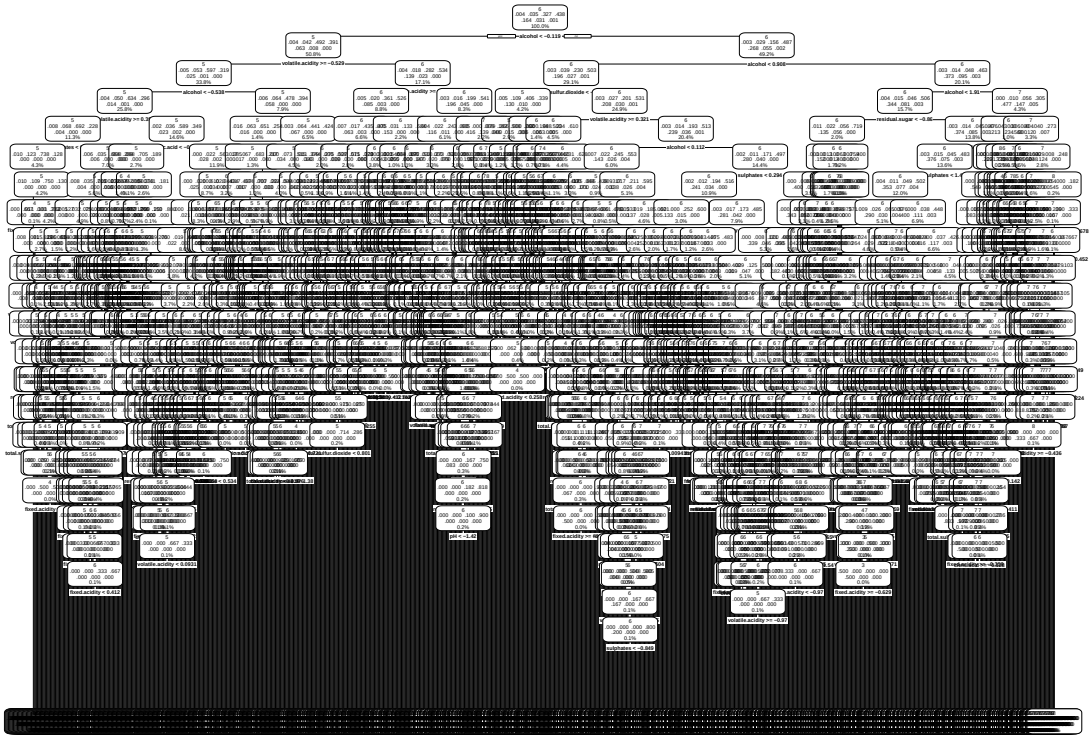
When I first learned the model I found that only classes 5 & 6 got learned in the model. The reason for this lies in our initial data exploration. We know that most of the wines data is for wines of either quality 5,6 or 7. Because of this massive data imbalance, our Decision Tree is ignoring those smaller data points and only learning these 2 classifications. I figured out that I can change the controls for rpart() i.e.

1. minsplit i.e. smallest number of observations allowed in a parent node which can be split further. The default value is 20 hence all parent nodes with less than 20 observations get labelled as terminal nodes,
2. minbucket i.e. minimum number of observations allowed in a terminal node. A split decision breaking up the data into a node with less than this value will result in the node getting rejected, and
3. cp or complexity parameter: which is the minimum improvement that the model needs at each node. It functions as a stopping parameter and has a default value of 0.01 to avoid formation of really deep trees.

Unfortunately, this problem requires a deeper tree for the lower and higher wine qualities to be included in the Decision tree.

```
## Warning: All boxes will be white (the box.palette argument will be ignored) because
## the number of classes in the response 7 is greater than length(box.palette) 6.
## To silence this warning use box.palette=0 or trace=-1.
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



```
##   3   4   5   6   7   8   9
##   3  53 670 839 304  76   5
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8   9
##          3   0   0   1   1   1   0   0
##          4   1  14  22  13   2   1   0
##          5   7  26 434 163  36   4   0
##          6   4  11 162 551 101  10   0
##          7   1   3  23  98 166  12   1
##          8   0   1  10  18  23  24   0
##          9   0   0   1   0   4   0   0
##
## Overall Statistics
##
##                Accuracy : 0.6097
##                  95% CI : (0.5877, 0.6315)
##     No Information Rate : 0.4328
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4178
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
```

