3. SVM, Random Forest and Boosting Models

Code:

```
Train.data=read.csv("WineData.csv",header=T)
Test.data=read.csv("WineHoldoutData.csv",header=T)
table(Train.data$quality)

##
##    3    4    5    6    7    8    9
##   23  176 1740 2243  852  159    5

table(Test.data$quality)

##
##   3   4   5   6   7   8
##   7  40 398 593 227  34


Test.data$quality=as.factor(Test.data$quality)

Train.data$quality=ifelse(Train.data$quality==3 | Train.data$quality==4,"A",
                          ifelse(Train.data$quality==5,"B",
                                 ifelse(Train.data$quality==6,"C",
                                        ifelse(Train.data$quality==7,"D","E"
))))
Train.data$quality=as.factor(Train.data$quality)
table(Train.data$quality)

##
##    A    B    C    D    E
##  199 1740 2243  852  164

Test.data$quality=ifelse(Test.data$quality==3 | Test.data$quality==4,"A",
                         ifelse(Test.data$quality==5,"B",
                                ifelse(Test.data$quality==6,"C",
                                       ifelse(Test.data$quality==7,"D","E"))
))
Test.data$quality=as.factor(Test.data$quality)
table(Test.data$quality)

##
##   A   B   C   D   E
##  47 398 593 227  34
```

Explanation:

The quality data is converted to factor for classification. It is seen from the table that there is a large class imbalance. There are only 23 datapoints in label 3 and over 2000 datapoints in label 6. The classes with

very few datapoints are combined (3 and 4, 8 and 9) to provide some additional data to the model, so it can predict better. The new labels are A (3 or 4), B (5), C (6), D (7) and E (8 or 9). Therefore, E has the best wine quality and A has the worst.

