FINAL-PROJECT-

1. Question-1

   Develop SVM, Random Forests, and Boosting based regression model to predict wine quality. Perform hyper parameter tuning using 10-fold Cross Validation (CV). Summarize performance results and identify the best regression model. Report performance of best regression model on holdout data.

Code-

```
train = read.csv("C:¥¥Users¥¥KIRAN KONDISETTI¥¥Desktop¥¥WineData.csv
")
test = read.csv('C:¥¥Users¥¥KIRAN
KONDISETTI¥¥Desktop¥¥WineHoldoutData.csv ')



sum(is.na(train))
sum(is.na(test))
library(Amelia)
missmap(train, main="Train Data - Missings Map",
        col=c("yellow", "black"), legend=FALSE)
missmap(test, main="Test Data - Missings Map",
        col=c("yellow", "black"), legend=FALSE)


boxplot(train)
boxplot(test)
```

```
OutVals = boxplot(train, plot=FALSE)$out
OutVals1 = boxplot(test, plot=FALSE)$out
plot(OutVals1)
plot(OutVals)
boxplot(train)
y = c(1,2,3,4,5,6,7,9,10,11)
for (i in y)
{
  x <- train[,i]
  qnt <- quantile(x, probs=c(.25, .75))
  caps <- quantile(x, probs=c(.05, .95))
  H <- 1.5 * IQR(x)
  x[x < (qnt[1] - H)] <- caps[1]
  x[x > (qnt[2] + H)] <- caps[2]
  train[,i] = x
}
for (i in y)
{
  x <- test[,i]
  qnt <- quantile(x, probs=c(.25, .75))
  caps <- quantile(x, probs=c(.05, .95))
  H <- 1.5 * IQR(x)
  x[x < (qnt[1] - H)] <- caps[1]
  x[x > (qnt[2] + H)] <- caps[2]
  test[,i] = x
}
```

```
train= train[!duplicated(train),]
test = test[!duplicated(test),]

train1 = train
test1=  test

library(randomForest)
rf = randomForest(quality~.,data= train1, mtry= 3.60 , n.trees= 500 )
summary(rf)
pred = predict(rf, newdata = test1)
mean((pred - test1$quality)^2)
importance(rf)
varImpPlot(rf)
library(e1071)
library(MASS)
lr = glm(quality~., data =train)
summary(lr)
svm = svm(quality~., data=train1, kernel='linear', cost= 10)
summary(svm)
pred1 = predict(svm, newdata = test1)
mean((pred1 - test1$quality)^2)

svm1 = svm(quality~., data=train1, kernel='polynomial', cost= 10, degree = 2)
summary(svm1)
pred2 = predict(svm1, newdata = test1)
mean((pred2 - test1$quality)^2)

svm2 = svm(quality~., data=train1, kernel='radial', cost= 10, gamma= 2)
summary(svm2)
```

```
pred3 = predict(svm2, newdata = test1)
mean((pred3 - test1$quality)^2)

library(gbm)
boosting = gbm(quality~., data=train1,distribution="gaussian",n.trees=100 ,
interaction.depth=5)
summary(boosting)
pred4 = predict(boosting, newdata = test1,n.trees= 100)
mean((pred4 - test1$quality)^2)


tune= tune(randomForest,quality~.,data= train1, ranges
=list(mtry=c(3,5,6),n.trees= c(500,100,400)))
summary(tune)
pred.tune = predict(tune$best.model, newdata = test1)
mean((pred.tune - test1$quality)^2)


tune1= tune(svm, quality~.,data= train1,kernel ='linear' , range = list(cost=
c(3,4,5)))
summary(tune1)
pred.tune1 = predict(tune1$best.model, newdata = test1)
mean((pred.tune1 - test1$quality)^2)

tune2= tune(svm,quality~.,data= train1,kernel= 'polynomial', range = list(cost=
c(3,4,5),degree= c(2,3,4)))
summary(tune2)
pred.tune2 = predict(tune2$best.model, newdata = test1)
mean((pred.tune2 - test1$quality)^2)
```

```
tune3= tune(svm,quality~.,data= train1,kernel= 'radial', range = list(cost=
c(3,4,5),gamma= c(10,0.01,0.1)),scale= TRUE)
summary(tune3)
pred.tune3 = predict(tune3$best.model, newdata = test1)
mean((pred.tune3 - test1$quality)^2)

library(caret)
library(gbm)

caretGrid <- expand.grid(interaction.depth=c(1,3,4,2, 5), n.trees = (1:5)*100,
                    shrinkage=c(0.01, 0.001),
                    n.minobsinnode=10)
trainControl <- trainControl(method="cv", number=10)
set.seed(99)
gbm.caret <- train( quality~.,data=train1,distribution="gaussian",
method="gbm",
                trControl=trainControl, verbose=FALSE,
                tuneGrid=caretGrid, bag.fraction=0.75)



print(gbm.caret)
predict_gbm <- predict(gbm.caret,newdata = test1)
mean((predict_gbm - test1$quality)^2)
```

Output-

```
summary(rf)
                Length Class  Mode
call               5   -none- call
type               1   -none- character
predicted       4433   -none- numeric
mse              500   -none- numeric
rsq              500   -none- numeric
```

```
 oob.times        4433   -none- numeric
 importance         12   -none- numeric
 importanceSD        0   -none- NULL
 localImportance     0   -none- NULL
 proximity           0   -none- NULL
 ntree               1   -none- numeric
 mtry                1   -none- numeric
 forest             11   -none- list
 coefs               0   -none- NULL
 y                4433   -none- numeric
 test                0   -none- NULL
 inbag               0   -none- NULL
 terms               3   terms  call

 >
```

```
> pred = predict(rf, newdata = test1)
> mean((pred - test1$quality)^2)
[1] 0.353619
> importance(rf)
                     IncNodePurity
fixed_acidity           189.850140
volatile_acidity        329.284608
citric_acid             227.716462
residual_sugar          240.572610
chlorides               258.371612
free_sulfur_dioxide     278.258763
total_sulfur_dioxide    250.579400
density                 365.003137
pH                      223.735186
sulphates               240.366319
alcohol                 647.968658
style                     9.896365
> varImpPlot(rf)
> summary(svm)

Call:
svm(formula = quality ~ ., data = train1, kernel = "linear", cost = 10)


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  linear
       cost:  10
      gamma:  0.07692308
    epsilon:  0.1


Number of Support Vectors:  3938
```

```
> pred1 = predict(svm, newdata = test1)
> mean((pred1 - test1$quality)^2)
[1] 0.5236219
> summary(svm1)

Call:
svm(formula = quality ~ ., data = train1, kernel = "polynomial", cost =
10, degree = 2)
```

```
Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  polynomial
       cost:  10
     degree:  2
      gamma:  0.07692308
     coef.0:  0
    epsilon:  0.1


Number of Support Vectors:  3930
```

```
> pred2 = predict(svm1, newdata = test1)
> mean((pred2 - test1$quality)^2)
[1] 0.4839679
> summary(svm2)

Call:
svm(formula = quality ~ ., data = train1, kernel = "radial", cost = 10,
gamma = 2)


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  radial
       cost:  10
      gamma:  2
    epsilon:  0.1


Number of Support Vectors:  4134
```

```
> pred3 = predict(svm2, newdata = test1)
> mean((pred3 - test1$quality)^2)
[1] 0.4436906
> summary(boosting)
                                      var   rel.inf
alcohol                           alcohol 41.289873
volatile_acidity         volatile_acidity 13.786460
free_sulfur_dioxide   free_sulfur_dioxide  8.619492
total_sulfur_dioxide total_sulfur_dioxide  6.807215
sulphates                       sulphates  6.475949
citric_acid                   citric_acid  4.878457
residual_sugar             residual_sugar  4.602424
pH                                     pH  3.877992
chlorides                       chlorides  3.753926
density                           density  3.197752
fixed_acidity               fixed_acidity  2.334180
style                               style  0.376279
> pred4 = predict(boosting, newdata = test1,n.trees= 100)
> mean((pred4 - test1$quality)^2)
[1] 0.4671313
> summary(tune)
```

```
Parameter tuning of ‘randomForest’:

- sampling method: 10-fold cross validation

- best parameters:
 mtry n.trees
    3     100

- best performance: 0.4724391

- Detailed performance results:
  mtry n.trees     error dispersion
1    3     500 0.4735494 0.03406022
2    5     500 0.4746772 0.03494738
3    6     500 0.4749754 0.03341758
4    3     100 0.4724391 0.03388449
5    5     100 0.4730533 0.03268685
6    6     100 0.4756811 0.03401635
7    3     400 0.4729257 0.03403943
8    5     400 0.4738272 0.03273083
9    6     400 0.4749516 0.03272062

> pred.tune = predict(tune$best.model, newdata = test1)
> mean((pred.tune - test1$quality)^2)
[1] 0.3541678


 summary(tune1)

 Parameter tuning of ‘svm’:

 - sampling method: 10-fold cross validation

 - best parameters:
  cost
     5

 - best performance: 0.5431084

 - Detailed performance results:
   cost     error dispersion
 1    3 0.5431286 0.05900065
 2    4 0.5431134 0.05901925
 3    5 0.5431084 0.05900812

 > pred.tune1 = predict(tune1$best.model, newdata = test1)
 > mean((pred.tune1 - test1$quality)^2)
 [1] 0.5236431
 > summary(tune2)

 Parameter tuning of ‘svm’:

 - sampling method: 10-fold cross validation

 - best parameters:
  cost degree
     5      2

 - best performance: 0.5056934

 - Detailed performance results:
   cost degree      error  dispersion
 1    3      2  0.5079785  0.04358202
```

```
2     4       2  0.5057572  0.04176807
3     5       2  0.5056934  0.04185304
4     3       3  0.5760307  0.16408261
5     4       3  0.6006576  0.23305636
6     5       3  0.5847613  0.17946036
7     3       4 27.4220753 84.91885259
8     4       4 27.9030363 86.40798165
9     5       4 26.6391151 82.38508296

> pred.tune2 = predict(tune2$best.model, newdata = test1)
> mean((pred.tune2 - test1$quality)^2)
[1] 0.4826443
> summary(tune3)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
    5  0.01

- best performance: 0.5001132

- Detailed performance results:
  cost gamma     error dispersion
1    3 10.00 0.7471470 0.05057976
2    4 10.00 0.7471355 0.05055442
3    5 10.00 0.7471483 0.05053373
4    3  0.01 0.5035804 0.04985586
5    4  0.01 0.5017992 0.04918025
6    5  0.01 0.5001132 0.04883237
7    3  0.10 0.5040378 0.05073573
8    4  0.10 0.5091824 0.05150720
9    5  0.10 0.5138824 0.05230460

> pred.tune3 = predict(tune3$best.model, newdata = test1)
> mean((pred.tune3 - test1$quality)^2)
[1] 0.4814823

>
```

print(gbm.caret)
Stochastic Gradient Boosting

4433 samples
12 predictor

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 3990, 3989, 3989, 3990, 3990, 3990, ...
Resampling results across tuning parameters:

