Code ▾

EVALUATING WINE QUALITY VIA PHYSICOCHEMICAL TESTS

MATH 2319 Machine Learning Applied Project Phase II

HUYNH AI LOAN (s3655461)

11 June 2018

1 Introduction

The objective of this project is to build classifiers to predict whether physicochemical tests make thequality of wine larger than 5 grade in range of score between 0 (very bad) and 10 (very excellent) which are made by wine experts. The data sets were collected from the UCI Machine Learning Repository. In Phase I, we cleaned the data and re-categorised some descriptive features to be less granular. In Phase II, we built three binary-classifiers on the cleaned data. Section 2 describes an overview of our methodology. Section 3 discusses the classifiers and their tunning process. Section 4 compares the performance of the classifiers using the same resampling method. The last section concludes with a summary.

2 Methodology

In this report, the three classifiers - Random Forest (RF), K-Nearest Neighbour (KNN) and Support Vector Machince (SVM) are considered to deal with the problem. The target feature in dataset was grouped into two levels which were less than or equal 5 (<=5) and larger than 5 (>5). The datet was splitted into training and test set with ratio 7:3. Each classifier was trainned to make probability predictions in order that we could adjust prediction threshold to evaluate the performance. For fine-tuning process, we used 5-folded cross validation stratified sampling on each classifier.

Using the tuned hyperparmeters and the optimal thresholds defined from previous steps, we made prediction on the test data for each classifier. We used mean misclassification error rate (mmce) and confusion matrix on the test data to evaluate the classifiers's performance.

3 Hyperparameter Tune-Fining

3.1 K-Nearest Neighbour

KNN uses distance to classify the features. Therefore, it is necessary to standardize the predictor variables. There are two type of distances used in this report including Manhattan and Euclidian distance. In addition, we also ran a grid search on k values in range from 0 to 10 in order to define which the best k neighbours give the best result for model. The tunning result is as below:

Hide

```
inTrain <- createDataPartition(cleaned_data$quality, p = 0.7, list = FALSE)
training_data <- cleaned_data[inTrain,]
test_data <- cleaned_data[-inTrain,]
task <- makeClassifTask(data = training_data, target = 'quality', id = 'wine')
knn_learner <- makeLearner('classif.kknn', predict.type = 'prob')
ps_knn <- makeParamSet(
  makeDiscreteParam('k', values = seq(2, 10, by = 1)),
  makeDiscreteParam('distance', values=c(1,2)),
  makeDiscreteParam('kernel', values = "cos")
)
ctrl  <- makeTuneControlGrid()
rdesc <- makeResampleDesc("CV", iters = 5L, stratify = TRUE)
# Configure tune wrapper with tune-tuning settings
knn_tunedLearner <- makeTuneWrapper(learner = knn_learner, resampling = rdesc, measures = mmc
e, par.set= ps_knn, control = ctrl)
# # Train the tune wrappers
knn_tuneWrapper  <- mlr::train(knn_tunedLearner, task)
```

```
[Tune] Started tuning learner classif.kknn for parameter set:
             Type len Def             Constr Req Tunable Trafo
k        discrete   -   - 2,3,4,5,6,7,8,9,10   -    TRUE     -
distance discrete   -   -                1,2   -    TRUE     -
kernel   discrete   -   -                cos   -    TRUE     -
With control class: TuneControlGrid
Imputation value: 1
[Tune-x] 1: k=2; distance=1; kernel=cos
[Tune-y] 1: mmce.test.mean=0.2192030; time: 0.0 min
[Tune-x] 2: k=3; distance=1; kernel=cos
[Tune-y] 2: mmce.test.mean=0.2157200; time: 0.0 min
[Tune-x] 3: k=4; distance=1; kernel=cos
[Tune-y] 3: mmce.test.mean=0.2154045; time: 0.0 min
[Tune-x] 4: k=5; distance=1; kernel=cos
[Tune-y] 4: mmce.test.mean=0.2103377; time: 0.0 min
[Tune-x] 5: k=6; distance=1; kernel=cos
[Tune-y] 5: mmce.test.mean=0.2112876; time: 0.0 min
[Tune-x] 6: k=7; distance=1; kernel=cos
[Tune-y] 6: mmce.test.mean=0.2103362; time: 0.0 min
[Tune-x] 7: k=8; distance=1; kernel=cos
[Tune-y] 7: mmce.test.mean=0.2100218; time: 0.0 min
[Tune-x] 8: k=9; distance=1; kernel=cos
[Tune-y] 8: mmce.test.mean=0.2093884; time: 0.0 min
[Tune-x] 9: k=10; distance=1; kernel=cos
[Tune-y] 9: mmce.test.mean=0.2084405; time: 0.0 min
[Tune-x] 10: k=2; distance=2; kernel=cos
[Tune-y] 10: mmce.test.mean=0.2220536; time: 0.0 min
[Tune-x] 11: k=3; distance=2; kernel=cos
[Tune-y] 11: mmce.test.mean=0.2195235; time: 0.0 min
[Tune-x] 12: k=4; distance=2; kernel=cos
[Tune-y] 12: mmce.test.mean=0.2163534; time: 0.0 min
[Tune-x] 13: k=5; distance=2; kernel=cos
[Tune-y] 13: mmce.test.mean=0.2128704; time: 0.0 min
[Tune-x] 14: k=6; distance=2; kernel=cos
[Tune-y] 14: mmce.test.mean=0.2100228; time: 0.0 min
[Tune-x] 15: k=7; distance=2; kernel=cos
[Tune-y] 15: mmce.test.mean=0.2090724; time: 0.0 min
[Tune-x] 16: k=8; distance=2; kernel=cos
[Tune-y] 16: mmce.test.mean=0.2062208; time: 0.0 min
[Tune-x] 17: k=9; distance=2; kernel=cos
[Tune-y] 17: mmce.test.mean=0.2090749; time: 0.0 min
[Tune-x] 18: k=10; distance=2; kernel=cos
[Tune-y] 18: mmce.test.mean=0.2065433; time: 0.0 min
[Tune] Result: k=8; distance=2; kernel=cos : mmce.test.mean=0.2062208
```