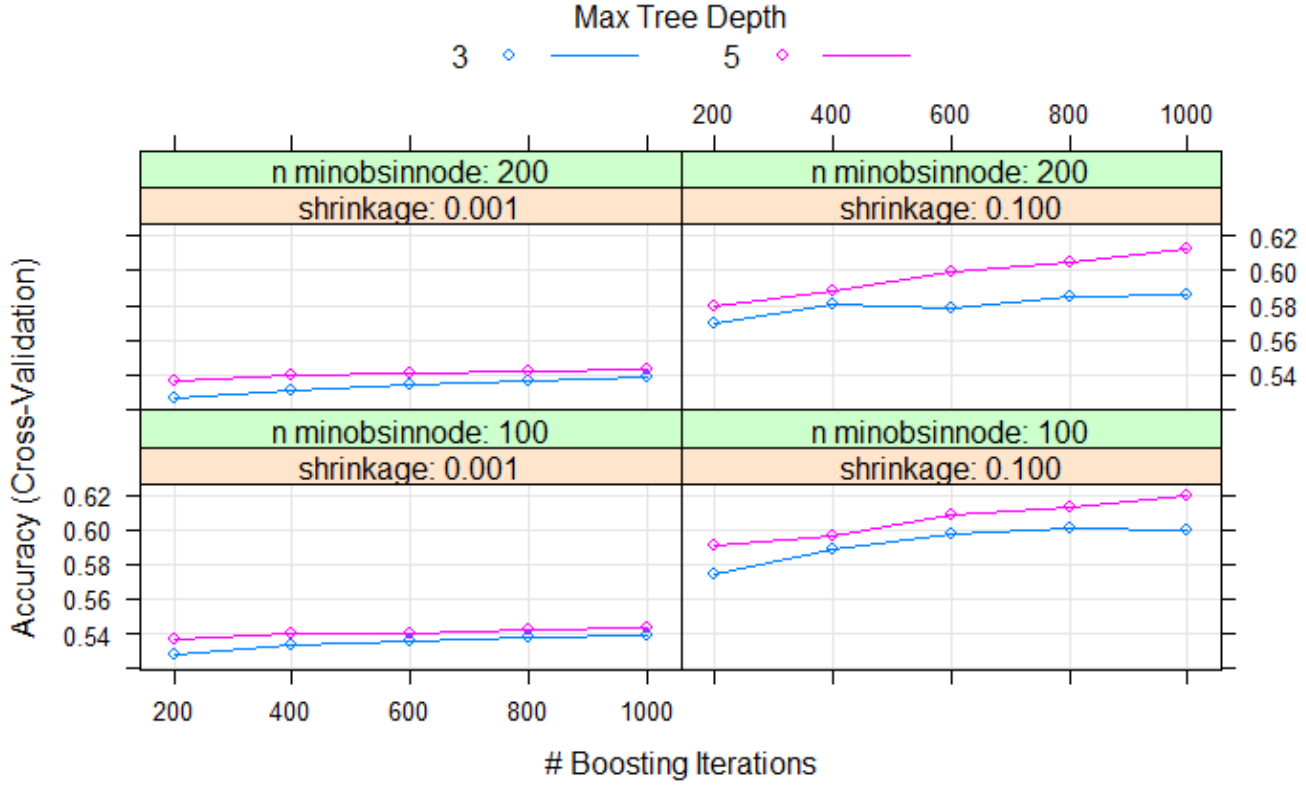
Code for Boosting:

```
#Boosting-----------------------------------------------

gbm.grid=expand.grid(interaction.depth=c(3,5),n.trees=seq(from=200,to=1000,b
y=200),shrinkage=c(0.001,0.1),n.minobsinnode=c(100,200))
trainControl=trainControl(method="cv",number=10)
set.seed(1)
gbm.caret=train(quality~.,data=Train.data,method="gbm",
                trControl=trainControl,verbose=FALSE,
                tuneGrid=gbm.grid)
gbm.caret
```

```
## Stochastic Gradient Boosting
##
## 5198 samples
##   12 predictor
##    5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 4679, 4678, 4677, 4679, 4677, 4678, ...
## Resampling results across tuning parameters:
##
##   shrinkage  interaction.depth  n.minobsinnode  n.trees  Accuracy   Kappa
##   0.001      3                  100              200     0.5277009  0.2184365
##   0.001      3                  100              400     0.5328932  0.2312178
##   0.001      3                  100              600     0.5359746  0.2398263
##   0.001      3                  100              800     0.5378995  0.2458620
##   0.001      3                  100             1000     0.5384749  0.2481033
##   0.001      3                  200              200     0.5257722  0.2135945
##   0.001      3                  200              400     0.5311606  0.2272943
##   0.001      3                  200              600     0.5342397  0.2362582
##   0.001      3                  200              800     0.5365515  0.2422452
##   0.001      3                  200             1000     0.5386684  0.2468114
##   0.001      5                  100              200     0.5371314  0.2435414
##   0.001      5                  100              400     0.5403999  0.2507812
##   0.001      5                  100              600     0.5403980  0.2518725
##   0.001      5                  100              800     0.5427053  0.2564552
##   0.001      5                  100             1000     0.5432830  0.2587779
##   0.001      5                  200              200     0.5361691  0.2433536
##   0.001      5                  200              400     0.5390522  0.2490644
##   0.001      5                  200              600     0.5409790  0.2529149
##   0.001      5                  200              800     0.5423248  0.2558391
##   0.001      5                  200             1000     0.5430940  0.2576034
##   0.100      3                  100              200     0.5748304  0.3340691
##   0.100      3                  100              400     0.5888844  0.3622207
```

```
##   0.100      3                 100              600      0.5977302  0.3790613
##   0.100      3                 100              800      0.6006186  0.3850320
##   0.100      3                 100             1000      0.5998538  0.3855364
##   0.100      3                 200              200      0.5690571  0.3223893
##   0.100      3                 200              400      0.5800253  0.3452655
##   0.100      3                 200              600      0.5786792  0.3481572
##   0.100      3                 200              800      0.5846559  0.3597969
##   0.100      3                 200             1000      0.5858012  0.3614851
##   0.100      5                 100              200      0.5906060  0.3643279
##   0.100      5                 100              400      0.5967646  0.3793455
##   0.100      5                 100              600      0.6086988  0.4004664
##   0.100      5                 100              800      0.6133161  0.4095589
##   0.100      5                 100             1000      0.6200517  0.4207947
##   0.100      5                 200              200      0.5794440  0.3445313
##   0.100      5                 200              400      0.5879185  0.3637513
##   0.100      5                 200              600      0.5988889  0.3835074
##   0.100      5                 200              800      0.6050439  0.3950316
##   0.100      5                 200             1000      0.6127380  0.4067470
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 1000, interaction.dept
h =
##  5, shrinkage = 0.1 and n.minobsinnode = 100.
```

```
plot(gbm.caret)
```