shrinkage	interaction.depth	n.trees	RMSE	Rsquared	MAE
0.001	1	100	0.8691300	0.1826989	0.6859919
0.001	1	200	0.8576338	0.1904343	0.6751792
0.001	1	300	0.8477037	0.1966069	0.6653516
0.001	1	400	0.8391196	0.2009346	0.6562793
0.001	1	500	0.8315671	0.2044322	0.6490554
0.001	2	100	0.8659800	0.2288333	0.6827758

```
  0.001      2                200      0.8516225  0.2338527  0.6687851
  0.001      2                300      0.8391951  0.2385674  0.6559244
  0.001      2                400      0.8284256  0.2421324  0.6440879
  0.001      2                500      0.8190746  0.2452373  0.6387201
  0.001      3                100      0.8637574  0.2424070  0.6808681
  0.001      3                200      0.8478490  0.2462521  0.6652629
  0.001      3                300      0.8343495  0.2500210  0.6510216
  0.001      3                400      0.8227488  0.2546384  0.6404084
  0.001      3                500      0.8127788  0.2584559  0.6346933
  0.001      4                100      0.8627026  0.2559990  0.6801917
  0.001      4                200      0.8458013  0.2595402  0.6639073
  0.001      4                300      0.8314238  0.2627420  0.6489322
  0.001      4                400      0.8191289  0.2662866  0.6389704
  0.001      4                500      0.8086086  0.2696581  0.6328706
  0.001      5                100      0.8619682  0.2659020  0.6795769
  0.001      5                200      0.8444559  0.2692609  0.6627486
  0.001      5                300      0.8295745  0.2721700  0.6477299
  0.001      5                400      0.8169052  0.2747211  0.6381349
  0.001      5                500      0.8060435  0.2775324  0.6310102
  0.010      1                100      0.8061191  0.2163157  0.6402062
  0.010      1                200      0.7809820  0.2485081  0.6230801
  0.010      1                300      0.7677622  0.2635478  0.6125783
  0.010      1                400      0.7595901  0.2743787  0.6049379
  0.010      1                500      0.7537508  0.2834248  0.5989786
  0.010      2                100      0.7869195  0.2610555  0.6211660
  0.010      2                200      0.7588438  0.2811868  0.6031728
  0.010      2                300      0.7445656  0.3025909  0.5898230
  0.010      2                400      0.7359437  0.3150768  0.5811484
  0.010      2                500      0.7300378  0.3238921  0.5749551
  0.010      3                100      0.7790712  0.2742293  0.6143561
  0.010      3                200      0.7492621  0.2989154  0.5930998
  0.010      3                300      0.7345449  0.3189809  0.5793193
  0.010      3                400      0.7258892  0.3314498  0.5705842
  0.010      3                500      0.7203317  0.3395073  0.5647668
  0.010      4                100      0.7736035  0.2846300  0.6105937
  0.010      4                200      0.7429763  0.3104289  0.5881003
  0.010      4                300      0.7279457  0.3301630  0.5736923
  0.010      4                400      0.7194433  0.3419794  0.5644687
  0.010      4                500      0.7142992  0.3491358  0.5588100
  0.010      5                100      0.7702181  0.2916519  0.6073068
  0.010      5                200      0.7382941  0.3188024  0.5841101
  0.010      5                300      0.7231245  0.3380128  0.5694117
  0.010      5                400      0.7150638  0.3489039  0.5603447
  0.010      5                500      0.7102667  0.3555916  0.5545739

Tuning parameter 'n.minobsinnode' was held constant at a value of 10
RMSE was used to select the optimal model using the smallest value.
The final values used for the model were n.trees = 500, interaction.depth = 5, shrinkage = 0.01 and n.mi
> predict_gbm <- predict(gbm.caret,newdata = test1)
```

```
> mean((predict_gbm - test1$quality)^2)
[1] 0.4728238
```

Explanation- Data cleaning is done by removing outliers and duplicates that are present in the data set. Outliers in the data set are removed by using capping. WineData.csv is used for training and for hyper parameter tuning, and WineHoldoutData.csv is used to test the models.

Three regression models namely Random forest, SVM and Boosting are developed using the train dataset and are tested on the holdout data. Hyper parameter tuning for the models are done to improve the accuracy of the models.

The best models of Random forest, SVM and Boosting have a test accuracy of 0.65, 0.52,0 0.53. The best regression model is Random forest since it has a better test accuracy when compared with other models.

2. Question 2

   What are the assumptions made about wine quality data when using a regression model? Do you think it is justified to use a regression model on this data?

   Explanation-

   Assumptions –

   a. We treat the response variable as a continuous variable when regression model is used.
   b. We also assume that there is some relation between wine quality and other predictors.
   c. We also assume that the error is normally distributed.

   I think it is doesn't make complete sense to use a regression model on this data, since in a classification problem, what we are interested in is the probability of an outcome occurring. But in linear regression, when we are prediction a response with different levels (some whole numbers) and there is a chance that the predictions can be a real number, which is not appropriate in this case since quality of the wine is a whole number (regression models is a better approach if the response variable has continuous values).

3. Question 3

   Develop SVM, Random Forests, and Boosting based classification model to predict wine quality. Perform hyper parameter tuning using 10-fold Cross Validation (CV). Summarize performance results and identify the best classification model. Report performance of best classification model on holdout data.

   Code-

   Case-1(removing level 9)

```
install.packages('mltest')
library(mltest)
train2 = train1[!(train1$quality==9),]
test2 = test1
train2$quality = as.factor(train2$quality)
test2$quality= as.factor(test2$quality)
fix(train2)
levels(train2$quality)
rf.c = randomForest(quality~.,data= train2, mtry= 3.60 , n.trees= 500 )
summary(rf.c)
pred.c = predict(rf.c, newdata = test2)
table(pred.c , test2$quality)
ml_test(pred.c , test2$quality)


svm.c = svm(quality~., data=train2, kernel='linear', cost= 10)
summary(svm.c)
pred1.c = predict(svm.c, newdata = test2)
```

```
mean((pred1.c != test2$quality))
table(pred1.c  , test2$quality)
ml_test(pred1.c  , test2$quality)

svm1.c = svm(quality~., data=train2, kernel='polynomial', cost= 10, degree = 2)
summary(svm1.c)
pred2.c = predict(svm1.c, newdata = test2)
mean(pred2.c != test2$quality)
table(pred2.c  , test2$quality)
ml_test(pred2.c  , test2$quality)

svm2.c = svm(quality~., data=train2, kernel='radial', cost= 10, gamma= 2)
summary(svm2.c)
pred3.c = predict(svm2.c, newdata = test2)
mean(pred3.c != test2$quality)
table(pred3.c,test2$quality)
ml_test(pred3.c,test2$quality)

tune.c= tune(randomForest,quality~.,data= train2, range = list(mtry= c(3.6,4,5),n.trees = c(100,300,400)))
summary(tune.c)
pred.tune.c = predict(tune.c$best.model, newdata = test2)
mean(pred.tune.c != test2$quality)
table(pred.tune.c,test2$quality)
ml_test(pred.tune.c,test2$quality)


tune1.c= tune(svm, quality~.,data= train2,kernel ='linear' , range = list(cost= c(3,4,5,6,7,8,9)))
```

```
summary(tune1.c)
pred.tune1.c = predict(tune1.c$best.model, newdata = test2)
mean(pred.tune1.c != test2$quality)
table(pred.tune1.c,test2$quality)
ml_test(pred.tune1.c,test2$quality)

tune2.c= tune(svm, quality~., data= train2 , kernel= 'polynomial', range = list(cost= c(3,4,5),degree= c(2,3,4,5)))
summary(tune2.c)
pred.tune2.c = predict(tune2.c$best.model, newdata = test2)
mean(pred.tune2.c != test2$quality)
table(pred.tune2.c,test2$quality)
ml_test(pred.tune2.c,test2$quality)

tune3.c= tune(svm,quality~.,data= train2,kernel= 'radial', range = list(cost= c(3,4,5),gamma= c(2,10,0.01,0.1)))
summary(tune3.c)
pred.tune3.c = predict(tune3.c$best.model, newdata = test2)
mean(pred.tune3.c != test2$quality)
table(pred.tune3.c != test2$quality)
ml_test(pred.tune3.c != test2$quality)
caretGrid <- expand.grid(interaction.depth=c(1,3,4,2), n.trees = (1:3)*100,
                    shrinkage=c(0.01, 0.001),
                    n.minobsinnode=10)
trainControl <- trainControl(method="cv", number=10)
set.seed(99)
gbm.caret1 <- train( quality~.,data=train2,distribution="multinomial", method="gbm",
                 trControl=trainControl, verbose=FALSE,
```

```
                tuneGrid=caretGrid, bag.fraction=0.75)


print(gbm.caret1)
predict_gbm1 <- predict(gbm.caret1,newdata = test2)
mean(predict_gbm1 !=test2$quality)
table(predict_gbm1 ,test2$quality)
ml_test(test2$quality, predict_gbm1 )
```

Case-2 (replacing quality 9 with 8)

```
train5 = train1
test5 = test1
sum(train5$quality==9)
train5$quality[train5$quality == 9]= 8

train5$quality = as.factor(train5$quality)
test5$quality= as.factor(test5$quality)
fix(train2)
levels(train5$quality)
rf.c.r = randomForest(quality~.,data= train5, mtry= 3.60 , n.trees= 500 )
summary(rf.c.r)
pred.c.r = predict(rf.c.r, newdata = test5)
pred.c.r
mean(pred.c.r != test5$quality)
table(pred.c.r , test5$quality)
ml_test(pred.c.r , test5$quality)
```

```
svm.c.r = svm(quality~., data=train5, kernel='linear', cost= 10)
summary(svm.c.r)
pred1.c.r = predict(svm.c.r, newdata = test5)
mean(pred1.c.r != test5$quality)
ml_test(pred.c.r , test5$quality)
table(pred.c.r , test5$quality)

svm1.c.r = svm(quality~., data=train5, kernel='polynomial', cost= 10, degree = 2)
summary(svm1.c.r)
pred2.c.r = predict(svm1.c.r, newdata = test5)
mean(pred2.c.r != test5$quality)
ml_test(pred.c.r , test5$quality)
table(pred.c.r , test5$quality)

svm2.c.r = svm(quality~., data=train5, kernel='radial', cost= 10, gamma= 2)
summary(svm2.c.r)
pred3.c.r = predict(svm2.c.r, newdata = test5)
mean(pred3.c.r != test5$quality)
ml_test(pred3.c.r , test5$quality)
table(pred3.c.r , test5$quality)

tune.c.r= tune(randomForest,quality~.,data= train5, range = list(mtry= c(3.6,4,5),n.trees = c(100,300,400)))
summary(tune.c.r)
pred.tune.c.r = predict(tune.c.r$best.model, newdata = test5)
mean(pred.tune.c.r != test5$quality)
ml_test(pred.tune.c.r  , test5$quality)
table(pred.tune.c.r  , test5$quality)
```

```
tune1.c.r= tune(svm, quality~.,data= train5,kernel ='linear' , range = list(cost= c(3,4,5,6,7,8,9)))
summary(tune1.c.r)
pred.tune1.c.r = predict(tune1.c.r$best.model, newdata = test5)
mean((pred.tune1.c.r != test5$quality)^2)
ml_test(pred.tune1.c.r , test5$quality)
table(pred.tune.c.r  , test5$quality)

tune2.c.r= tune(svm,quality~.,data= train5,kernel= 'polynomial', range = list(cost= c(3,4,5),degree= c(2,3,4,5)))
summary(tune2.c.r)
pred.tune2.c.r = predict(tune2.c.r$best.model, newdata = test5)
mean((pred.tune2.c.r != test5$quality)^2)
ml_test(pred.c.r , test5$quality)
table(pred.c.r , test5$quality)

tune3.c.r= tune(svm,quality~.,data= train5,kernel= 'radial', range = list(cost= c(3,4,5),gamma= c(2,10,0.01,0.1)))
summary(tune3.c.r)
pred.tune3.c.r = predict(tune3.c.r$best.model, newdata = test5)
mean(pred.tune3.c.r != test5$quality)
ml_test(pred.tune3.c.r , test5$quality)
table(pred.tune3.c.r , test5$quality)
caretGrid <- expand.grid(interaction.depth=c(1,3,4,2), n.trees = (1:3)*100,
                   shrinkage=c(0.01, 0.001),
                   n.minobsinnode=10)
trainControl <- trainControl(method="cv", number=10)
set.seed(99)
```

gbm.caret2 <- train(quality~.,data=train5,distribution="multinomial", method="gbm",

trControl=trainControl, verbose=FALSE,

tuneGrid=caretGrid, bag.fraction=0.75)

print(gbm.caret2)

predict_gbm1 <- predict(gbm.caret2,newdata = test5)

mean(predict_gbm1 !=test5$quality)

table(predict_gbm1 ,test5$quality)

ml_test(test5$quality, predict_gbm1)