Hide

```
# Get Tune Result
print(getTuneResult(knn_tuneWrapper))
```

```
Tune result:
Op. pars: k=8; distance=2; kernel=cos
mmce.test.mean=0.2062208
```

3.2 Support Vector Machine

We considered `gamma` and `cost` parameters for tuning. The `gamma` parameter defines how far the influence of a training data reaches. The higher value of gamma will try to fit training dataset. The `cost` of contrainst violation controls the trade off between smooth decision boundary and classifying the training points correctly. We experimented with `gamma` value from 0.5 to 3 and `cost` value in range of (0.5, 3).

Hide

```
svm_learner <- makeLearner('classif.svm', predict.type = 'prob')
ps_svm <- makeParamSet(
  makeDiscreteParam('gamma', values = c(0.5,1, 1.5, 2, 2.5, 3)),
  makeDiscreteParam('cost', values = c(0.5,1, 1.5, 2, 2.5, 3))
)
svm_tunedLearner <- makeTuneWrapper(svm_learner, rdesc, measures=list(acc,mmce), ps_svm, ctr
l)
svm_tuneWrapper  <- mlr::train(svm_tunedLearner, task)
```

```
[Tune] Started tuning learner classif.svm for parameter set:
          Type len Def            Constr Req Tunable Trafo
gamma discrete   -   - 0.5,1,1.5,2,2.5,3   -    TRUE     -
cost  discrete   -   - 0.5,1,1.5,2,2.5,3   -    TRUE     -
With control class: TuneControlGrid
Imputation value: -0Imputation value: 1
[Tune-x] 1: gamma=0.5; cost=0.5
[Tune-y] 1: acc.test.mean=0.7963174,mmce.test.mean=0.2036826; time: 0.3 min
[Tune-x] 2: gamma=1; cost=0.5
[Tune-y] 2: acc.test.mean=0.7947371,mmce.test.mean=0.2052629; time: 0.4 min
[Tune-x] 3: gamma=1.5; cost=0.5
[Tune-y] 3: acc.test.mean=0.7836541,mmce.test.mean=0.2163459; time: 0.4 min
[Tune-x] 4: gamma=2; cost=0.5
[Tune-y] 4: acc.test.mean=0.7773205,mmce.test.mean=0.2226795; time: 0.4 min
[Tune-x] 5: gamma=2.5; cost=0.5
[Tune-y] 5: acc.test.mean=0.7738355,mmce.test.mean=0.2261645; time: 0.4 min
[Tune-x] 6: gamma=3; cost=0.5
[Tune-y] 6: acc.test.mean=0.7754203,mmce.test.mean=0.2245797; time: 0.4 min
[Tune-x] 7: gamma=0.5; cost=1
[Tune-y] 7: acc.test.mean=0.8020186,mmce.test.mean=0.1979814; time: 0.4 min
[Tune-x] 8: gamma=1; cost=1
[Tune-y] 8: acc.test.mean=0.7944201,mmce.test.mean=0.2055799; time: 0.5 min
[Tune-x] 9: gamma=1.5; cost=1
[Tune-y] 9: acc.test.mean=0.7817549,mmce.test.mean=0.2182451; time: 0.5 min
[Tune-x] 10: gamma=2; cost=1
[Tune-y] 10: acc.test.mean=0.7773215,mmce.test.mean=0.2226785; time: 0.5 min
[Tune-x] 11: gamma=2.5; cost=1
[Tune-y] 11: acc.test.mean=0.7744694,mmce.test.mean=0.2255306; time: 0.5 min
[Tune-x] 12: gamma=3; cost=1
[Tune-y] 12: acc.test.mean=0.7747869,mmce.test.mean=0.2252131; time: 0.5 min
[Tune-x] 13: gamma=0.5; cost=1.5
[Tune-y] 13: acc.test.mean=0.8048712,mmce.test.mean=0.1951288; time: 0.4 min
[Tune-x] 14: gamma=1; cost=1.5
[Tune-y] 14: acc.test.mean=0.7966383,mmce.test.mean=0.2033617; time: 0.5 min
[Tune-x] 15: gamma=1.5; cost=1.5
```

