logistic_regression

Rishika Cherivirala

2025-04-25

```
wine_df <- read.csv("data/wine-quality-white-and-red (1).csv")
head(wine_df)
```

```
##    type fixed.acidity volatile.acidity citric.acid residual.sugar chlorides
## 1 white           7.0             0.27        0.36           20.7     0.045
## 2 white           6.3             0.30        0.34            1.6     0.049
## 3 white           8.1             0.28        0.40            6.9     0.050
## 4 white           7.2             0.23        0.32            8.5     0.058
## 5 white           7.2             0.23        0.32            8.5     0.058
## 6 white           8.1             0.28        0.40            6.9     0.050
##   free.sulfur.dioxide total.sulfur.dioxide density   pH sulphates alcohol
## 1                  45                  170  1.0010 3.00      0.45     8.8
## 2                  14                  132  0.9940 3.30      0.49     9.5
## 3                  30                   97  0.9951 3.26      0.44    10.1
## 4                  47                  186  0.9956 3.19      0.40     9.9
## 5                  47                  186  0.9956 3.19      0.40     9.9
## 6                  30                   97  0.9951 3.26      0.44    10.1
##   quality
## 1       6
## 2       6
## 3       6
## 4       6
## 5       6
## 6       6
```

```
quality_counts <- table(wine_df$quality)
print(quality_counts)
```

```
##
##    3    4    5    6    7    8    9
##   30  216 2138 2836 1079  193    5
```

Multinomial Logistic Regression

```
wine_df$quality <- as.factor(wine_df$quality)

set.seed(1)
```

```r
index <- sample(1:nrow(wine_df), 0.7 * nrow(wine_df))
train <- wine_df[index, ]
test <- wine_df[-index, ]

multi_model <- multinom(quality ~ ., data = train)
```

```
## # weights:  98 (78 variable)
## initial  value 8848.053448
## iter  10 value 6112.967244
## iter  20 value 5795.162701
## iter  30 value 5483.743022
## iter  40 value 4942.177481
## iter  50 value 4848.170190
## iter  60 value 4825.676791
## iter  70 value 4814.244712
## iter  80 value 4812.501302
## iter  90 value 4811.705139
## iter 100 value 4810.173610
## final  value 4810.173610
## stopped after 100 iterations
```

```r
predictions <- predict(multi_model, test)

confusionMatrix(predictions, test$quality)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8   9
##          3   1   0   3   0   0   0   0
##          4   0   5   4   0   0   0   0
##          5   2  39 393 190  24   6   0
##          6   4  22 232 599 228  35   1
##          7   0   1   3  71  65  18   2
##          8   0   0   0   1   0   0   0
##          9   1   0   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.5451
##                  95% CI : (0.5227, 0.5674)
##     No Information Rate : 0.4415
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.2704
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7  Class: 8
## Sensitivity          0.1250000 0.074627   0.6189   0.6957  0.20505 0.0000000
```

```
## Specificity          0.9984552 0.997876   0.8015   0.5207  0.94182 0.9994712
## Pos Pred Value       0.2500000 0.555556   0.6009   0.5343  0.40625 0.0000000
## Neg Pred Value       0.9964029 0.968058   0.8133   0.6840  0.85922 0.9697281
## Prevalence           0.0041026 0.034359   0.3256   0.4415  0.16256 0.0302564
## Detection Rate       0.0005128 0.002564   0.2015   0.3072  0.03333 0.0000000
## Detection Prevalence 0.0020513 0.004615   0.3354   0.5749  0.08205 0.0005128
## Balanced Accuracy    0.5617276 0.536251   0.7102   0.6082  0.57344 0.4997356
##                       Class: 9
## Sensitivity          0.0000000
## Specificity          0.9994864
## Pos Pred Value       0.0000000
## Neg Pred Value       0.9984607
## Prevalence           0.0015385
## Detection Rate       0.0000000
## Detection Prevalence 0.0005128
## Balanced Accuracy    0.4997432
```

ROC Curve

```
predicted_probs_multi <- predict(multi_model, test, type = "probs")

true_labels <- test$quality

roc_list <- lapply(levels(true_labels), function(class_label) {
  binary_labels <- ifelse(true_labels == class_label, 1, 0)

  roc(binary_labels, predicted_probs_multi[, class_label])
})
```

```
## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

## Setting levels: control = 0, case = 1
```