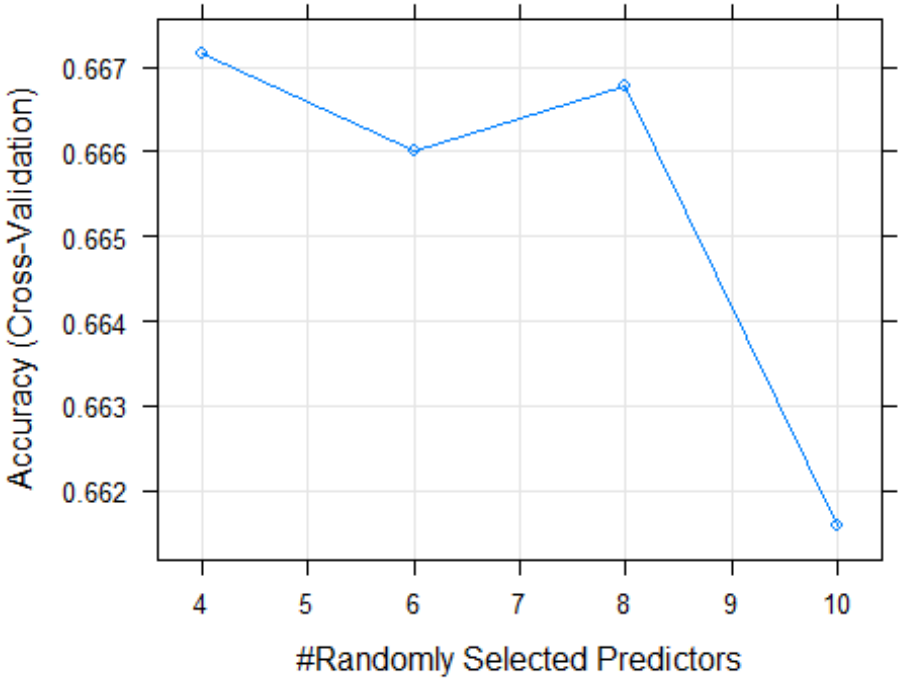


Explanation:

Boosting was used to fit the training data and the parameters used were shrinkage of 0.001 and 0.1, minimum number of observations at node between 100 and 200 and interaction depth between 3 and 5. It is observed that accuracy is higher for higher interaction depth and increases with increasing number of trees. It is also higher for higher shrinkage value of 0.1 as compared to 0.001.