```
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity          0.000000 0.254545   0.6646   0.6528  0.49850  0.47059
## Specificity          0.998451 0.979420   0.8180   0.7396  0.91466  0.97262
## Pos Pred Value       0.000000 0.264151   0.6478   0.6567  0.54605  0.31579
## Neg Pred Value       0.993323 0.978387   0.8289   0.7363  0.89854  0.98559
## Prevalence           0.006667 0.028205   0.3349   0.4328  0.17077  0.02615
## Detection Rate       0.000000 0.007179   0.2226   0.2826  0.08513  0.01231
## Detection Prevalence 0.001538 0.027179   0.3436   0.4303  0.15590  0.03897
## Balanced Accuracy    0.499226 0.616982   0.7413   0.6962  0.70658  0.72160
##                       Class: 9
## Sensitivity          0.0000000
## Specificity          0.9974346
## Pos Pred Value       0.0000000
## Neg Pred Value       0.9994859
## Prevalence           0.0005128
## Detection Rate       0.0000000
## Detection Prevalence 0.0025641
## Balanced Accuracy    0.4987173
```

```
## [1] "Misclassification error for test :  0.39"
```

So by building a very deep Decision Tree we are able to improve the classification sensitivity for more classes. the overall accuracy of the decision tree has to ~61% and now we have decent true positive rates for classes '4' through '8.'

I'll make one final attempt at improving the sensitivity for our lowest and highest quality of wines i.e. class '3' and class '9' by building on the improvement from decision trees and attempting to grow a random forest.