<u>Output-</u>

<u>Case-1</u>

```
> summary(rf.c)
                Length Class  Mode
call                5  -none- call
type                1  -none- character
predicted        4428  factor numeric
err.rate         3500  -none- numeric
confusion          42  -none- numeric
votes           26568  matrix numeric
oob.times        4428  -none- numeric
classes             6  -none- character
importance         12  -none- numeric
importanceSD        0  -none- NULL
localImportance     0  -none- NULL
proximity           0  -none- NULL
ntree               1  -none- numeric
mtry                1  -none- numeric
forest             14  -none- list
y                4428  factor numeric
test                0  -none- NULL
inbag               0  -none- NULL
terms               3  terms  call
> pred.c = predict(rf.c, newdata = test2)
mean((pred.c!= test2$quality))
[1] 0.2931034

> table(pred.c , test2$quality)

pred.c   3   4   5   6   7   8
     3   0   0   0   0   0   0
     4   0   6   2   0   0   0
     5   5  18 284  87   4   0
     6   2  15  88 447  92  12
     7   0   0   2  23 111   6
```

```
       8    0    0    0    0    1   13
> summary(svm.c)

Call:
svm(formula = quality ~ ., data = train2, kernel = "linear", cost = 10)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  10

Number of Support Vectors:  4124

 ( 1296 1798 707 170 130 23 )


Number of Classes:  6

Levels:
 3 4 5 6 7 8



> pred1.c = predict(svm.c, newdata = test2)
> mean((pred1.c != test2$quality))
[1] 0.456486
> table(pred1.c  , test2$quality)

pred1.c   3   4   5   6   7   8
      3   0   0   0   0   0   0
      4   0   0   0   0   0   0
      5   4  24 242 137  12   2
      6   3  15 134 420 196  29
      7   0   0   0   0   0   0
      8   0   0   0   0   0   0
> summary(svm1.c)

Call:
svm(formula = quality ~ ., data = train2, kernel = "polynomial", cost =
 10, degree = 2)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  10
     degree:  2
     coef.0:  0

Number of Support Vectors:  3922

 ( 1199 1694 706 170 130 23 )


Number of Classes:  6

Levels:
 3 4 5 6 7 8



> pred2.c = predict(svm1.c, newdata = test2)
> mean(pred2.c != test2$quality)
```

```
[1] 0.4384236
> table(pred2.c  , test2$quality)

pred2.c   3   4   5   6   7   8
      3   0   0   0   0   0   0
      4   0   0   0   0   0   0
      5   3  25 241 135   6   0
      6   4  14 134 406 165  23
      7   0   0   1  16  37   8
      8   0   0   0   0   0   0
> summary(svm2.c)

Call:
svm(formula = quality ~ ., data = train2, kernel = "radial", cost = 10,
 gamma = 2)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  10

Number of Support Vectors:  4402

 ( 1462 1913 704 170 130 23 )


Number of Classes:  6

Levels:
 3 4 5 6 7 8



> pred3.c = predict(svm2.c, newdata = test2)
> mean(pred3.c != test2$quality)
[1] 0.3349754
> table(pred3.c,test2$quality)

pred3.c   3   4   5   6   7   8
      3   0   0   0   0   0   0
      4   0   4   1   1   0   0
      5   2   6 211  57   9   1
      6   5  29 162 483 101  15
      7   0   0   2  16  98   1
      8   0   0   0   0   0  14
> summary(tune.c)

Parameter tuning of ‘randomForest’:

- sampling method: 10-fold cross validation

- best parameters:
 mtry n.trees
    5     400

- best performance: 0.4351782

- Detailed performance results:
  mtry n.trees     error dispersion
1  3.6     100 0.4394743 0.02437159
2  4.0     100 0.4394712 0.02752295
3  5.0     100 0.4378911 0.02902205
4  3.6     300 0.4410524 0.02431434
```

```
5  4.0      300 0.4435319 0.02822201
6  5.0      300 0.4385734 0.02849866
7  3.6      400 0.4392465 0.02511363
8  4.0      400 0.4376633 0.02535214
9  5.0      400 0.4351782 0.02649971

> pred.tune.c = predict(tune.c$best.model, newdata = test2)
> mean(pred.tune.c != test2$quality)
[1] 0.2947455
> table(pred.tune.c,test2$quality)

pred.tune.c   3   4   5   6   7   8
          3   0   0   0   0   0   0
          4   0   6   2   0   0   0
          5   4  16 285  90   4   0
          6   3  17  86 444  93  10
          7   0   0   3  23 111   8
          8   0   0   0   0   0  13
> pred.tune1.c = predict(tune1.c$best.model, newdata = test2)
> mean(pred.tune1.c != test2$quality)
[1] 0.454844
> table(pred.tune1.c,test2$quality)

pred.tune1.c   3   4   5   6   7   8
           3   0   0   0   0   0   0
           4   0   0   0   0   0   0
           5   4  24 244 137  12   2
           6   3  15 132 420 196  29
           7   0   0   0   0   0   0
           8   0   0   0   0   0   0
> pred.tune2.c = predict(tune2.c$best.model, newdata = test2)
> mean(pred.tune2.c != test2$quality)
[1] 0.4244663
> table(pred.tune2.c,test2$quality)

pred.tune2.c   3   4   5   6   7   8
           3   0   0   0   1   0   0
           4   0   1   2   0   0   0
           5   4  23 237 112   6   0
           6   2  15 132 417 156  23
           7   1   0   5  27  46   8
           8   0   0   0   0   0   0
> pred.tune3.c = predict(tune3.c$best.model, newdata = test2)
> mean(pred.tune3.c != test2$quality)
[1] 0.410509
> table(pred.tune3.c != test2$quality)

FALSE  TRUE
  718   500
print(gbm.caret1)
Stochastic Gradient Boosting

4428 samples
  12 predictor
   6 classes: '3', '4', '5', '6', '7', '8'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 3985, 3984, 3985, 3986, 3987, 3986, ...
Resampling results across tuning parameters:

  shrinkage  interaction.depth  n.trees  Accuracy   Kappa
  0.001      1                  100      0.5101724  0.1940931
  0.001      1                  200      0.5113026  0.1944809
```

```
  0.001       1                    300       0.5144598  0.1990030
  0.001       2                    100       0.5232696  0.1978072
  0.001       2                    200       0.5241679  0.2002518
  0.001       2                    300       0.5232639  0.1998567
  0.001       3                    100       0.5239468  0.2043641
  0.001       3                    200       0.5264278  0.2105169
  0.001       3                    300       0.5316284  0.2226277
  0.001       4                    100       0.5291361  0.2231207
  0.001       4                    200       0.5361385  0.2362031
  0.001       4                    300       0.5386241  0.2416846
  0.010       1                    100       0.5194270  0.2097906
  0.010       1                    200       0.5295957  0.2305528
  0.010       1                    300       0.5359148  0.2448222
  0.010       2                    100       0.5332065  0.2269786
  0.010       2                    200       0.5379418  0.2441742
  0.010       2                    300       0.5408686  0.2545610
  0.010       3                    100       0.5417895  0.2478165
  0.010       3                    200       0.5456121  0.2601209
  0.010       3                    300       0.5483271  0.2690110
  0.010       4                    100       0.5433563  0.2550539
  0.010       4                    200       0.5478695  0.2673759
  0.010       4                    300       0.5487709  0.2734662

Tuning parameter 'n.minobsinnode' was held constant at a value of 10
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were n.trees = 300, interaction.dep
th = 4, shrinkage = 0.01 and n.minobsinnode = 10.
> predict_gbm1 <- predict(gbm.caret1,newdata = test2)
> mean(predict_gbm1 !=test2$quality)
[1] 0.4334975
> table(predict_gbm1 ,test2$quality)

predict_gbm1   3   4   5   6   7   8
           3   0   0   0   0   0   0
           4   0   1   2   1   0   0
           5   5  21 242 130  10   1
           6   2  17 131 401 150  23
           7   0   0   1  25  45   6
           8   0   0   0   0   3   1
```