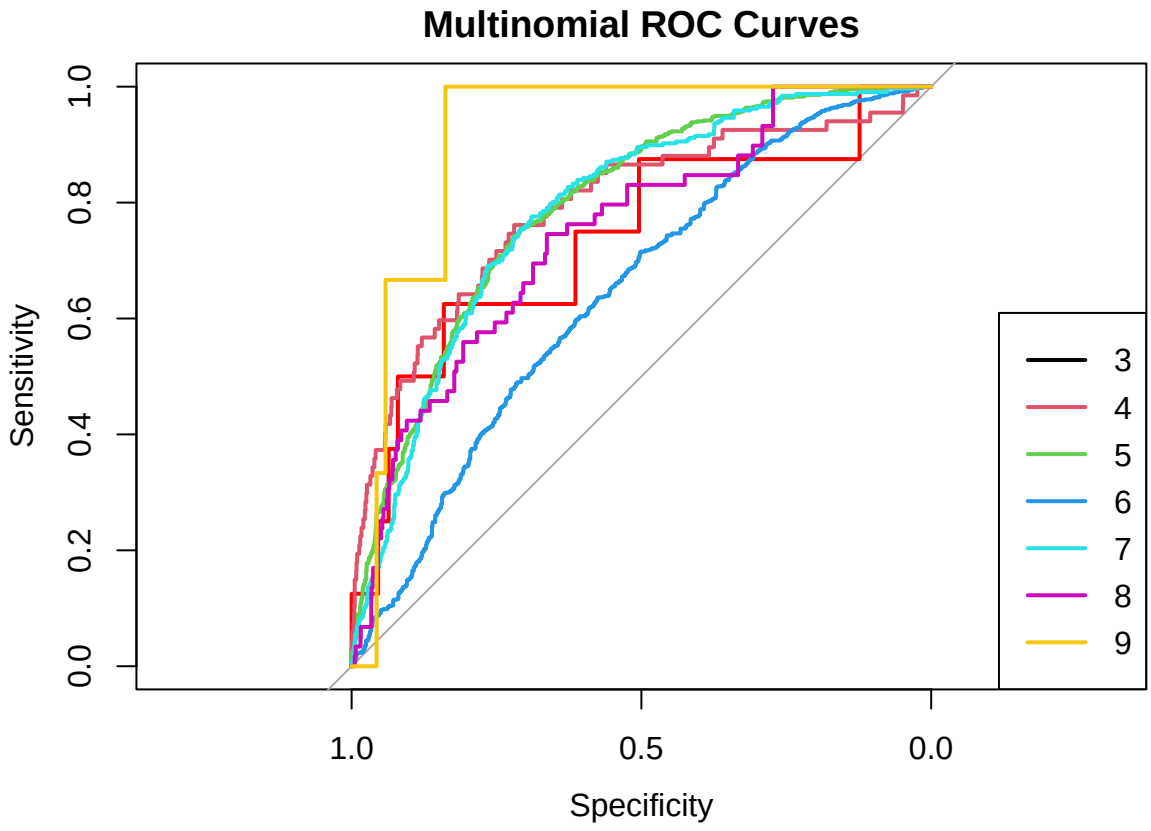
```
## Setting direction: controls < cases

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```

```
plot(roc_list[[1]], col = "red", main = "Multinomial ROC Curves", lwd = 2)
for (i in 2:length(roc_list)) {
  lines(roc_list[[i]], col = i, lwd = 2)
}

legend("bottomright", legend = levels(true_labels), col = 1:length(roc_list), lwd = 2)
```




```
sapply(roc_list, auc)
```

```
## [1] 0.7366117 0.7912033 0.7940301 0.6434688 0.7860067 0.7477525 0.9121726
```

##Cross Validation

```
wine_df$quality <- as.factor(wine_df$quality)

control <- trainControl(method = "cv", number = 10)

set.seed(1)
multi_model_cv <- train(quality ~ ., data = wine_df, method = "multinom", trControl = control)
```

```
## # weights:  98 (78 variable)
## initial  value 11379.682552
## iter  10 value 7789.984548
## iter  20 value 7248.872632
## iter  30 value 6895.252501
## iter  40 value 6349.905775
## iter  50 value 6233.115941
## iter  60 value 6211.114660
## iter  70 value 6201.118348
## iter  80 value 6198.760848
## iter  90 value 6197.357098
## iter 100 value 6193.240337
## final  value 6193.240337
## stopped after 100 iterations
## # weights:  98 (78 variable)
## initial  value 11379.682552
## iter  10 value 7789.989830
## iter  20 value 7249.198748
## iter  30 value 6899.742524
## iter  40 value 6375.066443
## iter  50 value 6267.744821
## iter  60 value 6260.194216
## iter  70 value 6259.728145
## final  value 6259.727790
## converged
## # weights:  98 (78 variable)
## initial  value 11379.682552
## iter  10 value 7789.984553
## iter  20 value 7248.872958
## iter  30 value 6895.257066
## iter  40 value 6349.936986
## iter  50 value 6233.261836
## iter  60 value 6211.200732
## iter  70 value 6201.605819
## iter  80 value 6199.346314
## iter  90 value 6198.168534
## iter 100 value 6195.603970
## final  value 6195.603970
## stopped after 100 iterations
## # weights:  98 (78 variable)
## initial  value 11377.736642
## iter  10 value 7817.729333
## iter  20 value 7397.143388
## iter  30 value 6950.100847
## iter  40 value 6322.235893
## iter  50 value 6231.510086
## iter  60 value 6210.016631
## iter  70 value 6200.664646
## iter  80 value 6197.819510
## iter  90 value 6195.793615
## iter 100 value 6192.662398
## final  value 6192.662398
## stopped after 100 iterations
## # weights:  98 (78 variable)
```