3.3 Random Forest

For RF, we did experiment with `mtry` of 1 through 10. The result is as below:

Hide

```
rf_learner <- makeLearner('classif.randomForest', predict.type = 'prob')
ps_rf <- makeParamSet(
  makeDiscreteParam('mtry', values = seq(1,10, by = 1))
)
rf_tunedLearner <- makeTuneWrapper(rf_learner, rdesc, measures=list(acc,mmce), ps_rf, ctrl)
rf_tuneWrapper  <- mlr::train(rf_tunedLearner, task)
```

```
[Tune] Started tuning learner classif.randomForest for parameter set:
         Type len Def               Constr Req Tunable Trafo
mtry discrete   -   - 1,2,3,4,5,6,7,8,9,10   -    TRUE     -
With control class: TuneControlGrid
Imputation value: -0Imputation value: 1
[Tune-x] 1: mtry=1
[Tune-y] 1: acc.test.mean=0.8305422,mmce.test.mean=0.1694578; time: 0.1 min
[Tune-x] 2: mtry=2
[Tune-y] 2: acc.test.mean=0.8359285,mmce.test.mean=0.1640715; time: 0.1 min
[Tune-x] 3: mtry=3
[Tune-y] 3: acc.test.mean=0.8321265,mmce.test.mean=0.1678735; time: 0.2 min
[Tune-x] 4: mtry=4
[Tune-y] 4: acc.test.mean=0.8305422,mmce.test.mean=0.1694578; time: 0.2 min
[Tune-x] 5: mtry=5
[Tune-y] 5: acc.test.mean=0.8286405,mmce.test.mean=0.1713595; time: 0.2 min
[Tune-x] 6: mtry=6
[Tune-y] 6: acc.test.mean=0.8235727,mmce.test.mean=0.1764273; time: 0.2 min
[Tune-x] 7: mtry=7
[Tune-y] 7: acc.test.mean=0.8232572,mmce.test.mean=0.1767428; time: 0.2 min
[Tune-x] 8: mtry=8
[Tune-y] 8: acc.test.mean=0.8232582,mmce.test.mean=0.1767418; time: 0.2 min
[Tune-x] 9: mtry=9
[Tune-y] 9: acc.test.mean=0.8251585,mmce.test.mean=0.1748415; time: 0.2 min
[Tune-x] 10: mtry=10
[Tune-y] 10: acc.test.mean=0.8194563,mmce.test.mean=0.1805437; time: 0.2 min
[Tune] Result: mtry=2 : acc.test.mean=0.8359285,mmce.test.mean=0.1640715
```

Hide

```
# Get Tune Result
print(getTuneResult(rf_tuneWrapper))
```

```
Tune result:
Op. pars: mtry=2
acc.test.mean=0.8359285,mmce.test.mean=0.1640715
```