Code for Random Forest:

```
#Random Forest-----------------------------------------------
library(caret)

rf.caretGrid=expand.grid(mtry=c(4,6,8,10))
trainControl=trainControl(method="cv",number=10)
set.seed(1)
rf.caret=train(quality~., data=Train.data,method="rf",
               trControl=trainControl, verbose=FALSE,
               tuneGrid=rf.caretGrid, ntree=500)

print(rf.caret)

## Random Forest
##
## 5198 samples
##   12 predictor
##    5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 4679, 4678, 4677, 4679, 4677, 4678, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    4    0.6671660  0.4811999
##    6    0.6660074  0.4796505
##    8    0.6667759  0.4821338
##   10    0.6615802  0.4739825
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 4.

plot(rf.caret)
```



Explanation:

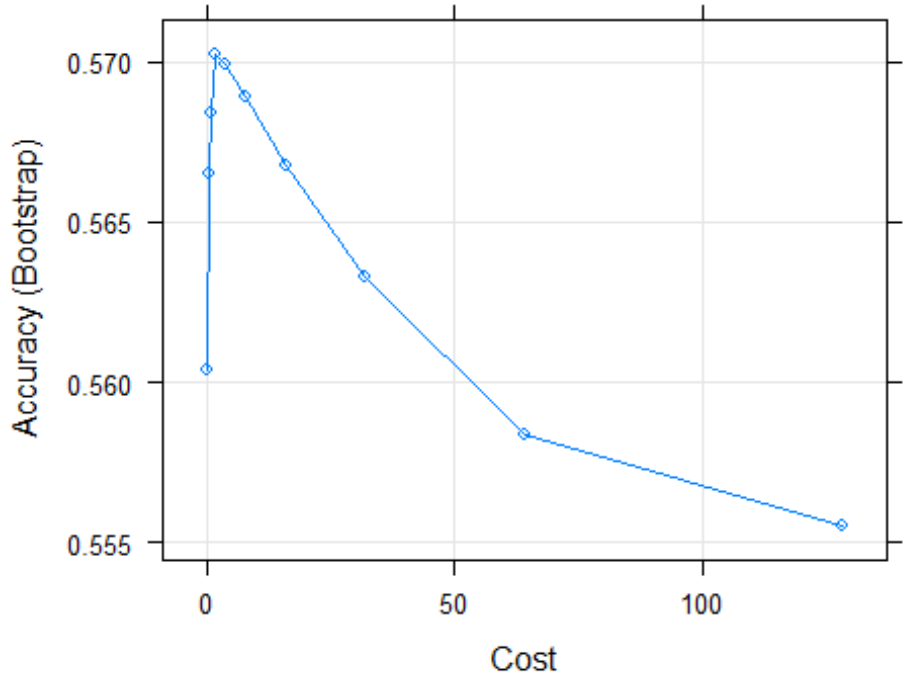
For random forest, the number of variables tried at each node (mtry) is square root of number of predictors (which is equal to 3.4 ~ 4) in this case. Therefore, values of mtry used to determine best model were 4, 6, 8 and 10 and number of trees was kept constant at 500. Best model with highest accuracy was obtained at mtry = 4.

Code for SVM:

```
#SVM-------------------------------------------------

svm.caretGrid=expand.grid(cost=c(0.001,0.01,0.1,1,10,100),gamma=c(0.5,1,2,3,
10))
trainControl=trainControl(method="cv",number=10)

set.seed(1)
svm.caret=train(quality~.,data=Train.data,method="svmRadial",
                preProcess=c("center","scale"),
                tuneLength=10)
svm.caret

## Support Vector Machines with Radial Basis Function Kernel
##
## 5198 samples
##   12 predictor
##    5 classes: 'A', 'B', 'C', 'D', 'E'
##
## Pre-processing: centered (12), scaled (12)
```

```
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 5198, 5198, 5198, 5198, 5198, 5198, ...
## Resampling results across tuning parameters:
##
##   C       Accuracy   Kappa
##     0.25  0.5603576  0.2823606
##     0.50  0.5665194  0.2997065
##     1.00  0.5683830  0.3092842
##     2.00  0.5702515  0.3188358
##     4.00  0.5698869  0.3239847
##     8.00  0.5689173  0.3294474
##    16.00  0.5667807  0.3332346
##    32.00  0.5632620  0.3347674
##    64.00  0.5583949  0.3334332
##   128.00  0.5554997  0.3334569
##
## Tuning parameter 'sigma' was held constant at a value of 0.09179926
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.09179926 and C = 2.
```

```
plot(svm.caret)
```



Explanation:

For SVM, tuneLength = 10 was used to vary the values of cost. Radial kernel was used. Scaling and centering were done to ensure the predictors are on the same scale. From the curve, it is seen that the accuracy increases upto certain extent with increase in cost, and then decreases.