```
## initial  value 11377.736642
## iter  10 value 7817.734021
## iter  20 value 7397.433219
## iter  30 value 6954.157628
## iter  40 value 6345.731202
## iter  50 value 6272.771653
## iter  60 value 6263.366862
## iter  70 value 6263.226602
## final  value 6263.226098
## converged
## # weights:  98 (78 variable)
## initial  value 11377.736642
## iter  10 value 7817.729337
## iter  20 value 7397.143678
## iter  30 value 6950.104942
## iter  40 value 6322.262036
## iter  50 value 6231.569813
## iter  60 value 6210.153249
## iter  70 value 6201.113218
## iter  80 value 6198.521275
## iter  90 value 6196.816373
## iter 100 value 6194.793923
## final  value 6194.793923
## stopped after 100 iterations
## # weights:  98 (78 variable)
## initial  value 11375.790731
## iter  10 value 7899.098477
## iter  20 value 7233.077353
## iter  30 value 6780.977481
## iter  40 value 6380.216306
## iter  50 value 6247.868176
## iter  60 value 6228.043457
## iter  70 value 6220.456516
## iter  80 value 6217.750250
## iter  90 value 6216.529040
## iter 100 value 6210.776119
## final  value 6210.776119
## stopped after 100 iterations
## # weights:  98 (78 variable)
## initial  value 11375.790731
## iter  10 value 7899.106420
## iter  20 value 7233.493995
## iter  30 value 6761.103288
## iter  40 value 6367.868840
## iter  50 value 6283.892568
## iter  60 value 6277.304670
## iter  70 value 6276.869923
## iter  70 value 6276.869890
## iter  70 value 6276.869888
## final  value 6276.869888
## converged
## # weights:  98 (78 variable)
## initial  value 11375.790731
## iter  10 value 7899.098485
```

```
## iter  20 value 7233.077770
## iter  30 value 6780.944035
## iter  40 value 6380.189942
## iter  50 value 6247.914992
## iter  60 value 6228.152700
## iter  70 value 6220.779338
## iter  80 value 6218.245602
## iter  90 value 6217.211534
## iter 100 value 6213.854243
## final  value 6213.854243
## stopped after 100 iterations
## # weights:  98 (78 variable)
## initial  value 11375.790731
## iter  10 value 7775.957782
## iter  20 value 7179.013683
## iter  30 value 6864.324111
## iter  40 value 6348.011645
## iter  50 value 6227.188505
## iter  60 value 6205.289610
## iter  70 value 6196.514308
## iter  80 value 6194.271830
## iter  90 value 6193.275286
## iter 100 value 6190.626730
## final  value 6190.626730
## stopped after 100 iterations
## # weights:  98 (78 variable)
## initial  value 11375.790731
## iter  10 value 7775.962810
## iter  20 value 7179.652566
## iter  30 value 6862.835256
## iter  40 value 6375.954170
## iter  50 value 6260.385345
## iter  60 value 6254.268314
## iter  70 value 6254.101227
## final  value 6254.098899
## converged
## # weights:  98 (78 variable)
## initial  value 11375.790731
## iter  10 value 7775.957787
## iter  20 value 7179.014322
## iter  30 value 6864.320019
## iter  40 value 6348.145497
## iter  50 value 6227.224227
## iter  60 value 6205.411808
## iter  70 value 6196.902552
## iter  80 value 6194.814920
## iter  90 value 6193.961520
## iter 100 value 6192.161393
## final  value 6192.161393
## stopped after 100 iterations
## # weights:  98 (78 variable)
## initial  value 11379.682552
## iter  10 value 7836.698538
## iter  20 value 7249.291284
```