Case-2

```
summary(rf.c.r)
                Length Class  Mode
call                5  -none- call
type                1  -none- character
predicted        4433  factor numeric
err.rate         3500  -none- numeric
confusion          42  -none- numeric
votes           26598  matrix numeric
oob.times        4433  -none- numeric
classes             6  -none- character
importance         12  -none- numeric
importanceSD        0  -none- NULL
localImportance     0  -none- NULL
proximity           0  -none- NULL
ntree               1  -none- numeric
mtry                1  -none- numeric
forest             14  -none- list
y                4433  factor numeric
```

```
test                 0   -none- NULL
inbag                0   -none- NULL
terms                3   terms  call
> pred.c.r = predict(rf.c.r, newdata = test5)
> mean(pred.c.r != test5$quality)
[1] 0.2939245
> table(pred.c.r , test5$quality)

pred.c.r   3   4   5   6   7   8
       3   0   0   0   0   0   0
       4   0   6   2   0   0   0
       5   4  18 283  88   3   0
       6   2  15  89 446  93  12
       7   1   0   2  23 112   6
       8   0   0   0   0   0  13
> pred1.c.r = predict(svm.c.r, newdata = test5)
> mean(pred1.c.r != test5$quality)
[1] 0.456486
> table(pred.c.r , test5$quality)

pred.c.r   3   4   5   6   7   8
       3   0   0   0   0   0   0
       4   0   6   2   0   0   0
       5   4  18 283  88   3   0
       6   2  15  89 446  93  12
       7   1   0   2  23 112   6
       8   0   0   0   0   0  13
> summary(svm1.c.r)

Call:
svm(formula = quality ~ ., data = train5, kernel = "polynomial", cost =
 10, degree = 2)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  10
     degree:  2
     coef.0:  0

Number of Support Vectors:  3918

 ( 1198 1686 706 170 135 23 )


Number of Classes:  6

Levels:
 3 4 5 6 7 8



> pred2.c.r = predict(svm1.c.r, newdata = test5)
> mean(pred2.c.r != test5$quality)
[1] 0.4392447
> table(pred.c.r , test5$quality)

pred.c.r   3   4   5   6   7   8
       3   0   0   0   0   0   0
       4   0   6   2   0   0   0
       5   4  18 283  88   3   0
       6   2  15  89 446  93  12
       7   1   0   2  23 112   6
```

```
       8    0    0    0    0    0   13
> summary(svm2.c.r)

Call:
svm(formula = quality ~ ., data = train5, kernel = "radial", cost = 10,
 gamma = 2)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  10

Number of Support Vectors:  4407

 ( 1462 1913 704 170 135 23 )


Number of Classes:  6

Levels:
 3 4 5 6 7 8



> pred3.c.r = predict(svm2.c.r, newdata = test5)
> mean(pred3.c.r != test5$quality)
[1] 0.3349754
> table(pred3.c.r , test5$quality)

pred3.c.r   3   4   5   6   7   8
        3   0   0   0   0   0   0
        4   0   4   1   1   0   0
        5   2   6 211  57   9   1
        6   5  29 162 483 101  15
        7   0   0   2  16  98   1
        8   0   0   0   0   0  14
> summary(tune.c.r)

Parameter tuning of ‘randomForest’:

- sampling method: 10-fold cross validation

- best parameters:
 mtry n.trees
  3.6     300

- best performance: 0.4301822

- Detailed performance results:
  mtry n.trees     error dispersion
1  3.6     100 0.4344666 0.02544456
2  4.0     100 0.4369502 0.02624513
3  5.0     100 0.4371769 0.02705364
4  3.6     300 0.4301822 0.02634986
5  4.0     300 0.4389772 0.02668663
6  5.0     300 0.4383015 0.02657417
7  3.6     400 0.4364982 0.02470734
8  4.0     400 0.4385283 0.02575580
9  5.0     400 0.4401089 0.02734714

> pred.tune.c.r = predict(tune.c.r$best.model, newdata = test5)
> mean(pred.tune.c.r != test5$quality)
[1] 0.2980296
```

```
> table(pred.tune.c.r   , test5$quality)

pred.tune.c.r   3   4   5   6   7   8
            3   0   0   0   0   0   0
            4   0   6   2   0   0   0
            5   4  19 281  91   4   0
            6   3  14  90 445  94  12
            7   0   0   3  21 110   6
            8   0   0   0   0   0  13
> summary(tune1.c.r)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost
    4

- best performance: 0.4619918

- Detailed performance results:
  cost     error dispersion
1    3 0.4622176 0.02458717
2    4 0.4619918 0.02493500
3    5 0.4619918 0.02493500
4    6 0.4622176 0.02488472
5    7 0.4619918 0.02493500
6    8 0.4619918 0.02493500
7    9 0.4619918 0.02493500

> pred.tune1.c.r = predict(tune1.c.r$best.model, newdata = test5)
> mean((pred.tune1.c.r != test5$quality)^2)
[1] 0.455665
> table(pred.tune.c.r   , test5$quality)

pred.tune.c.r   3   4   5   6   7   8
            3   0   0   0   0   0   0
            4   0   6   2   0   0   0
            5   4  19 281  91   4   0
            6   3  14  90 445  94  12
            7   0   0   3  21 110   6
            8   0   0   0   0   0  13
> summary(tune2.c.r)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost degree
    5      3

- best performance: 0.4443907

- Detailed performance results:
   cost degree     error dispersion
1     3      2 0.4558920 0.02359617
2     4      2 0.4570191 0.02474219
3     5      2 0.4563404 0.02494857
4     3      3 0.4516101 0.02371887
5     4      3 0.4491296 0.02149696
6     5      3 0.4443907 0.02405709
7     3      4 0.4536361 0.02369485
```

```
8      4      4 0.4545442 0.01970349
9      5      4 0.4516091 0.02289741
10     3      5 0.4721285 0.02894602
11     4      5 0.4703247 0.03041640
12     5      5 0.4691950 0.02894444

> pred.tune2.c.r = predict(tune2.c.r$best.model, newdata = test5)
> mean((pred.tune2.c.r != test5$quality)^2)
[1] 0.4211823
> table(pred.c.r , test5$quality)

pred.c.r   3   4   5   6   7   8
       3   0   0   0   0   0   0
       4   0   6   2   0   0   0
       5   4  18 283  88   3   0
       6   2  15  89 446  93  12
       7   1   0   2  23 112   6
       8   0   0   0   0   0  13
> summary(tune3.c.r)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
    3   0.1

- best performance: 0.4423591

- Detailed performance results:
   cost gamma     error dispersion
1     3  2.00 0.4829754 0.02836210
2     4  2.00 0.4834264 0.02835077
3     5  2.00 0.4829749 0.02764535
4     3 10.00 0.5382374 0.02686545
5     4 10.00 0.5380117 0.02728139
6     5 10.00 0.5380117 0.02728139
7     3  0.01 0.4606369 0.01968251
8     4  0.01 0.4601829 0.02035122
9     5  0.01 0.4579286 0.01953790
10    3  0.10 0.4423591 0.02197205
11    4  0.10 0.4437100 0.02076027
12    5  0.10 0.4459678 0.01976991

> pred.tune3.c.r = predict(tune3.c.r$best.model, newdata = test5)
> mean(pred.tune3.c.r != test5$quality)
[1] 0.410509
> table(pred.tune3.c.r , test5$quality)

pred.tune3.c.r   3   4   5   6   7   8
             3   0   0   0   0   0   0
             4   0   3   1   0   0   0
             5   5  23 253 126   2   0
             6   1  13 118 408 152  24
             7   1   0   4  23  54   7
             8   0   0   0   0   0   0
print(gbm.caret2)
Stochastic Gradient Boosting

4433 samples
  12 predictor
   6 classes: '3', '4', '5', '6', '7', '8'
```

```
No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 3989, 3990, 3991, 3990, 3989, 3989, ...
Resampling results across tuning parameters:

  shrinkage  interaction.depth  n.trees  Accuracy   Kappa
  0.001      1                  100      0.5098118  0.1940698
  0.001      1                  200      0.5107122  0.1940380
  0.001      1                  300      0.5136478  0.1978896
  0.001      2                  100      0.5231149  0.2003299
  0.001      2                  200      0.5226624  0.1996659
  0.001      2                  300      0.5233376  0.2006778
  0.001      3                  100      0.5233406  0.2060598
  0.001      3                  200      0.5255960  0.2122789
  0.001      3                  300      0.5292097  0.2197791
  0.001      4                  100      0.5301152  0.2252604
  0.001      4                  200      0.5325922  0.2297170
  0.001      4                  300      0.5364292  0.2380923
  0.010      1                  100      0.5159025  0.2041913
  0.010      1                  200      0.5246925  0.2226667
  0.010      1                  300      0.5328210  0.2386925
  0.010      2                  100      0.5301086  0.2214100
  0.010      2                  200      0.5373356  0.2430366
  0.010      2                  300      0.5409464  0.2532123
  0.010      3                  100      0.5368842  0.2398375
  0.010      3                  200      0.5416231  0.2539932
  0.010      3                  300      0.5440985  0.2628170
  0.010      4                  100      0.5422896  0.2536902
  0.010      4                  200      0.5459034  0.2650968
  0.010      4                  300      0.5461322  0.2695668

Tuning parameter 'n.minobsinnode' was held constant at a value of 10
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were n.trees = 300, interaction.dep
th = 4, shrinkage = 0.01 and n.minobsinnode = 10.
> predict_gbm1 <- predict(gbm.caret2,newdata = test5)
> mean(predict_gbm1 !=test5$quality)
[1] 0.4277504
> table(predict_gbm1 ,test5$quality)

predict_gbm1   3   4   5   6   7   8
           3   0   0   0   0   0   0
           4   0   1   2   1   0   0
           5   5  21 244 127  10   0
           6   2  17 129 406 151  25
           7   0   0   1  23  44   4
           8   0   0   0   0   3   2
```

Explanation-

In this question, SVM, Random Forests, and Boosting based classification model are used to predict wine quality. Data set can be modified to make it a classification-based problem. The first step is converting the response variable in-to a factor using as.factor. The levels of the response in test data is one more than the levels present in the train data. For this reason, I have done this classification in two cases. In the first case I have removed the rows with quality value equal to 9. In this case, the best models of Random forest, SVM (radial Kernel) and Boosting after hyper parameter tuning have a misclassification rate of 0.293, 0.336, 0.4334.

In the second case I have replaced the rows quality equal to 9 with 8, since its next highest quality level. In this case, the best models of Random forest, SVM (radial Kernel) and Boosting after hyper parameter tuning have a misclassification rate of 0.293, 0.33, 0.4227.

It can be seen the accuracies in both the cases are nearly equal to each other. I am considering the case where the quality level 9 is removed since there are only 5 rows with quality equal to 9 in the train data set. The best classification model is Random forest in case-1, since it has a better test accuracy when compared with other models (misclassification rate of 0.293).

4. Question-4

What information / detail about wine quality rating is lost when modeled as a classification problem? What kind of misclassification errors can this lead to? Based on this, suggest alternate supplemental metric that can be used in addition to standard misclassification rate. Document the "Misclassification Rate" and "Suggested Supplemental Misclassification Metric" for the three classification models on the holdout dataset.

Code-

```
install.packages('mltest')
library(mltest)
train2 = train1[!(train1$quality==9),]
test2 = test1
train2$quality = as.factor(train2$quality)
test2$quality= as.factor(test2$quality)
fix(train2)
levels(train2$quality)
```

```
rf.c = randomForest(quality~.,data= train2, mtry= 3.60 , n.trees= 500 )
summary(rf.c)
pred.c = predict(rf.c, newdata = test2)
mean((pred.c!= test2$quality))
table(pred.c , test2$quality)
ml_test(pred.c , test2$quality)


svm.c = svm(quality~., data=train2, kernel='linear', cost= 10)
summary(svm.c)
pred1.c = predict(svm.c, newdata = test2)
mean((pred1.c != test2$quality))
table(pred1.c  , test2$quality)
ml_test(pred1.c  , test2$quality)

svm1.c = svm(quality~., data=train2, kernel='polynomial', cost= 10, degree = 2)
summary(svm1.c)
pred2.c = predict(svm1.c, newdata = test2)
mean(pred2.c != test2$quality)
table(pred2.c  , test2$quality)
ml_test(pred2.c  , test2$quality)

svm2.c = svm(quality~., data=train2, kernel='radial', cost= 10, gamma= 2)
summary(svm2.c)
pred3.c = predict(svm2.c, newdata = test2)
mean(pred3.c != test2$quality)
table(pred3.c,test2$quality)
ml_test(pred3.c,test2$quality)
```

```
tune.c= tune(randomForest,quality~.,data= train2, range = list(mtry=
c(3.6,4,5),n.trees = c(100,300,400)))
summary(tune.c)
pred.tune.c = predict(tune.c$best.model, newdata = test2)
mean(pred.tune.c != test2$quality)
table(pred.tune.c,test2$quality)
ml_test(pred.tune.c,test2$quality)


tune1.c= tune(svm, quality~.,data= train2,kernel ='linear' , range = list(cost=
c(3,4,5,6,7,8,9)))
summary(tune1.c)
pred.tune1.c = predict(tune1.c$best.model, newdata = test2)
mean(pred.tune1.c != test2$quality)
table(pred.tune1.c,test2$quality)
ml_test(pred.tune1.c,test2$quality)

tune2.c= tune(svm, quality~., data= train2 , kernel= 'polynomial', range =
list(cost= c(3,4,5),degree= c(2,3,4,5)))
summary(tune2.c)
pred.tune2.c = predict(tune2.c$best.model, newdata = test2)
mean(pred.tune2.c != test2$quality)
table(pred.tune2.c,test2$quality)
ml_test(pred.tune2.c,test2$quality)

tune3.c= tune(svm,quality~.,data= train2,kernel= 'radial', range = list(cost=
c(3,4,5),gamma= c(2,10,0.01,0.1)))
summary(tune3.c)
pred.tune3.c = predict(tune3.c$best.model, newdata = test2)
```

```
mean(pred.tune3.c != test2$quality)
table(pred.tune3.c != test2$quality)
ml_test(pred.tune3.c != test2$quality)
caretGrid <- expand.grid(interaction.depth=c(1,3,4,2), n.trees = (1:3)*100,
             shrinkage=c(0.01, 0.001),
             n.minobsinnode=10)
trainControl <- trainControl(method="cv", number=10)
set.seed(99)
gbm.caret1 <- train( quality~.,data=train2,distribution="multinomial", method="gbm",
          trControl=trainControl, verbose=FALSE,
          tuneGrid=caretGrid, bag.fraction=0.75)


print(gbm.caret1)
predict_gbm1 <- predict(gbm.caret1,newdata = test2)
mean(predict_gbm1 !=test2$quality)
table(predict_gbm1 ,test2$quality)
ml_test(test2$quality, predict_gbm1 )
```