Random Forest

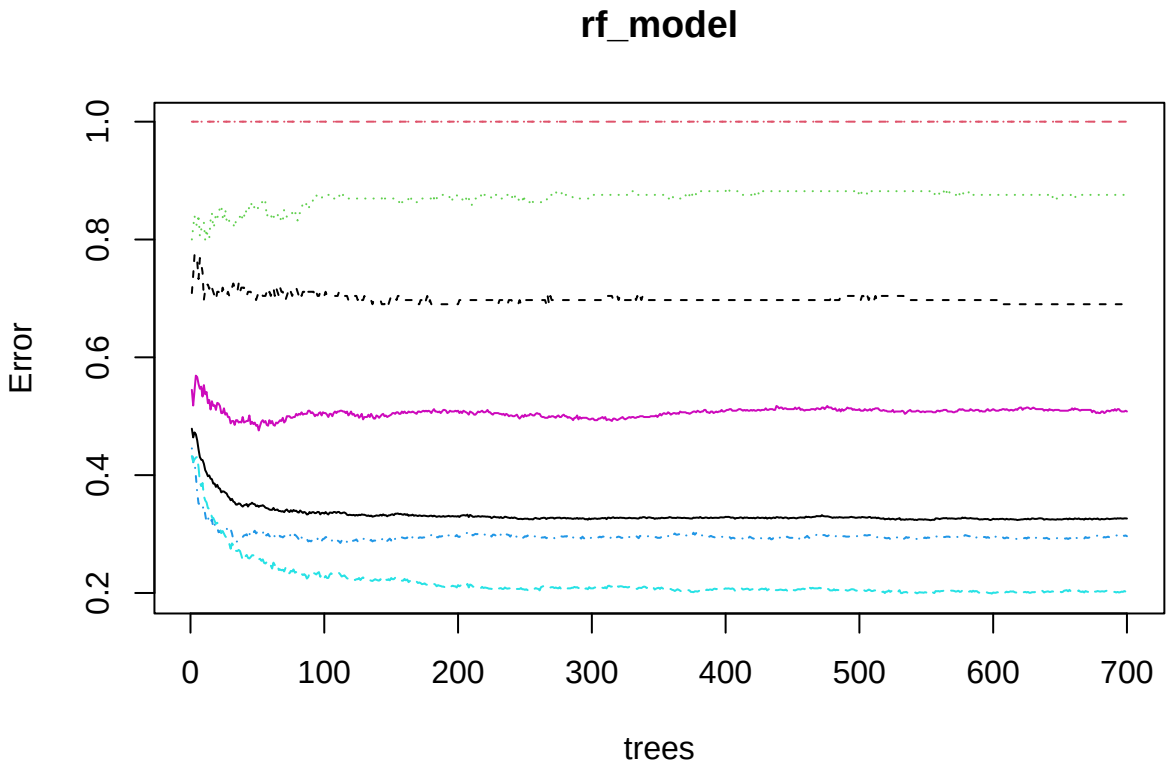
Random forests are a bagging technique in which we sample with replacement from the training data because the training data is assumed to be a good representation of the population. By utilizing a bagging technique I am attempting to reduce the variance by averaging the class probabilities of each ensemble model. This should be particularly useful to solve our recurring problem of lower and higher wine quality classes getting misclassified in our models thus far due to a lack of training observations. By bootstrapping the training data set we might be able to coerce a model that is able to improve the sensitivity for the lower and higher quality classes.

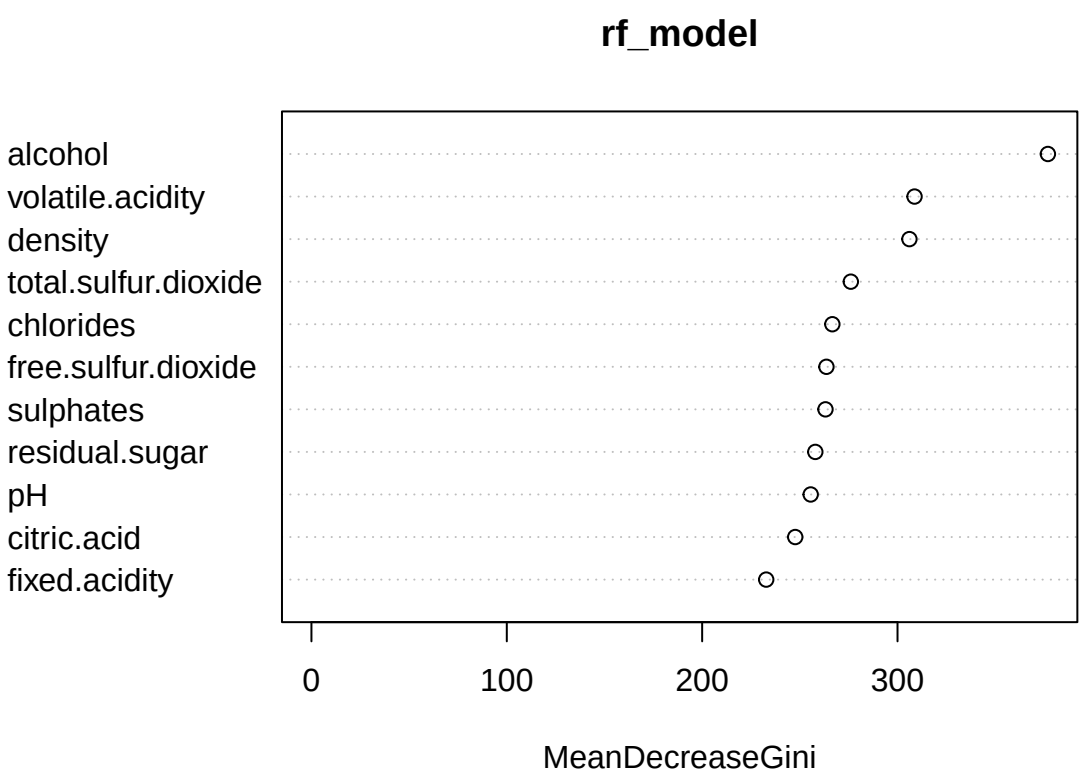
THe principle of random forest is that at each node split, we dont consider all input variables, instead a random subset is picked with lesser number of variables being considered for splitting. This increases the chances of the lower and higher quality classes being considered as a different training subset is used for each ensemble decision tree. So, while the individual variance of each tree might increase but overall prediction variance of the model is expected to decrease.

We implement the random forest using the randomForest library.