```
## iter  30 value 6914.992392
## iter  40 value 6366.554112
## iter  50 value 6235.271376
## iter  60 value 6215.678099
## iter  70 value 6206.621367
## iter  80 value 6204.270447
## iter  90 value 6202.802569
## iter 100 value 6198.276737
## final  value 6198.276737
## stopped after 100 iterations
## # weights:  98 (78 variable)
## initial  value 11379.682552
## iter  10 value 7836.703008
## iter  20 value 7249.787843
## iter  30 value 6917.371043
## iter  40 value 6372.536210
## iter  50 value 6274.082317
## iter  60 value 6268.462571
## iter  70 value 6268.286218
## iter  70 value 6268.286182
## iter  70 value 6268.286181
## final  value 6268.286181
## converged
## # weights:  98 (78 variable)
## initial  value 11379.682552
## iter  10 value 7836.698543
## iter  20 value 7249.291781
## iter  30 value 6914.994759
## iter  40 value 6366.818497
## iter  50 value 6234.797479
## iter  60 value 6215.771518
## iter  70 value 6206.972357
## iter  80 value 6204.872022
## iter  90 value 6203.619736
## iter 100 value 6200.683978
## final  value 6200.683978
## stopped after 100 iterations
## # weights:  98 (78 variable)
## initial  value 11377.736642
## iter  10 value 7759.227667
## iter  20 value 7194.851477
## iter  30 value 6776.982915
## iter  40 value 6328.805520
## iter  50 value 6231.509612
## iter  60 value 6209.489811
## iter  70 value 6200.648608
## iter  80 value 6197.667187
## iter  90 value 6195.829704
## iter 100 value 6193.355369
## final  value 6193.355369
## stopped after 100 iterations
## # weights:  98 (78 variable)
## initial  value 11377.736642
## iter  10 value 7759.232502
```

```
## iter  20 value 7195.097158
## iter  30 value 6779.863967
## iter  40 value 6343.713913
## iter  50 value 6268.541475
## iter  60 value 6260.997612
## iter  70 value 6260.637369
## final  value 6260.633272
## converged
## # weights:  98 (78 variable)
## initial  value 11377.736642
## iter  10 value 7759.227672
## iter  20 value 7194.851722
## iter  30 value 6776.985815
## iter  40 value 6328.820114
## iter  50 value 6231.577714
## iter  60 value 6209.614010
## iter  70 value 6201.024321
## iter  80 value 6198.285488
## iter  90 value 6196.684047
## iter 100 value 6195.010575
## final  value 6195.010575
## stopped after 100 iterations
## # weights:  98 (78 variable)
## initial  value 11379.682552
## iter  10 value 7838.513037
## iter  20 value 7256.343888
## iter  30 value 6855.581734
## iter  40 value 6319.402238
## iter  50 value 6218.773319
## iter  60 value 6197.181161
## iter  70 value 6187.014927
## iter  80 value 6184.669824
## iter  90 value 6183.011529
## iter 100 value 6177.631330
## final  value 6177.631330
## stopped after 100 iterations
## # weights:  98 (78 variable)
## initial  value 11379.682552
## iter  10 value 7838.518552
## iter  20 value 7256.711601
## iter  30 value 6859.483825
## iter  40 value 6341.838449
## iter  50 value 6256.260030
## iter  60 value 6249.419896
## iter  70 value 6248.801891
## final  value 6248.798010
## converged
## # weights:  98 (78 variable)
## initial  value 11379.682552
## iter  10 value 7838.513043
## iter  20 value 7256.344256
## iter  30 value 6855.585676
## iter  40 value 6319.425962
## iter  50 value 6218.826944
```

```
## iter  60 value 6197.304268
## iter  70 value 6187.456034
## iter  80 value 6185.281022
## iter  90 value 6183.891293
## iter 100 value 6180.530269
## final  value 6180.530269
## stopped after 100 iterations
## # weights:  98 (78 variable)
## initial  value 11377.736642
## iter  10 value 7765.965221
## iter  20 value 7177.777466
## iter  30 value 6796.732000
## iter  40 value 6322.306037
## iter  50 value 6210.813596
## iter  60 value 6191.879484
## iter  70 value 6181.922834
## iter  80 value 6179.750285
## iter  90 value 6178.260831
## iter 100 value 6174.454626
## final  value 6174.454626
## stopped after 100 iterations
## # weights:  98 (78 variable)
## initial  value 11377.736642
## iter  10 value 7765.970181
## iter  20 value 7178.147978
## iter  30 value 6800.706093
## iter  40 value 6364.103190
## iter  50 value 6251.083918
## iter  60 value 6244.874247
## iter  70 value 6244.670430
## final  value 6244.669911
## converged
## # weights:  98 (78 variable)
## initial  value 11377.736642
## iter  10 value 7765.965226
## iter  20 value 7177.777836
## iter  30 value 6796.736017
## iter  40 value 6322.349710
## iter  50 value 6210.870031
## iter  60 value 6191.998401
## iter  70 value 6182.357118
## iter  80 value 6180.348161
## iter  90 value 6179.079845
## iter 100 value 6176.566245
## final  value 6176.566245
## stopped after 100 iterations
## # weights:  98 (78 variable)
## initial  value 11379.682552
## iter  10 value 7808.534913
## iter  20 value 7244.313548
## iter  30 value 6883.382777
## iter  40 value 6357.515593
## iter  50 value 6256.152562
## iter  60 value 6237.914473
```