Outputs-

```
summary(rf.c)
                Length Class  Mode
call                5  -none- call
type                1  -none- character
predicted        4428  factor numeric
err.rate         3500  -none- numeric
confusion          42  -none- numeric
votes           26568  matrix numeric
oob.times        4428  -none- numeric
classes             6  -none- character
importance         12  -none- numeric
importanceSD        0  -none- NULL
localImportance     0  -none- NULL
proximity           0  -none- NULL
ntree               1  -none- numeric
mtry                1  -none- numeric
forest             14  -none- list
y                4428  factor numeric
```

```
test                 0   -none- NULL
inbag                0   -none- NULL
terms                3   terms  call
> mean((pred.c!= test2$quality))
[1] 0.2931034
> ml_test(pred.c , test2$quality)
$accuracy
[1] 0.7068966

$balanced.accuracy
        3         4         5         6         7         8 
0.5000000 0.5757562 0.7951704 0.7335200 0.7469806 0.7090885 

$DOR
        3         4         5         6         7         8 
      NaN  77.72727  15.62433   8.04950  27.68540 612.44444 

$error.rate
[1] 0.2931034

$F0.5
        3         4         5         6         7         8 
      NaN 0.4225352 0.7215447 0.7026092 0.7152062 0.7471264 

$F1
        3         4         5         6         7         8 
      NaN 0.2553191 0.7338501 0.7370157 0.6342857 0.5777778 

$F2
        3         4         5         6         7         8 
      NaN 0.1829268 0.7465825 0.7749653 0.5698152 0.4710145 

$FDR
         3          4          5          6          7          8 
       NaN 0.25000000 0.28643216 0.31859756 0.21830986 0.07142857 

$FNR
        3         4         5         6         7         8 
1.0000000 0.8461538 0.2446809 0.1974865 0.4663462 0.5806452 

$FOR
          3           4           5           6           7           8 

0.008064516 0.037162162 0.137518685 0.209923664 0.114521842 0.020785219 


$FPR
          3           4           5           6           7           8 

0.000000000 0.002333722 0.164978292 0.335473515 0.039692702 0.001177856 


$geometric.mean
        3         4         5         6         7         8 
0.0000000 0.3917743 0.7941712 0.7302681 0.7158713 0.6471946 

$Jaccard
        3         4         5         6         7         8 
0.0000000 0.1463415 0.5795918 0.5835509 0.4644351 0.4062500 

$L
         3          4          5          6          7          8 
       NaN  65.923077   4.578294   2.392181  13.444634 356.032258 
```

```
$lambda
        3         4         5         6         7         8 
1.0000000 0.8481332 0.2930233 0.2971838 0.4856218 0.5813299 

$MCC
        3         4         5         6         7         8 
      NaN 0.3286393 0.5831512 0.4692541 0.5740690 0.6161293 

$MK
        3         4         5         6         7         8 
      NaN 0.7128378 0.5760492 0.4714788 0.6671683 0.9077862 

$NPV
        3         4         5         6         7         8 
0.9919355 0.9628378 0.8624813 0.7900763 0.8854782 0.9792148 

$OP
          3           4           5           6           7           8 

-0.29310345 -0.02589634  0.65677990  0.61283846  0.42131118  0.29829642 


$precision
        3         4         5         6         7         8 
      NaN 0.7500000 0.7135678 0.6814024 0.7816901 0.9285714 

$recall
        3         4         5         6         7         8 
0.0000000 0.1538462 0.7553191 0.8025135 0.5336538 0.4193548 

$specificity
        3         4         5         6         7         8 
1.0000000 0.9976663 0.8350217 0.6645265 0.9603073 0.9988221 

$Youden
        3         4         5         6         7         8 
0.0000000 0.1515124 0.5903409 0.4670399 0.4939611 0.4181770 

> summary(svm.c)

Call:
svm(formula = quality ~ ., data = train2, kernel = "linear", cost = 10)


Parameters:
   SVM-Type:  C-classification 
 SVM-Kernel:  linear 
       cost:  10 

Number of Support Vectors:  4124

 ( 1296 1798 707 170 130 23 )


Number of Classes:  6 

Levels: 
 3 4 5 6 7 8



> mean((pred1.c != test2$quality))
[1] 0.456486
> ml_test(pred1.c  , test2$quality)
```

```
$accuracy
[1] 0.543514

$balanced.accuracy
        3         4         5         6         7         8 
0.5000000 0.5000000 0.6723928 0.5724963 0.5000000 0.5000000 

$DOR
       3        4        5        6        7        8 
     NaN      NaN 4.237472 1.967899      NaN      NaN 

$error.rate
[1] 0.456486

$F0.5
        3         4         5         6         7         8 
      NaN       NaN 0.5873786 0.5607477       NaN       NaN 

$F1
        3         4         5         6         7         8 
      NaN       NaN 0.6072773 0.6203840       NaN       NaN 

$F2
        3         4         5         6         7         8 
      NaN       NaN 0.6285714 0.6942149       NaN       NaN 

$FDR
        3         4         5         6         7         8 
      NaN       NaN 0.4251781 0.4730238       NaN       NaN 

$FNR
        3         4         5         6         7         8 
1.0000000 1.0000000 0.3563830 0.2459605 1.0000000 1.0000000 

$FOR
         3          4          5          6          7          8 
0.01046338 0.05563481 0.24187726 0.36147757 0.23908046 0.04473304 

$FPR
        3         4         5         6         7         8 
0.0000000 0.0000000 0.2988314 0.6090468 0.0000000 0.0000000 

$geometric.mean
        3         4         5         6         7         8 
0.0000000 0.0000000 0.6717768 0.5429495 0.0000000 0.0000000 

$Jaccard
        3         4         5         6         7         8 
0.0000000 0.0000000 0.4360360 0.4496788 0.0000000 0.0000000 

$L
       3        4        5        6        7        8 
     NaN      NaN 2.153780 1.238065      NaN      NaN 

$lambda
        3         4         5         6         7         8 
1.0000000 1.0000000 0.5082700 0.6291304 1.0000000 1.0000000 

$MCC
        3         4         5         6         7         8 
      NaN       NaN 0.3388134 0.1549067       NaN       NaN 

$MK
        3         4         5         6         7         8 
```

```
      NaN       NaN 0.3329446 0.1654986       NaN       NaN

$NPV
        3         4         5         6         7         8 
0.9895366 0.9443652 0.7581227 0.6385224 0.7609195 0.9552670 

$OP
         3          4          5          6          7          8 
-0.4564860 -0.4564860  0.5007178  0.2264059 -0.4564860 -0.4564860 

$precision
        3         4         5         6         7         8 
      NaN       NaN 0.5748219 0.5269762       NaN       NaN 

$recall
        3         4         5         6         7         8 
0.0000000 0.0000000 0.6436170 0.7540395 0.0000000 0.0000000 

$specificity
        3         4         5         6         7         8 
1.0000000 1.0000000 0.7011686 0.3909532 1.0000000 1.0000000 

$Youden
        3         4         5         6         7         8 
0.0000000 0.0000000 0.3447856 0.1449926 0.0000000 0.0000000 

> summary(svm1.c)

Call:
svm(formula = quality ~ ., data = train2, kernel = "polynomial", cost = 
 10, degree = 2)


Parameters:
   SVM-Type:  C-classification 
 SVM-Kernel:  polynomial 
       cost:  10 
     degree:  2 
     coef.0:  0 

Number of Support Vectors:  3922

 ( 1199 1694 706 170 130 23 )


Number of Classes:  6 

Levels: 
 3 4 5 6 7 8



> mean(pred2.c != test2$quality)
[1] 0.4384236
> ml_test(pred2.c  , test2$quality)
$accuracy
[1] 0.5615764

$balanced.accuracy
        3         4         5         6         7         8 
0.5000000 0.5000000 0.6824068 0.5893715 0.5703411 0.5000000 

$DOR
       3        4        5        6        7        8 
```

```
     NaN      NaN 4.679509 2.198442 5.599766      NaN

$error.rate
[1] 0.4384236

$F0.5
        3         4         5         6         7         8 
      NaN       NaN 0.5977183 0.5732844 0.4057018       NaN 

$F1
        3         4         5         6         7         8 
      NaN       NaN 0.6132316 0.6231773 0.2740741       NaN 

$F2
        3         4         5         6         7         8 
      NaN       NaN 0.6295716 0.6825824 0.2069351       NaN 

$FDR
        3         4         5         6         7         8 
      NaN       NaN 0.4121951 0.4557641 0.4032258       NaN 

$FNR
        3         4         5         6         7         8 
1.0000000 1.0000000 0.3590426 0.2710952 0.8221154 1.0000000 

$FOR
         3          4          5          6          7          8 
0.01013025 0.05394191 0.23356401 0.35198135 0.20904645 0.04335664 

$FPR
         3          4          5          6          7          8 
0.00000000 0.00000000 0.27614379 0.55016181 0.03720238 0.00000000 

$geometric.mean
        3         4         5         6         7         8 
0.0000000 0.0000000 0.6811468 0.5726161 0.4138440 0.0000000 

$Jaccard
        3         4         5         6         7         8 
0.0000000 0.0000000 0.4422018 0.4526198 0.1587983 0.0000000 

$L
       3        4        5        6        7        8 
     NaN      NaN 2.321100 1.324892 4.781538      NaN 

$lambda
        3         4         5         6         7         8 
1.0000000 1.0000000 0.4960136 0.6026504 0.8538818 1.0000000 

$MCC
        3         4         5         6         7         8 
      NaN       NaN 0.3594884 0.1853757 0.2335517       NaN 

$MK
        3         4         5         6         7         8 
      NaN       NaN 0.3542409 0.1922546 0.3877277       NaN 

$NPV
        3         4         5         6         7         8 
0.9898698 0.9460581 0.7664360 0.6480186 0.7909535 0.9566434 

$OP
         3          4          5          6          7          8 
-0.4384236 -0.4384236  0.5008364  0.3248270 -0.1265320 -0.4384236 
```

```
$precision
        3         4         5         6         7         8 
      NaN       NaN 0.5878049 0.5442359 0.5967742       NaN 

$recall
        3         4         5         6         7         8 
0.0000000 0.0000000 0.6409574 0.7289048 0.1778846 0.0000000 

$specificity
        3         4         5         6         7         8 
1.0000000 1.0000000 0.7238562 0.4498382 0.9627976 1.0000000 

$Youden
        3         4         5         6         7         8 
0.0000000 0.0000000 0.3648137 0.1787430 0.1406822 0.0000000 

> summary(svm2.c)

Call:
svm(formula = quality ~ ., data = train2, kernel = "radial", cost = 10, 
 gamma = 2)


Parameters:
   SVM-Type:  C-classification 
 SVM-Kernel:  radial 
       cost:  10 

Number of Support Vectors:  4402

 ( 1462 1913 704 170 130 23 )


Number of Classes:  6 

Levels: 
 3 4 5 6 7 8



> mean(pred3.c != test2$quality)
[1] 0.3349754
> ml_test(pred3.c,test2$quality)
$accuracy
[1] 0.6650246

$balanced.accuracy
        3         4         5         6         7         8 
0.5000000 0.5500444 0.7249471 0.6894413 0.7225810 0.7258065 

$DOR
        3         4         5         6         7         8 
      NaN 46.057143 10.213253  6.840826 33.385646       Inf 

$error.rate
[1] 0.3349754

$F0.5
        3         4         5         6         7         8 
      NaN 0.3174603 0.6940789 0.6462403 0.7248521 0.8045977 