```
##
## Call:
##  randomForest(formula = fit, data = train1, ntree = 700)
##                Type of random forest: classification
##                      Number of trees: 700
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 32.64%
## Confusion matrix:
```

```
##   3  4    5    6   7  8 9 class.error
## 3 0  1    8    8   0  0 0   1.0000000
## 4 1 20   86   53   1  0 0   0.8757764
## 5 0  5 1045  429   6  0 0   0.2962963
## 6 0  1  297 1588 104  2 0   0.2028112
## 7 0  0   18  357 367  4 0   0.5080429
## 8 0  0    0   60  39 43 0   0.6971831
## 9 0  0    0    2   2  0 0   1.0000000
```





So the output of the random forest shows us that we have grown a forest with 1000 trees and 3 variables tried for each tree. We can already see improved results as we are now getting predictions for the higher and lower classes now. From the plot we can see that after around 400 trees the error has stabilized. From the variable importance plot we can see that the variables that contribute most to the model are alcohol, volatile acidity and density. This is inline with what we saw when we plotted the principal component loadings earlier. There we saw that these 3 features were standing out from the rest.

Lets see the results for the test data

```
##    3    4    5    6    7    8    9
##    0    9  635 1059  218   29    0
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8   9
##          3   0   0   0   0   0   0   0
##          4   0   6   2   1   0   0   0
##          5   8  31 461 125  10   0   0
##          6   5  18 184 685 150  17   0
##          7   0   0   5  33 168  11   1
##          8   0   0   1   0   5  23   0
##          9   0   0   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.6887
##                  95% CI : (0.6676, 0.7092)
```

```
##     No Information Rate : 0.4328
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5108
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity          0.000000 0.109091   0.7060   0.8116  0.50450  0.45098
## Specificity          1.000000 0.998417   0.8658   0.6618  0.96908  0.99684
## Pos Pred Value            NaN 0.666667   0.7260   0.6468  0.77064  0.79310
## Neg Pred Value       0.993333 0.974755   0.8540   0.8215  0.90473  0.98542
## Prevalence           0.006667 0.028205   0.3349   0.4328  0.17077  0.02615
## Detection Rate       0.000000 0.003077   0.2364   0.3513  0.08615  0.01179
## Detection Prevalence 0.000000 0.004615   0.3256   0.5431  0.11179  0.01487
## Balanced Accuracy    0.500000 0.553754   0.7859   0.7367  0.73679  0.72391
##                       Class: 9
## Sensitivity          0.0000000
## Specificity          1.0000000
## Pos Pred Value             NaN
## Neg Pred Value       0.9994872
## Prevalence           0.0005128
## Detection Rate       0.0000000
## Detection Prevalence 0.0000000
## Balanced Accuracy    0.5000000
```

So, we've been able to further improve our classification accuracy to ~69%. But most of this improvement is coming from a higher sensitivity in our median classes i.e. 5,6 & 7. Infact the sensitivity for class 4 has reduced while out extreme end classes 3 & 9 remain plagued with high misclassification errors.

Discussion

The wines dataset is not particularly complicated to understand and the dataset is more or less very clean. it required minimum cleaning and even removal of outliers didnt result in a huge of loss of data. But as I was able to later conclude, that removing outliers was doing more harm than good in this case. I was able to do a lot with this data and ended up employing multiple models to get a thorough understanding of this data. While I have detailed my findings in the construction of this report, I would summarize some of my key takeaways as follows:

1. The data is very skewed in terms of the quality of wines captured. The biggest drawback of this was the inability to improve classification for the hgihest and lowest quality wines. This can be problematic in the real world because high quality wines can become luxury goods.
2. While I was able to gradually improve the accuracy of the model, I feel that data skewness really prevents better classification results. Off the top of my head, from the techniques I did not explore, building a neural network classification model might help in improving the prediction accuracy here. The reason I would expect this is because the classification neural network builds on the softmax parametrization that I used in the beginning and the multiclass logistic regression is a simple case of neural network with a single hidden layer. That said, the wine dataset does not really have a lot features which would potentially benefit for a deep learning network and necessarily provide a better output. Sometimes for a simpler dataset a simpler model is the most efficient one!

APPENDIX - CODE

```
knitr::opts_chunk$set(echo = TRUE)
library(ggplot2)
library(dplyr)
library(glmnet)
library(plotmo)
library(glmnet)
library(ggfortify)
library(rpart)
library(rpart.plot)
library(caret)
library(nnet)
library(randomForest)
#combine red & white wine data

redW <- read.csv("winequality/winequality-red.csv", sep = ";")
whiteW <- read.csv("winequality/winequality-white.csv", sep = ";")

redW$type <- "red"
whiteW$type <- "white"

wine_comb <- rbind(redW, whiteW)

#check for split in quality of wines in dataset
hist(wine_comb$quality, main = "Wine Quality")
# checking for na values in the data
table(is.na(wine_comb))

# checking for outliers in the data
par(mfrow = c(2,6))
for (i in 1:11) {
  boxplot(wine_comb[i])
}
#removing outliers
z_scores <- as.data.frame(sapply(wine_comb[1:11],
                                 function(df) (abs(df-mean(df))/sd(df))))

z_scores <- cbind.data.frame(z_scores, wine_comb[12:13])

wine_no_outlier <- wine_comb[!rowSums(z_scores[,1:11]>3), ]

#recheck boxplots
#checking for outliers in the data
par(mfrow = c(2,6))
for (i in 1:11) {
  boxplot(wine_no_outlier[i])
}

par(mfrow = c(2,6))
for (i in 1:12) {
  hist(wine_no_outlier[[i]], xlab = names(wine_no_outlier)[i])
}
```