```
## iter  70 value 6229.304711
## iter  80 value 6227.392719
## iter  90 value 6225.993559
## iter 100 value 6219.851646
## final  value 6219.851646
## stopped after 100 iterations
## # weights:  98 (78 variable)
## initial  value 11379.682552
## iter  10 value 7808.539657
## iter  20 value 7244.826802
## iter  30 value 6887.340414
## iter  40 value 6372.737081
## iter  50 value 6290.048697
## iter  60 value 6284.789825
## iter  70 value 6284.693417
## final  value 6284.693225
## converged
## # weights:  98 (78 variable)
## initial  value 11379.682552
## iter  10 value 7808.534918
## iter  20 value 7244.314062
## iter  30 value 6883.386771
## iter  40 value 6357.532556
## iter  50 value 6256.198747
## iter  60 value 6238.039628
## iter  70 value 6229.725454
## iter  80 value 6227.953680
## iter  90 value 6226.780786
## iter 100 value 6223.199529
## final  value 6223.199529
## stopped after 100 iterations
## # weights:  98 (78 variable)
## initial  value 11379.682552
## iter  10 value 7794.383458
## iter  20 value 7255.205070
## iter  30 value 6845.474746
## iter  40 value 6330.006753
## iter  50 value 6241.570980
## iter  60 value 6219.558168
## iter  70 value 6208.812399
## iter  80 value 6206.407266
## iter  90 value 6204.974572
## iter 100 value 6200.956420
## final  value 6200.956420
## stopped after 100 iterations
## # weights:  98 (78 variable)
## initial  value 11379.682552
## iter  10 value 7794.388521
## iter  20 value 7255.477142
## iter  30 value 6850.061807
## iter  40 value 6350.139101
## iter  50 value 6277.498192
## iter  60 value 6272.633250
## iter  70 value 6272.464475
```

```
## final  value 6272.463725
## converged
## # weights:  98 (78 variable)
## initial  value 11379.682552
## iter  10 value 7794.383463
## iter  20 value 7255.205342
## iter  30 value 6845.479369
## iter  40 value 6330.029819
## iter  50 value 6241.617236
## iter  60 value 6219.681425
## iter  70 value 6209.256309
## iter  80 value 6207.018371
## iter  90 value 6205.798895
## iter 100 value 6203.161203
## final  value 6203.161203
## stopped after 100 iterations
## # weights:  98 (78 variable)
## initial  value 12642.578238
## iter  10 value 8565.485030
## iter  20 value 7854.843779
## iter  30 value 7458.832546
## iter  40 value 7047.638740
## iter  50 value 6926.817085
## iter  60 value 6906.882621
## iter  70 value 6896.805511
## iter  80 value 6894.255795
## iter  90 value 6892.919267
## iter 100 value 6889.542414
## final  value 6889.542414
## stopped after 100 iterations
```

```
print(multi_model_cv)
```

```
## Penalized Multinomial Regression
##
## 6497 samples
##   12 predictor
##    7 classes: '3', '4', '5', '6', '7', '8', '9'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5848, 5847, 5846, 5846, 5848, 5847, ...
## Resampling results across tuning parameters:
##
##   decay  Accuracy   Kappa
##   0e+00  0.5397853  0.2587396
##   1e-04  0.5400932  0.2588258
##   1e-01  0.5396300  0.2548471
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was decay = 1e-04.
```

```r
predictions <- predict(multi_model_cv, newdata = test)

confusionMatrix(predictions, test$quality)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8   9
##          3   2   0   0   0   0   0   0
##          4   0   3   2   0   0   0   0
##          5   2  40 392 188  24   6   0
##          6   4  23 240 607 229  38   1
##          7   0   1   1  66  64  15   2
##          8   0   0   0   0   0   0   0
##          9   0   0   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.5477
##                  95% CI : (0.5253, 0.57)
##     No Information Rate : 0.4415
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.2705
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity          0.250000 0.044776   0.6173   0.7050  0.20189  0.00000
## Specificity          1.000000 0.998938   0.8023   0.5087  0.94795  1.00000
## Pos Pred Value       1.000000 0.600000   0.6012   0.5315  0.42953      NaN
## Neg Pred Value       0.996920 0.967095   0.8128   0.6856  0.85952  0.96974
## Prevalence           0.004103 0.034359   0.3256   0.4415  0.16256  0.03026
## Detection Rate       0.001026 0.001538   0.2010   0.3113  0.03282  0.00000
## Detection Prevalence 0.001026 0.002564   0.3344   0.5856  0.07641  0.00000
## Balanced Accuracy    0.625000 0.521857   0.7098   0.6069  0.57492  0.50000
##                      Class: 9
## Sensitivity          0.000000
## Specificity          1.000000
## Pos Pred Value            NaN
## Neg Pred Value       0.998462
## Prevalence           0.001538
## Detection Rate       0.000000
## Detection Prevalence 0.000000
## Balanced Accuracy    0.500000
```

Accuracy= 0.5477

#Logistic Regression

Since logistic regression only has 2 output variables: 0 or 1 or "Yes" or "No". I decided to split the data in a way where wines that have a quality >= 7, are 0 and wines that have a quality < 7, are 1.

(https://vineroutes.com/wine-rating-system) shows the scale for wine quality.