$F1
        3         4         5         6         7         8 
```

```
      NaN 0.1777778 0.6374622 0.7144970 0.6030769 0.6222222

$F2
        3         4         5         6         7         8 
      NaN 0.1234568 0.5893855 0.7988753 0.5163330 0.5072464 

$FDR
        3         4         5         6         7         8 
      NaN 0.3333333 0.2622378 0.3924528 0.1623932 0.0000000 

$FNR
        3         4         5         6         7         8 
1.0000000 0.8974359 0.4388298 0.1328546 0.5288462 0.5483871 

$FOR
          3           4           5           6           7           8 

0.008567931 0.041617122 0.215968586 0.184538653 0.133819951 0.020910209 


$FPR
          3           4           5           6           7           8 

0.000000000 0.002475248 0.111275964 0.488262911 0.025991792 0.000000000 


$geometric.mean
        3         4         5         6         7         8 
0.0000000 0.3198597 0.7062050 0.6661460 0.6774273 0.6720215 

$Jaccard
         3          4          5          6          7          8 
0.00000000 0.09756098 0.46784922 0.55581128 0.43171806 0.45161290 

$L
        3         4         5         6         7         8 
      NaN 41.435897  5.043050  1.775981 18.127024       Inf 

$lambda
        3         4         5         6         7         8 
1.0000000 0.8996628 0.4937751 0.2596149 0.5429586 0.5483871 

$MCC
        3         4         5         6         7         8 
      NaN 0.2501210 0.4845121 0.4003380 0.5597314 0.6649583 

$MK
        3         4         5         6         7         8 
      NaN 0.6250495 0.5217937 0.4230085 0.7037869 0.9790898 

$NPV
        3         4         5         6         7         8 
0.9914321 0.9583829 0.7840314 0.8154613 0.8661800 0.9790898 

$OP
         3          4          5          6          7          8 
-0.3349754 -0.1485102  0.4391090  0.4072736  0.3170676  0.2872469 

$precision
        3         4         5         6         7         8 
      NaN 0.6666667 0.7377622 0.6075472 0.8376068 1.0000000 

$recall
        3         4         5         6         7         8 
```

```
0.0000000 0.1025641 0.5611702 0.8671454 0.4711538 0.4516129

$specificity
        3         4         5         6         7         8 
1.0000000 0.9975248 0.8887240 0.5117371 0.9740082 1.0000000 

$Youden
        3         4         5         6         7         8 
0.0000000 0.1000889 0.4498942 0.3788825 0.4451621 0.4516129 

> summary(tune.c)

Parameter tuning of ‘randomForest’:

- sampling method: 10-fold cross validation 

- best parameters:
 mtry n.trees
    5     400

- best performance: 0.4351782 

- Detailed performance results:
  mtry n.trees     error dispersion
1  3.6     100 0.4394743 0.02437159
2  4.0     100 0.4394712 0.02752295
3  5.0     100 0.4378911 0.02902205
4  3.6     300 0.4410524 0.02431434
5  4.0     300 0.4435319 0.02822201
6  5.0     300 0.4385734 0.02849866
7  3.6     400 0.4392465 0.02511363
8  4.0     400 0.4376633 0.02535214
9  5.0     400 0.4351782 0.02649971

> mean(pred.tune.c != test2$quality)
[1] 0.2947455
> ml_test(pred.tune.c,test2$quality)
$accuracy
[1] 0.7052545

$balanced.accuracy
        3         4         5         6         7         8 
0.5000000 0.5757535 0.7961405 0.7310958 0.7450878 0.7096774 

$DOR
        3         4         5         6         7         8 
      NaN 77.545455 15.769231  7.802007 25.175258       Inf 

$error.rate
[1] 0.2947455

$F0.5
        3         4         5         6         7         8 
      NaN 0.4225352 0.7226166 0.7005364 0.7043147 0.7831325 

$F1
        3         4         5         6         7         8 
      NaN 0.2553191 0.7354839 0.7338843 0.6288952 0.5909091 

$F2
        3         4         5         6         7         8 
      NaN 0.1829268 0.7488177 0.7705658 0.5680655 0.4744526 

$FDR
```

```
        3         4         5         6         7         8 
      NaN 0.2500000 0.2857143 0.3200613 0.2344828 0.0000000 

$FNR
        3         4         5         6         7         8 
1.0000000 0.8461538 0.2420213 0.2028725 0.4663462 0.5806452 

$FOR
          3           4           5           6           7           8 

0.008083141 0.037246050 0.136842105 0.214015152 0.114792899 0.020833333 


$FPR
          3           4           5           6           7           8 

0.000000000 0.002339181 0.165697674 0.334935897 0.043478261 0.000000000 


$geometric.mean
        3         4         5         6         7         8 
0.0000000 0.3917733 0.7952254 0.7281077 0.7144589 0.6475761 

$Jaccard
        3         4         5         6         7         8 
0.0000000 0.1463415 0.5816327 0.5796345 0.4586777 0.4193548 

$L
        3         4         5         6         7         8 
      NaN 65.769231  4.574468  2.379940 12.274038       Inf 

$lambda
        3         4         5         6         7         8 
1.0000000 0.8481378 0.2900882 0.3050421 0.4875437 0.5806452 

$MCC
        3         4         5         6         7         8 
      NaN 0.3286140 0.5848153 0.4640538 0.5647736 0.6407950 

$MK
        3         4         5         6         7         8 
      NaN 0.7127540 0.5774436 0.4659236 0.6507243 0.9791667 

$NPV
        3         4         5         6         7         8 
0.9919169 0.9627540 0.8631579 0.7859848 0.8852071 0.9791667 

$OP
          3           4           5           6           7           8 

-0.29474548 -0.02753711  0.65732102  0.61493573  0.42148400  0.29616361 


$precision
        3         4         5         6         7         8 
      NaN 0.7500000 0.7142857 0.6799387 0.7655172 1.0000000 

$recall
        3         4         5         6         7         8 
0.0000000 0.1538462 0.7579787 0.7971275 0.5336538 0.4193548 

$specificity
        3         4         5         6         7         8 
1.0000000 0.9976608 0.8343023 0.6650641 0.9565217 1.0000000 
```

```
$Youden
        3         4         5         6         7         8 
0.0000000 0.1515070 0.5922810 0.4621916 0.4901756 0.4193548 

> summary(tune1.c)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation 

- best parameters:
 cost
    3

- best performance: 0.4620655 

- Detailed performance results:
  cost     error dispersion
1    3 0.4620655 0.02825759
2    4 0.4620655 0.02825759
3    5 0.4622912 0.02853186
4    6 0.4622912 0.02853186
5    7 0.4625170 0.02882120
6    8 0.4625170 0.02882120
7    9 0.4625170 0.02882120

> mean(pred.tune1.c != test2$quality)
[1] 0.454844
> ml_test(pred.tune1.c,test2$quality)
$accuracy
[1] 0.545156

$balanced.accuracy
        3         4         5         6         7         8 
0.5000000 0.5000000 0.6750524 0.5741118 0.5000000 0.5000000 

$DOR
       3        4        5        6        7        8 
     NaN      NaN 4.337227 1.994745      NaN      NaN 

$error.rate
[1] 0.454844

$F0.5
        3         4         5         6         7         8 
      NaN       NaN 0.5899420 0.5619481       NaN       NaN 

$F1
        3         4         5         6         7         8 
      NaN       NaN 0.6107635 0.6213018       NaN       NaN 

$F2
        3         4         5         6         7         8 
      NaN       NaN 0.6331085 0.6946742       NaN       NaN 

$FDR
        3         4         5         6         7         8 
      NaN       NaN 0.4231678 0.4716981       NaN       NaN 

$FNR
        3         4         5         6         7         8 
1.0000000 1.0000000 0.3510638 0.2459605 1.0000000 1.0000000 
```

```
$FOR
         3          4          5          6          7          8 
0.01043219 0.05547653 0.23913043 0.35958005 0.23853211 0.04460432 

$FPR
        3         4         5         6         7         8 
0.0000000 0.0000000 0.2988314 0.6058158 0.0000000 0.0000000 

$geometric.mean
        3         4         5         6         7         8 
0.0000000 0.0000000 0.6745470 0.5451884 0.0000000 0.0000000 

$Jaccard
        3         4         5         6         7         8 
0.0000000 0.0000000 0.4396396 0.4506438 0.0000000 0.0000000 

$L
       3        4        5        6        7        8 
     NaN      NaN 2.171580 1.244668      NaN      NaN 

$lambda
        3         4         5         6         7         8 
1.0000000 1.0000000 0.5006839 0.6239736 1.0000000 1.0000000 

$MCC
        3         4         5         6         7         8 
      NaN       NaN 0.3438473 0.1581410       NaN       NaN 

$MK
        3         4         5         6         7         8 
      NaN       NaN 0.3377017 0.1687218       NaN       NaN 

$NPV
        3         4         5         6         7         8 
0.9895678 0.9445235 0.7608696 0.6404199 0.7614679 0.9553957 

$OP
         3          4          5          6          7          8 
-0.4548440 -0.4548440  0.5064683  0.2317542 -0.4548440 -0.4548440 

$precision
        3         4         5         6         7         8 
      NaN       NaN 0.5768322 0.5283019       NaN       NaN 

$recall
        3         4         5         6         7         8 
0.0000000 0.0000000 0.6489362 0.7540395 0.0000000 0.0000000 

$specificity
        3         4         5         6         7         8 
1.0000000 1.0000000 0.7011686 0.3941842 1.0000000 1.0000000 

$Youden
        3         4         5         6         7         8 
0.0000000 0.0000000 0.3501048 0.1482237 0.0000000 0.0000000 

> summary(tune2.c)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation 

- best parameters:
 cost degree
```

```
    4      3

- best performance: 0.4469342 

- Detailed performance results:
   cost degree     error dispersion
1     3      2 0.4496399 0.02087887
2     4      2 0.4476099 0.02240681
3     5      2 0.4480603 0.02154069
4     3      3 0.4471620 0.02670821
5     4      3 0.4469342 0.02450999
6     5      3 0.4471635 0.02646017
7     3      4 0.4625103 0.02675210
8     4      4 0.4629633 0.02818077
9     5      4 0.4611565 0.02535700
10    3      5 0.4742551 0.02670802
11    4      5 0.4715422 0.02492235
12    5      5 0.4740268 0.02283545

> mean(pred.tune2.c != test2$quality)
[1] 0.4244663
> ml_test(pred.tune2.c,test2$quality)
$accuracy
[1] 0.5755337

$balanced.accuracy
        3         4         5         6         7         8 
0.4992877 0.5113960 0.6961120 0.6063529 0.5811229 0.5000000 

$DOR
       3        4        5        6        7        8 
0.000000 9.210526 5.456115 2.579007 4.536284      NaN 

$error.rate
[1] 0.4244663

$F0.5
         3          4          5          6          7          8 
       NaN 0.09803922 0.62237395 0.58948261 0.41366906        NaN 

$F1
         3          4          5          6          7          8 
       NaN 0.04761905 0.62532982 0.64055300 0.31186441        NaN 

$F2
         3          4          5          6          7          8 
       NaN 0.03144654 0.62831389 0.70131181 0.25027203        NaN 

$FDR
        3         4         5         6         7         8 
1.0000000 0.6666667 0.3795812 0.4402685 0.4712644       NaN 

$FNR
        3         4         5         6         7         8 
1.0000000 0.9743590 0.3696809 0.2513465 0.7788462 1.0000000 