The accuracy for training data for all 3 algorithms is compared in the table below:

Algorithm	**Training Accuracy %**	**Misclassification %**
Boosting	62	38
Random Forest	66.71	33.29
SVM (Radial)	57.02	42.98

Thus, Random Forest has the lowest misclassification rate for training data.

Code for performance on test data:

```
#Predict-------------------------------------------

gbm.predict=predict(gbm.caret,Test.data)
confusionMatrix(gbm.predict,Test.data$quality)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   A   B   C   D   E
##          A   9   8   4   0   0
##          B  25 287 122  12   0
##          C  13  94 406  98  14
##          D   0   9  57 111   4
##          E   0   0   4   6  16
##
## Overall Statistics
##
##                Accuracy : 0.6382
##                  95% CI : (0.6114, 0.6644)
##     No Information Rate : 0.4565
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4431
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity          0.191489   0.7211   0.6847  0.48899  0.47059
## Specificity          0.990415   0.8235   0.6898  0.93470  0.99209
## Pos Pred Value       0.428571   0.6435   0.6496  0.61326  0.61538
## Neg Pred Value       0.970266   0.8699   0.7226  0.89624  0.98586
```

```
## Prevalence           0.036182   0.3064   0.4565  0.17475  0.02617
## Detection Rate       0.006928   0.2209   0.3125  0.08545  0.01232
## Detection Prevalence 0.016166   0.3433   0.4811  0.13934  0.02002
## Balanced Accuracy    0.590952   0.7723   0.6872  0.71184  0.73134

#Accuracy=63.82%

rf.predict=predict(rf.caret,Test.data)
confusionMatrix(rf.predict,Test.data$quality)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   A   B   C   D   E
##          A   6   2   0   0   0
##          B  24 308  96   4   0
##          C  17  85 475  96  11
##          D   0   3  22 126   7
##          E   0   0   0   1  16
##
## Overall Statistics
##
##                Accuracy : 0.7167
##                  95% CI : (0.6913, 0.7411)
##     No Information Rate : 0.4565
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5545
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity          0.127660   0.7739   0.8010   0.5551  0.47059
## Specificity          0.998403   0.8624   0.7040   0.9701  0.99921
## Pos Pred Value       0.750000   0.7130   0.6944   0.7975  0.94118
## Neg Pred Value       0.968242   0.8962   0.8081   0.9115  0.98596
## Prevalence           0.036182   0.3064   0.4565   0.1747  0.02617
## Detection Rate       0.004619   0.2371   0.3657   0.0970  0.01232
## Detection Prevalence 0.006159   0.3326   0.5266   0.1216  0.01309
## Balanced Accuracy    0.563031   0.8181   0.7525   0.7626  0.73490

#Accuracy=71.67%

svmrad.predict=predict(svm.caret,Test.data)
confusionMatrix(svmrad.predict,Test.data$quality)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   A   B   C   D   E
##          A   3   0   0   0   0
##          B  28 261 135   4   0
##          C  16 136 440 163  25
##          D   0   1  18  60   9
##          E   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.5881
##                  95% CI : (0.5608, 0.6151)
##     No Information Rate : 0.4565
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3281
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity          0.063830   0.6558   0.7420  0.26432  0.00000
## Specificity          1.000000   0.8147   0.5184  0.97388  1.00000
## Pos Pred Value       1.000000   0.6098   0.5641  0.68182      NaN
## Neg Pred Value       0.966049   0.8427   0.7052  0.86210  0.97383
## Prevalence           0.036182   0.3064   0.4565  0.17475  0.02617
## Detection Rate       0.002309   0.2009   0.3387  0.04619  0.00000
## Detection Prevalence 0.002309   0.3295   0.6005  0.06774  0.00000
## Balanced Accuracy    0.531915   0.7352   0.6302  0.61910  0.50000
```

```
#Accuracy=58.81%
```