```
wine_df <- wine_df %>%
  mutate(quality = as.numeric(as.character(quality))) %>%
  mutate(quality_binary = ifelse(quality >= 7, 1, 0)) %>%
  mutate(quality_binary = as.factor(quality_binary))

set.seed(1)
index2 <- sample(1:nrow(wine_df), 0.7 * nrow(wine_df))
train2 <- wine_df[index2, ]
test2 <- wine_df[-index2, ]

log_model <- glm(quality_binary ~ . - quality, data = train2, family = binomial)

predictions2 <- predict(log_model, test2, type = "response")
predicted_classes2 <- ifelse(predictions2 > 0.5, "1", "0")

confusionMatrix(as.factor(predicted_classes2), test2$quality_binary)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1482  272
##          1   89  107
##
##                Accuracy : 0.8149
##                  95% CI : (0.7969, 0.8319)
##     No Information Rate : 0.8056
##     P-Value [Acc > NIR] : 0.1583
##
##                   Kappa : 0.2763
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9433
##             Specificity : 0.2823
##          Pos Pred Value : 0.8449
##          Neg Pred Value : 0.5459
##              Prevalence : 0.8056
##          Detection Rate : 0.7600
##    Detection Prevalence : 0.8995
##       Balanced Accuracy : 0.6128
##
##        'Positive' Class : 0
##
```

ROC Curve

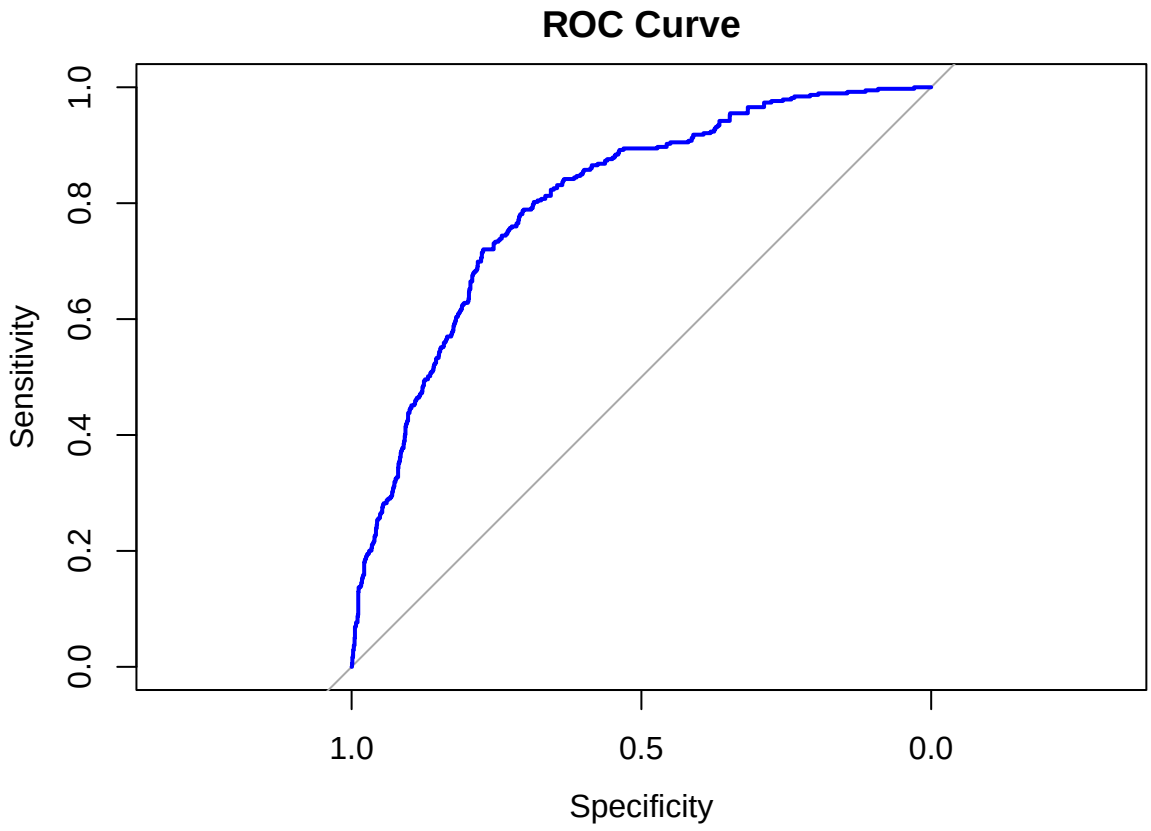
```
predictions_prob <- predict(log_model, test2, type = "response")

roc_curve <- roc(test2$quality_binary, predictions_prob)
```

```
## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```

```
plot(roc_curve, col = "blue", main = "ROC Curve")
```




```
auc(roc_curve)
```

```
## Area under the curve: 0.8028
```

AUC = 0.8028. That means the model does an okay job in predicting it and is not completely due to random chance.