$FOR
          3           4           5           6           7           8 

0.009887006 0.051490515 0.230514096 0.330188679 0.198286414 0.042349727 


$FPR
```

```
          3           4           5           6           7           8 
0.001424501 0.002849003 0.238095238 0.535947712 0.058908046 0.000000000 

$geometric.mean
        3         4         5         6         7         8 
0.0000000 0.1598999 0.6929958 0.5894187 0.4562084 0.0000000 

$Jaccard
         3          4          5          6          7          8 
0.00000000 0.02439024 0.45489443 0.47118644 0.18473896 0.00000000 

$L
       3        4        5        6        7        8 
0.000000 9.000000 2.647340 1.396878 3.754221      NaN 

$lambda
        3         4         5         6         7         8 
1.0014265 0.9771429 0.4852061 0.5416340 0.8275984 1.0000000 

$MCC
           3            4            5            6            7 
     8 
-0.003752873  0.080148412  0.391062612  0.220964015  0.231546967 
   NaN 

$MK
           3            4            5            6            7 
     8 
-0.009887006  0.281842818  0.389904752  0.229542864  0.330449218 
   NaN 

$NPV
        3         4         5         6         7         8 
0.9901130 0.9485095 0.7694859 0.6698113 0.8017136 0.9576503 

$OP
          3           4           5           6           7           8 
-0.42446634 -0.37432706  0.48101897  0.34085084 -0.04390339 -0.42446634 

$precision
        3         4         5         6         7         8 
0.0000000 0.3333333 0.6204188 0.5597315 0.5287356       NaN 

$recall
         3          4          5          6          7          8 
0.00000000 0.02564103 0.63031915 0.74865350 0.22115385 0.00000000 

$specificity
        3         4         5         6         7         8 
0.9985755 0.9971510 0.7619048 0.4640523 0.9410920 1.0000000 

$Youden
           3            4            5            6            7 
     8 
-0.001424501  0.022792023  0.392223911  0.212705788  0.162245800  0.000
000000 

> summary(tune3.c)

Parameter tuning of ‘svm’:
```

```
- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
    3   0.1

- best performance: 0.4408358

- Detailed performance results:
   cost gamma     error dispersion
1     3  2.00 0.4866884 0.02141370
2     4  2.00 0.4884947 0.02205311
3     5  2.00 0.4882685 0.02049033
4     3 10.00 0.5381837 0.02413940
5     4 10.00 0.5381837 0.02413940
6     5 10.00 0.5381837 0.02413940
7     3  0.01 0.4586764 0.01960067
8     4  0.01 0.4570938 0.01746337
9     5  0.01 0.4570963 0.01602156
10    3  0.10 0.4408358 0.02626395
11    4  0.10 0.4421882 0.02221451
12    5  0.10 0.4460272 0.02245305

> mean(pred.tune3.c != test2$quality)
[1] 0.410509
> ml_test(pred.tune3.c,test2$quality)
$accuracy
[1] 0.589491

$balanced.accuracy
        3         4         5         6         7         8 
0.5000000 0.5377632 0.7108323 0.6170568 0.6047719 0.5000000 

$DOR
        3         4         5         6         7         8 
      NaN 59.583333  6.131176  2.756036  6.652319       NaN 

$error.rate
[1] 0.410509

$F0.5
        3         4         5         6         7         8 
      NaN 0.2727273 0.6287276 0.5963169 0.4787234       NaN 

$F1
        3         4         5         6         7         8 
      NaN 0.1395349 0.6445860 0.6410055 0.3636364       NaN 

$F2
        3         4         5         6         7         8 
      NaN 0.0937500 0.6612650 0.6929348 0.2931596       NaN 

$FDR
        3         4         5         6         7         8 
      NaN 0.2500000 0.3814181 0.4301676 0.3932584       NaN 

$FNR
        3         4         5         6         7         8 
1.0000000 0.9230769 0.3271277 0.2675045 0.7403846 1.0000000 

$FOR
          3           4           5           6           7           8 
```

```
0.009655172 0.047936085 0.209183673 0.324618736 0.188264059 0.041388518

$FPR
          3           4           5           6           7           8 
0.000000000 0.001396648 0.251207729 0.498381877 0.050071531 0.000000000 

$geometric.mean
        3         4         5         6         7         8 
0.0000000 0.2771564 0.7098180 0.6061625 0.4966045 0.0000000 

$Jaccard
        3         4         5         6         7         8 
0.0000000 0.0750000 0.4755639 0.4716763 0.2222222 0.0000000 

$L
        3         4         5         6         7         8 
      NaN 55.076923  2.678550  1.469747  5.184890       NaN 

$lambda
        3         4         5         6         7         8 
1.0000000 0.9243679 0.4368737 0.5332831 0.7794109 1.0000000 

$MCC
        3         4         5         6         7         8 
      NaN 0.2302702 0.4154862 0.2395994 0.2961239       NaN 

$MK
        3         4         5         6         7         8 
      NaN 0.7020639 0.4093982 0.2452137 0.4184775       NaN 

$NPV
        3         4         5         6         7         8 
0.9903448 0.9520639 0.7908163 0.6753813 0.8117359 0.9586115 

$OP
          3           4           5           6           7           8 
-0.41050903 -0.26746638  0.53608883  0.40241145  0.01876914 -0.41050903 

$precision
        3         4         5         6         7         8 
      NaN 0.7500000 0.6185819 0.5698324 0.6067416       NaN 

$recall
         3          4          5          6          7          8 
0.00000000 0.07692308 0.67287234 0.73249551 0.25961538 0.00000000 

$specificity
        3         4         5         6         7         8 
1.0000000 0.9986034 0.7487923 0.5016181 0.9499285 1.0000000 

$Youden
         3          4          5          6          7          8 
0.00000000 0.07552643 0.42166461 0.23411363 0.20954385 0.00000000 


print(gbm.caret1)
Stochastic Gradient Boosting
```

```
4428 samples
  12 predictor
   6 classes: '3', '4', '5', '6', '7', '8'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 3985, 3984, 3985, 3986, 3987, 3986, ...
Resampling results across tuning parameters:

  shrinkage  interaction.depth  n.trees  Accuracy   Kappa
  0.001      1                  100      0.5101724  0.1940931
  0.001      1                  200      0.5113026  0.1944809
  0.001      1                  300      0.5144598  0.1990030
  0.001      2                  100      0.5232696  0.1978072
  0.001      2                  200      0.5241679  0.2002518
  0.001      2                  300      0.5232639  0.1998567
  0.001      3                  100      0.5239468  0.2043641
  0.001      3                  200      0.5264278  0.2105169
  0.001      3                  300      0.5316284  0.2226277
  0.001      4                  100      0.5291361  0.2231207
  0.001      4                  200      0.5361385  0.2362031
  0.001      4                  300      0.5386241  0.2416846
  0.010      1                  100      0.5194270  0.2097906
  0.010      1                  200      0.5295957  0.2305528
  0.010      1                  300      0.5359148  0.2448222
  0.010      2                  100      0.5332065  0.2269786
  0.010      2                  200      0.5379418  0.2441742
  0.010      2                  300      0.5408686  0.2545610
  0.010      3                  100      0.5417895  0.2478165
  0.010      3                  200      0.5456121  0.2601209
  0.010      3                  300      0.5483271  0.2690110
  0.010      4                  100      0.5433563  0.2550539
  0.010      4                  200      0.5478695  0.2673759
  0.010      4                  300      0.5487709  0.2734662

Tuning parameter 'n.minobsinnode' was held constant at a value of 10
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were n.trees = 300, interaction.dep
th = 4, shrinkage = 0.01 and n.minobsinnode = 10.
> predict_gbm1 <- predict(gbm.caret1,newdata = test2)
> mean(predict_gbm1 !=test2$quality)
[1] 0.4334975
> table(predict_gbm1 ,test2$quality)

predict_gbm1   3   4   5   6   7   8
           3   0   0   0   0   0   0
           4   0   1   2   1   0   0
           5   5  21 242 130  10   1
           6   2  17 131 401 150  23
           7   0   0   1  25  45   6
           8   0   0   0   0   3   1
> ml_test(test2$quality, predict_gbm1 )
$accuracy
[1] 0.5665025

$balanced.accuracy
        3         4         5         6         7         8
      NaN 0.5988652 0.6807232 0.6016528 0.6913415 0.6041377

$DOR
       3        4        5        6        7        8
     NaN 6.043860 4.844758 2.299933 5.564609 7.655556

$error.rate
```

```
[1] 0.4334975

$F0.5
        3         4         5         6         7         8 
      NaN 0.0312500 0.6325144 0.6792005 0.2475248 0.0390625 

$F1
         3          4          5          6          7          8 
       NaN 0.04651163 0.61656051 0.62607338 0.31578947 0.05714286 

$F2
         3          4          5          6          7          8 
       NaN 0.09090909 0.60139165 0.58065450 0.43604651 0.10638298 

$FDR
        3         4         5         6         7         8 
1.0000000 0.9743590 0.3563830 0.2800718 0.7836538 0.9677419 

$FNR
        3         4         5         6         7         8 
      NaN 0.7500000 0.4083130 0.4461326 0.4155844 0.7500000 

$FOR
         3          4          5          6          7          8 
0.00000000 0.00433526 0.27154472 0.52777778 0.04726736 0.00433526 

$FPR
         3          4          5          6          7          8 
0.01004304 0.05226960 0.23024055 0.35056180 0.20173267 0.04172462 

$geometric.mean
        3         4         5         6         7         8 
      NaN 0.4867572 0.6748753 0.5997522 0.6830226 0.4894577 

$Jaccard
         3          4          5          6          7          8 
0.00000000 0.02380952 0.44567219 0.45568182 0.18750000 0.02941176 

$L
       3        4        5        6        7        8 
     NaN 4.782895 2.569865 1.579942 2.896980 5.991667 

$lambda
        3         4         5         6         7         8 
      NaN 0.7913643 0.5304423 0.6869516 0.5206081 0.7826560 

$MCC
         3          4          5          6          7          8 
       NaN 0.06490607 0.36672092 0.19764932 0.25436896 0.07626030 

$MK
         3          4          5          6          7          8 
0.00000000 0.02130577 0.37207231 0.19215041 0.16907880 0.02792280 

$NPV
        3         4         5         6         7         8 
1.0000000 0.9956647 0.7284553 0.4722222 0.9527326 0.9956647 

$OP
          3           4           5           6           7           8 

        NaN -0.01604132  0.43570598  0.48707892  0.41183812 -0.01968459
```

```
$precision
         3          4          5          6          7          8 
0.00000000 0.02564103 0.64361702 0.71992819 0.21634615 0.03225806 

$recall
        3         4         5         6         7         8 
      NaN 0.2500000 0.5916870 0.5538674 0.5844156 0.2500000 

$specificity
        3         4         5         6         7         8 
0.9899570 0.9477304 0.7697595 0.6494382 0.7982673 0.9582754 

$Youden
        3         4         5         6         7         8 
      NaN 0.1977304 0.3614465 0.2033056 0.3826829 0.2082754 
```