Explanation:

The performance of all three algorithms were compared for holdout data using Misclassification %. The performance is summarized in the table below:

Algorithm	**Test Accuracy %**	**Misclassification %**
Boosting	63.82	36.18
Random Forest	**71.67**	**28.33**
SVM (Radial)	58.81	41.19

From the table, it can be seen that Random Forest has the best performance on holdout data, and SVM has the worst.

4. Errors on modeling as a classification problem

In a classification problem, two classes are combined (3 and 4, 8 and 9) due to very few datapoints in those classes. Therefore, wine having quality 3 and 4 will now be predicted to be belonging to the same class A. This is the information that will be lost in classification problem with class imbalance.

When modeling as a classification problem, class imbalance always causes a problem if the Misclassification % is used as a metric. This is because the model will incorrectly predict the smaller class data as belonging to larger class, simply because there are a larger number of training datapoints in the bigger class. Therefore, due to larger datapoints, the accuracy of the model will still be high even if it incorrectly predicts wrong class for the smaller label. Therefore, misclassification is not an appropriate metric for multi-class problems with class imbalance.

Another metric that can be used is the Kappa value. Kappa measures the agreement between the true value and predicted value and takes into consideration if the values are agreeing just by chance. Higher kappa value indicates better agreement and therefore better classification and score near 0 indicates chance agreement. Here, the kappa values for Boosting, Random Forest and SVM are 0.44, 0.55 and 0.32. According to this metric as well, Random Forest is the best algorithm for given dataset.

Some other classification metrics like Log-loss are applicable if the output is class probabilities and Matthew's Correlation Coefficient is applicable to binary class problems. F1 score was used in this case. Macro F1 score or simply F1 score gives equal weight to all classes regardless of number of observations in that class. Therefore, it is a good metric when there is class imbalance. Another metric called macro-weighted F1 score assigns weights to classes based on the number of observations in that class. Therefore, it would generate bias towards larger classes and so, it is not used here. Some other methods involve plotting the ROC curve by averaging sensitivity and specificity values for each class.

Code for macro-F1 score:

```
#For multiclass when there is class imbalance, F1 score is used to counter c
lass imbalance

library(yardstick)

f_meas_vec(gbm.predict,Test.data$quality,beta=1,estimator="macro")

## [1] 0.5377837

f_meas_vec(rf.predict,Test.data$quality,beta=1,estimator="macro")

## [1] 0.598445

f_meas_vec(svmrad.predict,Test.data$quality,beta=1,estimator="macro")

## [1] 0.4434615
```

Explanation:

The Kappa value, F1 score and misclassification % are summarized in table below:

Algorithm	**Misclassification %**	**F1 score**	**Kappa**
Boosting	36.18	0.5377837	0.44
Random Forest	**28.33**	**0.598445**	**0.55**
SVM (Radial)	41.19	0.4434615	0.32

From the above table, it is observed that F1 score, Kappa and Accuracy % are the highest for Random Forest. This means that Random Forest has the lowest variance for the current data for all 3 algorithms.