Cross Validation

```
control2 <- trainControl(method = "cv", number = 10)

set.seed(1)
cv_model <- train(quality_binary ~ . - quality, data = wine_df, method = "glm", family = binomial, trCo

print(cv_model)
```

```
## Generalized Linear Model
##
## 6497 samples
##   13 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5847, 5847, 5847, 5847, 5848, 5847, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.8174517  0.2741793
```

```
predictions <- predict(cv_model, newdata = test2)

confusionMatrix(predictions, test2$quality_binary)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1490  275
##          1   81  104
##
##                Accuracy : 0.8174
##                  95% CI : (0.7996, 0.8344)
##     No Information Rate : 0.8056
##     P-Value [Acc > NIR] : 0.09824
##
##                   Kappa : 0.2766
##
##  Mcnemar's Test P-Value : < 2e-16
##
##             Sensitivity : 0.9484
##             Specificity : 0.2744
##          Pos Pred Value : 0.8442
##          Neg Pred Value : 0.5622
##              Prevalence : 0.8056
##          Detection Rate : 0.7641
##    Detection Prevalence : 0.9051
##       Balanced Accuracy : 0.6114
##
##        'Positive' Class : 0
##
```

Since the data is a little skewed with a lot more samples landing in the 0 category than the 1 category, which might affect the accuracy of the model, I'm going to try adding weights to the regression to try to make it more accurate

Weighted Logistic Regression

```
wine_df <- wine_df %>%
  mutate(quality = as.numeric(as.character(quality))) %>%
  mutate(quality_binary = ifelse(quality >= 7, 1, 0))

set.seed(1)
index3 <- sample(1:nrow(wine_df), 0.7 * nrow(wine_df))
train3 <- wine_df[index3, ]
test3 <- wine_df[-index3, ]

weights <- ifelse(train3$quality_binary == 1, 0.8, 0.2)

log_model_weighted <- glm(quality_binary ~ . - quality, data = train3, family = binomial, weights = wei
```

```
## Warning in eval(family$initialize): non-integer #successes in a binomial glm!
```

```
predictions_weighted <- predict(log_model_weighted, test3, type = "response")
predicted_classes_weighted <- ifelse(predictions_weighted > 0.5, "1", "0")

confusionMatrix(as.factor(predicted_classes_weighted), as.factor(test3$quality_binary))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1124   91
##          1  447  288
##
##                Accuracy : 0.7241
##                  95% CI : (0.7037, 0.7438)
##     No Information Rate : 0.8056
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3505
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.7155
##             Specificity : 0.7599
##          Pos Pred Value : 0.9251
##          Neg Pred Value : 0.3918
##              Prevalence : 0.8056
##          Detection Rate : 0.5764
##    Detection Prevalence : 0.6231
##       Balanced Accuracy : 0.7377
##
##        'Positive' Class : 0
##
```

```r
predictions_prob2 <- predict(log_model_weighted, test3, type = "response")
```

```r
# For normal logistic regression
roc_normal <- roc(test2$quality_binary, predictions_prob)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

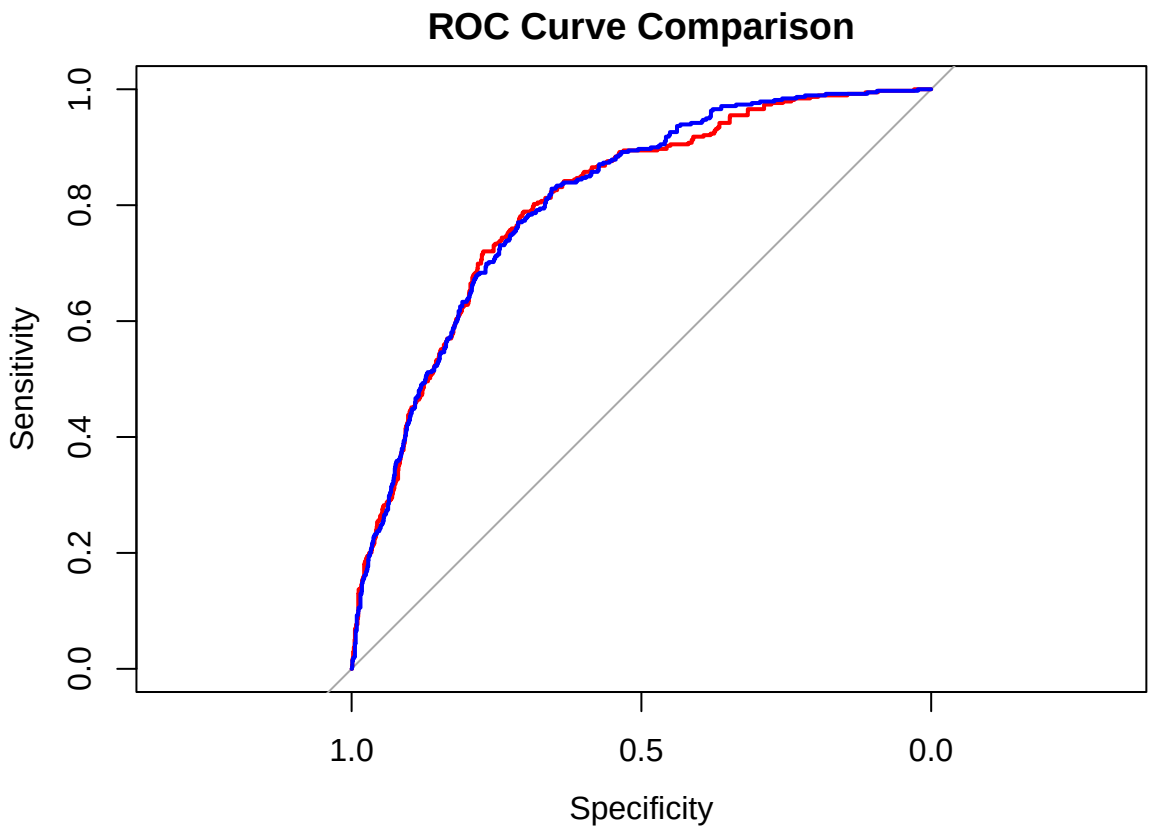
```r
plot(roc_normal, col = "red", main = "ROC Curve Comparison")
auc(roc_normal)
```