Explanation- In order to make this a classification problem, one of the levels of the wine quality had to be removed since train and holdout data sets have different levels. By following this approach, quality level 9 is lost. The disadvantage of using the misclassification rate is that it shows the overall misclassification rate not for different factors that are present in the data (rather than giving the misclassification rate of each factors present in the data set). By using this metric, the number of misclassifications made by each factor can't be seen which will puts us at a great disadvantage when doing multiclass classification.

An alternative metric that can be used instead of mis-classification rate is F-1 score. F1 score we can give us a more realistic measure of our classifier's performance. Moreover, we can avoid to be fooled by the arithmetic mean between a very poor PRECISION and very high RECALL, which can be obtained simply by classifying all of the documents as positive. High precision but lower recall, gives you an extremely accurate, but it then misses a large number of instances that are difficult to classify. The greater the F1 Score, the better is the performance of our model. For this classification problem, F-1 score is calculated for every factor in the response to check the model performance for every factor.

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Precision: It is the number of correct positive results divided by the number of positive results predicted by the classifier.

Recall: It is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive).

$$\text{F1} = 2 \text{ x } \frac{Precision * Recall}{Precision + Recall}$$

In this classification problem, ml_test () is used to get the F-1 scores of the three models used to do the classification. The misclassification rates of Random Forest, SVM and Boosting after hyper parameter tuning are 0.293,0.336,0.4334. The F-scores of the models are tabulated below: -

Quality Factors	F-1 Score
3	-
4	0.25531
5	0.7354
6	0.73388
7	0.6288
8	0.8909

Quality Factors	F-1 Score
3	-
4	0.177
5	0.63746
6	0.71441
7	0.60307
8	0.622

Quality Factors	F-1 Score
3	-
4	0.090
5	0.6013
6	0.5806
7	0.436046
8	0.10638298

It can be seen the F-1 score is calculated for every factor of the response. The F-1 score for the factor 3 is unknow because the precision for that factor is zero, this tells us that the model prediction for factor 3 is not good. I think the reason for low precision for 3 and 4 is because of the smaller number of observations of level 3 and 4 presents in training data set. From the above tables, it can be observed that F-1 scores for factors 5, 6, 7 and 8 are close to one, telling us that the model does a good job in predicting those values. From the above table, it can be concluded that random forest performs better than other classifiers based on the F-1 score.

5. Question 5

   Write an "executive summary" summarizing results of regression and classification models with your interpretation. Summarize results on holdout data. What are the features affecting wine quality? Are they different for red and white wines?

   Code-

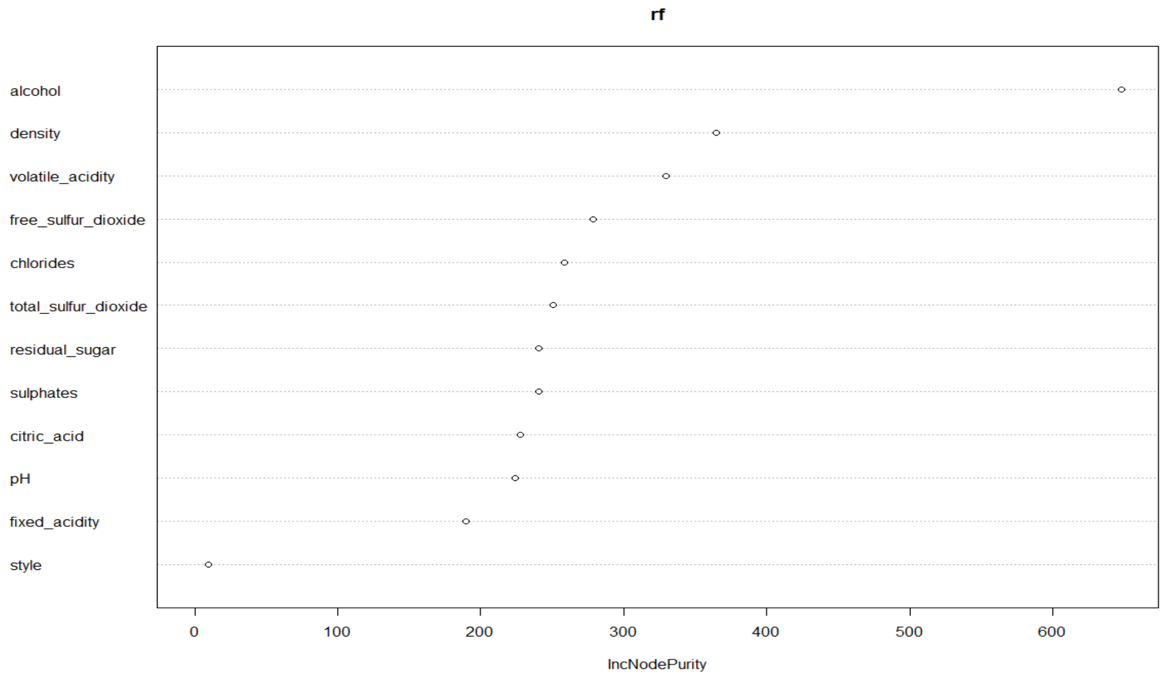
   importance(rf)

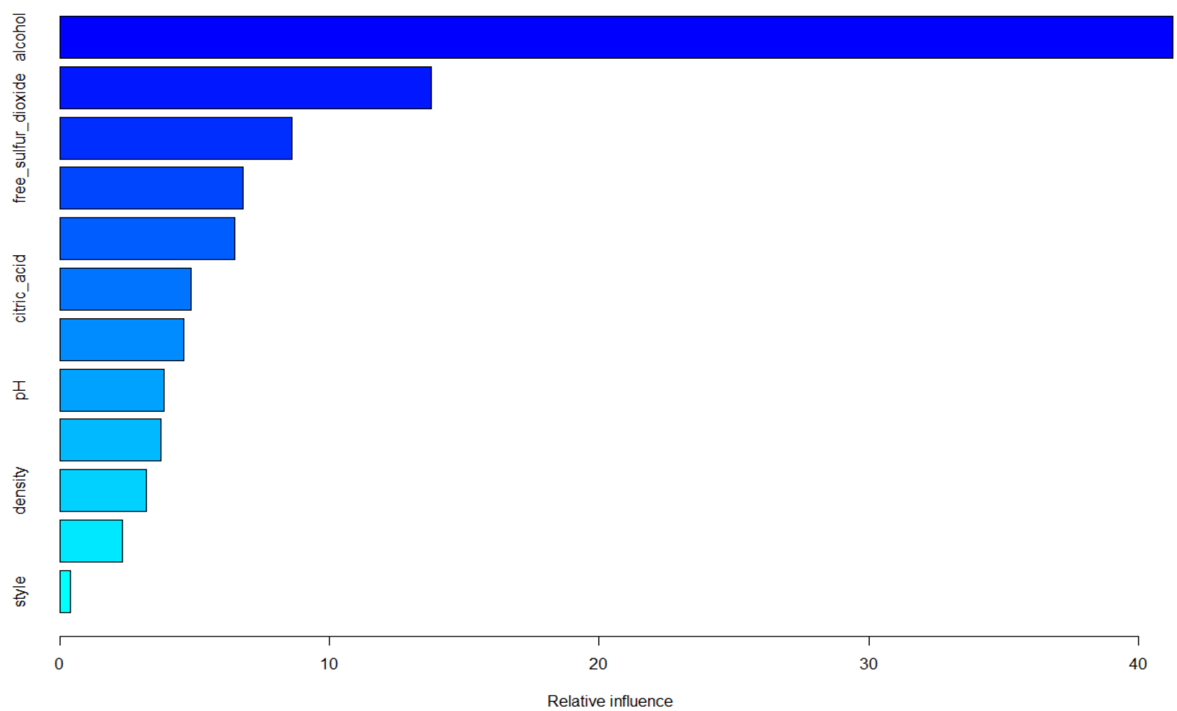
   varImpPlot(rf)

   summary(boosting)

   Output-

```
importance(rf)
                     IncNodePurity
fixed_acidity           189.850140
volatile_acidity        329.284608
citric_acid             227.716462
residual_sugar          240.572610
chlorides               258.371612
free_sulfur_dioxide     278.258763
total_sulfur_dioxide    250.579400
density                 365.003137
pH                      223.735186
sulphates               240.366319
alcohol                 647.968658
style                     9.896365
```


rf
alcohol
density
volatile_acidity
free_sulfur_dioxide
chlorides
total_sulfur_dioxide
residual_sugar
sulphates
citric_acid
pH
fixed_acidity
style
0
100
200
300
400
500
600
IncNodePurity


alcohol
free_sulfur_dioxide
citric_acid
pH
density
style
0
10
20
30
40
Relative influence

Explanation-

Wine Data set is used as the train data set to fit three models and Wine Holdout data set is used as the test data to find the test error. In this project, both classification and regression models are used to find the quality of the wine.
When regressions models are used the test, accuracy are found out to be of 0.65, 0.52,0 0.53. The best regression model is Random forest since it has a better test accuracy when compared with other models. And, when the predictions are done on wine quality when it is treated as a classification problem, the test errors are found out to be of 0.1083, 0.166, 0.17. Random forest had a better accuracy in both the cases. Using the other metric tells us the same.

Features affecting the wine quality can be found out using the above plots. It is clearly visible that alcohol has the highest influence on the wine quality. Every feature has an effect on the wine quality except for the style of the wine. This is understandable since the type of wine does not determine the quality.

But from out of all the features, from the above plots one can infer that, alcohol, density, volatile_acidity, free_sulfur_dioxide, chlorides, total_sulfur_dioxide, residual_sugar are the features that influence the quality of wine the most. The feature affecting wine quality are not different for red and white wine because the type of wine has the least influence on the quality of wine. In conclusion every feature has an affect over the quality of the wine but above set of features have the most influence on the wine quality.