```
rm(i, z_scores)

#collinearity
wine_corr <- cor(wine_no_outlier[,1:11])
wine_corr <- round(wine_corr,2)
wine_corr
cat('\n')
rm(wine_corr)

scaled_wine <- wine_no_outlier
scaled_wine[,1:11] <- scale(scaled_wine[, 1:11])
scaled_wine[,12] <- as.factor(scaled_wine[,12])

#split into train and test 70-30 and scaling

n <- dim(scaled_wine)[1]
set.seed(12345)
id <- sample(1:n, floor(n*0.7))
train <- as.data.frame(scaled_wine[id,])
test <- as.data.frame(scaled_wine[-id,])

rm(id, n)


#Logistic regression

fit <- quality ~ fixed.acidity+
                 volatile.acidity+
                 citric.acid+
                 residual.sugar+
                 chlorides+
                 free.sulfur.dioxide+
                 total.sulfur.dioxide+
                 density+
                 pH+
                 sulphates+
                 alcohol

logis_model <- multinom(fit, train)

summary(logis_model)

cat("\n Confusion matrix for Training data: \n")
#confusion matrix for train data
confusionMatrix(predict(logis_model),train[,12])

cat("\n Confusion matrix for Test data: \n")
#confusion matrix for train data
confusionMatrix(predict(logis_model, newdata = test), test[,12])


scaled_wine1 <- wine_comb
scaled_wine1[,1:11] <- scale(scaled_wine1[, 1:11])
```

```
scaled_wine1[,12] <- as.factor(scaled_wine1[,12])

#split into train and test 70-30 and scaling

n <- dim(scaled_wine1)[1]
set.seed(12345)
id <- sample(1:n, floor(n*0.7))
train1 <- as.data.frame(scaled_wine1[id,])
test1 <- as.data.frame(scaled_wine1[-id,])

rm(id, n)

logis_model1 <- multinom(fit, train1)

cat("\n Confusion matrix for Training data: \n")
#confusion matrix for train data
confusionMatrix(predict(logis_model1),train1[,12])

cat("\n Confusion matrix for Test data: \n")
#confusion matrix for train data
confusionMatrix(predict(logis_model1, newdata = test1), test1[,12])



rm(scaled_wine, test, train, wine_no_outlier)

pc_wine <- prcomp(scaled_wine1[,1:11], cor = TRUE)

pc_wine

summary(pc_wine)

screeplot(pc_wine, type = 'l')

autoplot(pc_wine,
         data = scaled_wine1,
         colour = 'quality',
         loadings = TRUE,
         loadings.label = TRUE)

fit_pc <- scaled_wine1$quality~pc_wine$x[,1]+pc_wine$x[,2]+pc_wine$x[,3]+
                               pc_wine$x[,4]+pc_wine$x[,5]+pc_wine$x[,6]+
                               pc_wine$x[,7]+pc_wine$x[,8]+pc_wine$x[,9]+
                               pc_wine$x[,10]+pc_wine$x[,11]

logis_model_pc <- multinom(fit_pc, scaled_wine1)

summary(logis_model_pc)

cat("\n Confusion matrix using PCA: \n")
#confusion matrix
confusionMatrix(predict(logis_model_pc),scaled_wine1[,12])
```

```
DT_train <- rpart(fit, train1, method = "class",
                  minsplit = 2, minbucket = 1, cp = 0.0001,
                  parms = list(split = c("gini")))

rpart.plot(DT_train, digits = 3, fallen.leaves = TRUE)

DT_predict_test <- predict(DT_train, test1, type = "class")

summary(DT_predict_test)

confmat <- confusionMatrix(DT_predict_test, test1[,12])
confmat

cat('\n')
print(paste("Misclassification error for test : ",
            round(1-(sum(diag(confmat$table))/sum(confmat$table)),2)))

#fit random forest model
rf_model <- randomForest(
  formula = fit,
  data = train1,
  ntree = 700
)

#display fitted model
rf_model

#display test MSE by number of trees
plot(rf_model)

#produce variable importance plot
varImpPlot(rf_model)

RF_predict_test <- predict(rf_model, test1, type = "class")

summary(RF_predict_test)

confmat <- confusionMatrix(RF_predict_test, test1[,12])
confmat
```

References

Cortez, Paulo, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. 2009. "Modeling Wine Preferences by Data Mining from Physicochemical Properties." *Decision Support Systems* 47 (4): 547–53. https://doi.org/https://doi.org/10.1016/j.dss.2009.05.016.