5. Important predictors for wine quality

Code for determining important predictors:

```
fit=lm(quality~.,Train.data)
summary(fit)

##
## Call:
## lm(formula = quality ~ ., data = Train.data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.6370 -0.4664 -0.0452  0.4562  3.0052
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)           1.102e+02  1.565e+01   7.041 2.15e-12 ***
## fixed_acidity         9.026e-02  1.774e-02   5.088 3.75e-07 ***
## volatile_acidity     -1.465e+00  8.991e-02 -16.290  < 2e-16 ***
## citric_acid          -3.011e-02  8.887e-02  -0.339 0.734734
## residual_sugar        6.724e-02  6.625e-03  10.149  < 2e-16 ***
## chlorides            -6.322e-01  3.802e-01  -1.663 0.096390 .
## free_sulfur_dioxide   4.313e-03  8.604e-04   5.013 5.55e-07 ***
## total_sulfur_dioxide -1.344e-03  3.641e-04  -3.691 0.000226 ***
## density              -1.097e+02  1.588e+01  -6.912 5.36e-12 ***
## pH                    5.810e-01  1.021e-01   5.693 1.32e-08 ***
## sulphates             7.290e-01  8.569e-02   8.508  < 2e-16 ***
## alcohol               2.240e-01  2.002e-02  11.186  < 2e-16 ***
## stylewhite           -3.603e-01  6.257e-02  -5.759 8.96e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7339 on 5185 degrees of freedom
## Multiple R-squared:  0.302,  Adjusted R-squared:  0.3004
## F-statistic: 186.9 on 12 and 5185 DF,  p-value: < 2.2e-16
```

Executive Summary

Wine quality data was used to train ML algorithms. In regression, the quality was a numeric value, signifying that the quality is derived mathematically, whereas for classification, quality was treated as a class label. Since misclassification rate is not reliable for datasets with class imbalances, macro-F1 score, and kappa value were used to deal with class imbalance and multi-class classification problem. On holdout data, random forest gave the best performance for both regression and classification, indicating that it has the lowest variance of all 3 algorithms for the current dataset. Based on Random Forest's variable importance plot (pg. 14) and Boosting's relative influence plot (pg. 11), following parameters were important in determining wine quality:

- Alcohol
- Volatile acidity
- Free sulfur dioxide
- Sulphates
- Residual sugar
- Density

Since it is not possible to determine the type of influence of predictors from Random Forest and Boosting, Linear Regression model was used to determine the predictor effect on wine quality. It was found that alcohol % is the most important predictor that affects wine quality. As seen from linear regression model, higher alcohol content gives a richer and fruitier flavor to the wine [1]. Volatile acidity (VA) is referencing volatility in wine, which causes it to go bad. As seen from the regression model, increased volatile acidity degrades wine quality, which is accurate [2]. Free sulfur dioxide improves the wine quality by keeping it fresh [3], which is in agreement with the regression model. Similarly, sulphates also improve the quality of wine by preventing microbial growth. Residual sugar is found in wines that have been fermented for longer time, indicating its superior quality with more refined taste [4]. Therefore, residual sugar improves the wine quality.