```
## Area under the curve: 0.8028
```

```r
# For weighted logistic regression
roc_weighted <- roc(test3$quality_binary, predictions_prob2)
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```r
plot(roc_weighted, col = "blue", add = TRUE)
```



```r
auc(roc_weighted)
```

```
## Area under the curve: 0.8053
```

The ROC curves for weighted/normal are basically the same. So the model's discriminatory power did not really improve.

Balances Logistic Regression

```r
minority_class <- wine_df %>% filter(quality_binary == 1)
majority_class <- wine_df %>% filter(quality_binary == 0)

set.seed(1)
majority_class_undersampled <- majority_class %>%
  sample_n(nrow(minority_class))

balanced_df <- bind_rows(minority_class, majority_class_undersampled)

set.seed(1)
index_balanced <- sample(1:nrow(balanced_df), 0.7 * nrow(balanced_df))
train_balanced <- balanced_df[index_balanced, ]
test_balanced <- balanced_df[-index_balanced, ]

log_model_balanced <- glm(quality_binary ~ . - quality, data = train_balanced, family = binomial)

predictions_balanced <- predict(log_model_balanced, test_balanced, type = "response")
predicted_classes_balanced <- ifelse(predictions_balanced > 0.5, "1", "0")

confusionMatrix(as.factor(predicted_classes_balanced), as.factor(test_balanced$quality_binary))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 301  72
##          1 104 290
##
##                Accuracy : 0.7705
##                  95% CI : (0.7391, 0.7999)
##     No Information Rate : 0.528
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.5418
##
##  Mcnemar's Test P-Value : 0.01945
##
##             Sensitivity : 0.7432
##             Specificity : 0.8011
##          Pos Pred Value : 0.8070
##          Neg Pred Value : 0.7360
```

```
##              Prevalence : 0.5280
##          Detection Rate : 0.3924
##    Detection Prevalence : 0.4863
##       Balanced Accuracy : 0.7722
##
##        'Positive' Class : 0
##
```

```r
# For normal logistic regression
roc_normal <- roc(test2$quality_binary, predictions_prob)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```r
plot(roc_normal, col = "red", main = "ROC Curve Comparison", lwd = 2)

auc_normal <- auc(roc_normal)
cat("AUC for Normal Logistic Regression:", auc_normal, "\n")
```

```
## AUC for Normal Logistic Regression: 0.8028011
```

```r
# For weighted logistic regression
roc_weighted <- roc(test3$quality_binary, predictions_prob2)
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```r
lines(roc_weighted, col = "blue", lwd = 2)

auc_weighted <- auc(roc_weighted)
cat("AUC for Weighted Logistic Regression:", auc_weighted, "\n")
```

```
## AUC for Weighted Logistic Regression: 0.8052633
```

```r
# For balanced logistic regression
roc_balanced <- roc(test_balanced$quality_binary, predictions_balanced)
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```r
lines(roc_balanced, col = "green", lwd = 2)

auc_balanced <- auc(roc_balanced)
cat("AUC for Undersampled Logistic Regression:", auc_balanced, "\n")
```

```
## AUC for Undersampled Logistic Regression: 0.8355296
```

```r
# Add Legend to the Plot
legend("bottomright",
       legend = c("Normal Logistic", "Weighted Logistic", "Undersampled Logistic"),
       col = c("red", "blue", "green"),
       lwd = 2)
```