Factors affecting quality for red and white wines:

```
#Features affecting white wine----------------------------

Train.data$style[Train.data$style=="red"]=NA
white=na.omit(Train.data)
white$style=NULL
lin.fit=lm(quality~.,data=white)
summary(lin.fit)

##
## Call:
## lm(formula = quality ~ ., data = white)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.7824 -0.4972 -0.0382  0.4669  3.1030
##
## Coefficients:
```

```
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)           1.430e+02  2.037e+01   7.019 2.62e-12 ***
## fixed_acidity         5.996e-02  2.338e-02   2.565 0.010368 *
## volatile_acidity     -1.826e+00  1.254e-01 -14.563  < 2e-16 ***
## citric_acid           5.627e-02  1.070e-01   0.526 0.598959
## residual_sugar        8.200e-02  8.253e-03   9.936  < 2e-16 ***
## chlorides            -4.483e-01  6.057e-01  -0.740 0.459197
## free_sulfur_dioxide   3.524e-03  9.472e-04   3.720 0.000202 ***
## total_sulfur_dioxide -2.927e-04  4.267e-04  -0.686 0.492778
## density              -1.434e+02  2.068e+01  -6.936 4.70e-12 ***
## pH                    7.518e-01  1.185e-01   6.344 2.49e-10 ***
## sulphates             6.252e-01  1.130e-01   5.532 3.37e-08 ***
## alcohol               2.126e-01  2.632e-02   8.077 8.74e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7518 on 3892 degrees of freedom
## Multiple R-squared:  0.2921, Adjusted R-squared:  0.2901
## F-statistic:   146 on 11 and 3892 DF,  p-value: < 2.2e-16
```

```
#Features affecting red wine

Train.data$style[is.na(Train.data$style)]="red"
Train.data$style[Train.data$style=="white"]=NA
red=na.omit(Train.data)
red$style=NULL
lin.fit2=lm(quality~.,data=red)
summary(lin.fit2)
```

```
##
## Call:
## lm(formula = quality ~ ., data = red)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.63553 -0.36556 -0.05337  0.43755  2.02004
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)           3.308e+01  2.410e+01   1.373  0.17013
## fixed_acidity         3.221e-02  2.921e-02   1.102  0.27050
## volatile_acidity     -1.089e+00  1.344e-01  -8.106 1.21e-15 ***
## citric_acid          -1.696e-01  1.636e-01  -1.037  0.29998
## residual_sugar        2.586e-02  1.739e-02   1.487  0.13734
## chlorides            -1.569e+00  4.833e-01  -3.246  0.00120 **
## free_sulfur_dioxide   8.649e-04  2.423e-03   0.357  0.72125
## total_sulfur_dioxide -2.630e-03  8.052e-04  -3.267  0.00112 **
```

```
## density              -2.909e+01  2.458e+01  -1.183  0.23693
## pH                   -3.743e-01  2.130e-01  -1.757  0.07916 .
## sulphates             8.960e-01  1.275e-01   7.026 3.43e-12 ***
## alcohol               2.612e-01  3.005e-02   8.693  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6488 on 1282 degrees of freedom
## Multiple R-squared:  0.3532, Adjusted R-squared:  0.3477
## F-statistic: 63.65 on 11 and 1282 DF,  p-value: < 2.2e-16
```

Executive Summary *(contd.)*

For white wine, the factors affecting its quality are fixed and volatile acidity, residual sugar, free sulfur dioxide, pH, sulfates alcohol and density. For red wine the factors affecting quality are volatile acidity, chlorides, total sulfur dioxide, sulfates and alcohol.

It is seen that increase in volatile acidity and density decreases wine quality for both red and white wine, whereas increase in fixed acidity and amount of residual sugar increases the quality for both. Total sulfur dioxide has a foul smell after binding with other compounds so higher concentrations of total sulfur dioxide degrades the wine quality for both wines. On the other hand, free sulfur dioxide helps protect wine from oxidation and spoilage [5], so more free sulfur dioxide and low bonded sulfur dioxide are essential for good wine quality. More basic white wines (high pH) have higher quality score whereas more acidic (low pH) red wines have higher quality. Higher alcohol content and sulfates improves quality for both red and white wine.

References

[1] Prati, S. (2015, April 17). How does alcohol content affect wine? (https://www.tennessean.com/story/life/food/2015/04/17/alcohol-content-affect-wine/25779589/)

[2] Weird Wine Flavors and the Science Behind Them. (2020, February 20). (https://winefolly.com/deep-dive/weird-wine-flavors-and-the-science-behind-them/)

[3] Peynaud E. Knowing and Making Wine. John Wiley & Sons; New York, NY, USA: 1984. pp. 135–136.pp. 267–269

[4] Bighammerwines.com. (n.d.). BHW Real Wine Club Insider Secrets: Residual Sugar. (*https://www.bighammerwines.com/blogs/news/residual-sugar*)

[5] Total Sulfur Dioxide - Why it Matters, Too! (n.d.). (https://www.extension.iastate.edu/wine/total-sulfur-dioxide-why-it-matters-too)