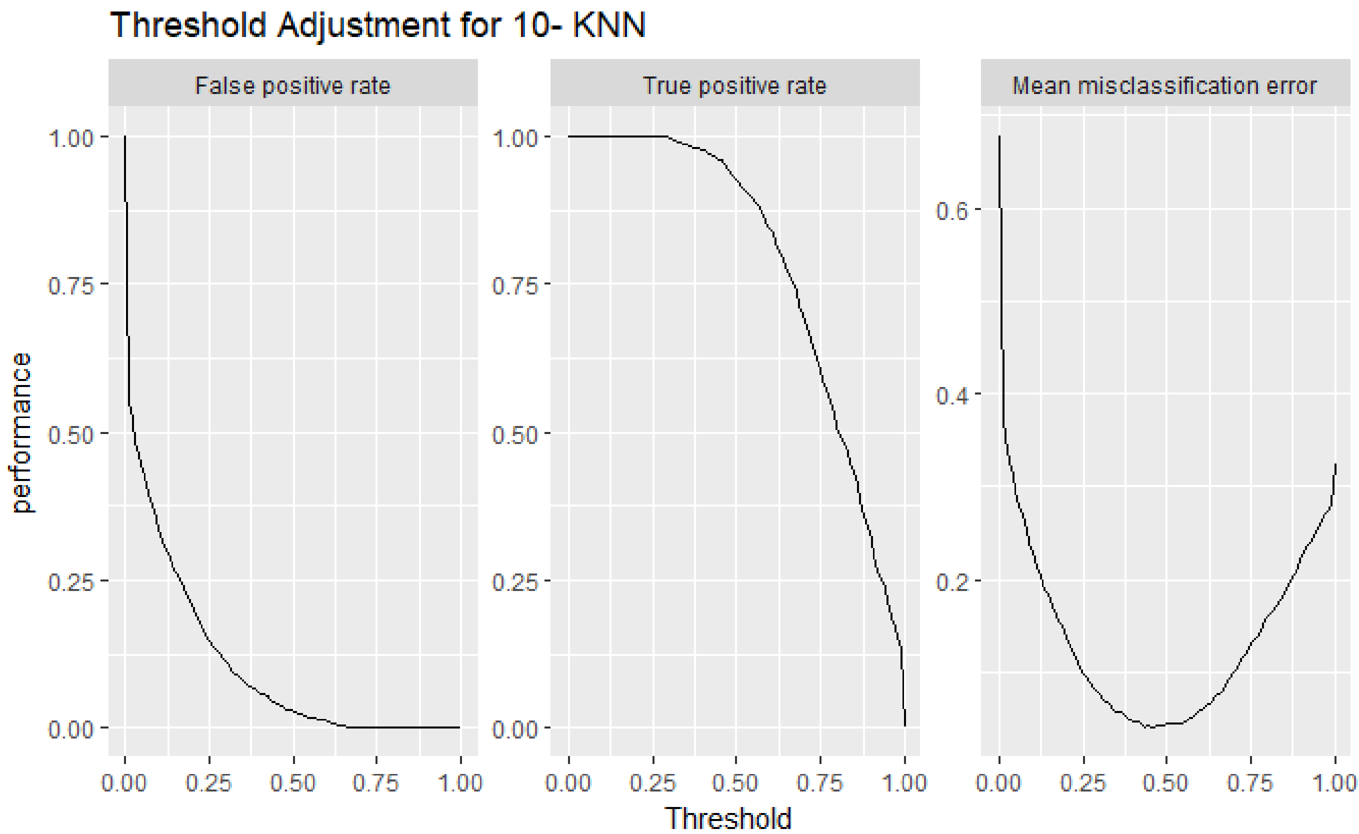
3.4 Threshold Adjustment

The following figures show the value of mmce vs the range of probability thresholds. The thresholds which may be used to determine the probability of wine with above average quality (quality >5) were approximately 0.45, 0.28 and 0.37 for 10-KNN, SVM and RF respectively.

3.4.1 KNN

Hide

```
# Predict on training data
knn_tunePredict <- predict(knn_tuneWrapper, task)
# Get threshold values for KNN learner ----
dt_knn_thresholds <- generateThreshVsPerfData(knn_tunePredict, measures = list(fpr, tpr, mmc
e))
# Plot thresholds adjustment for each learner
mlr::plotThreshVsPerf(dt_knn_thresholds) + labs(title = 'Threshold Adjustment for 10- KNN', x
 = 'Threshold')
```

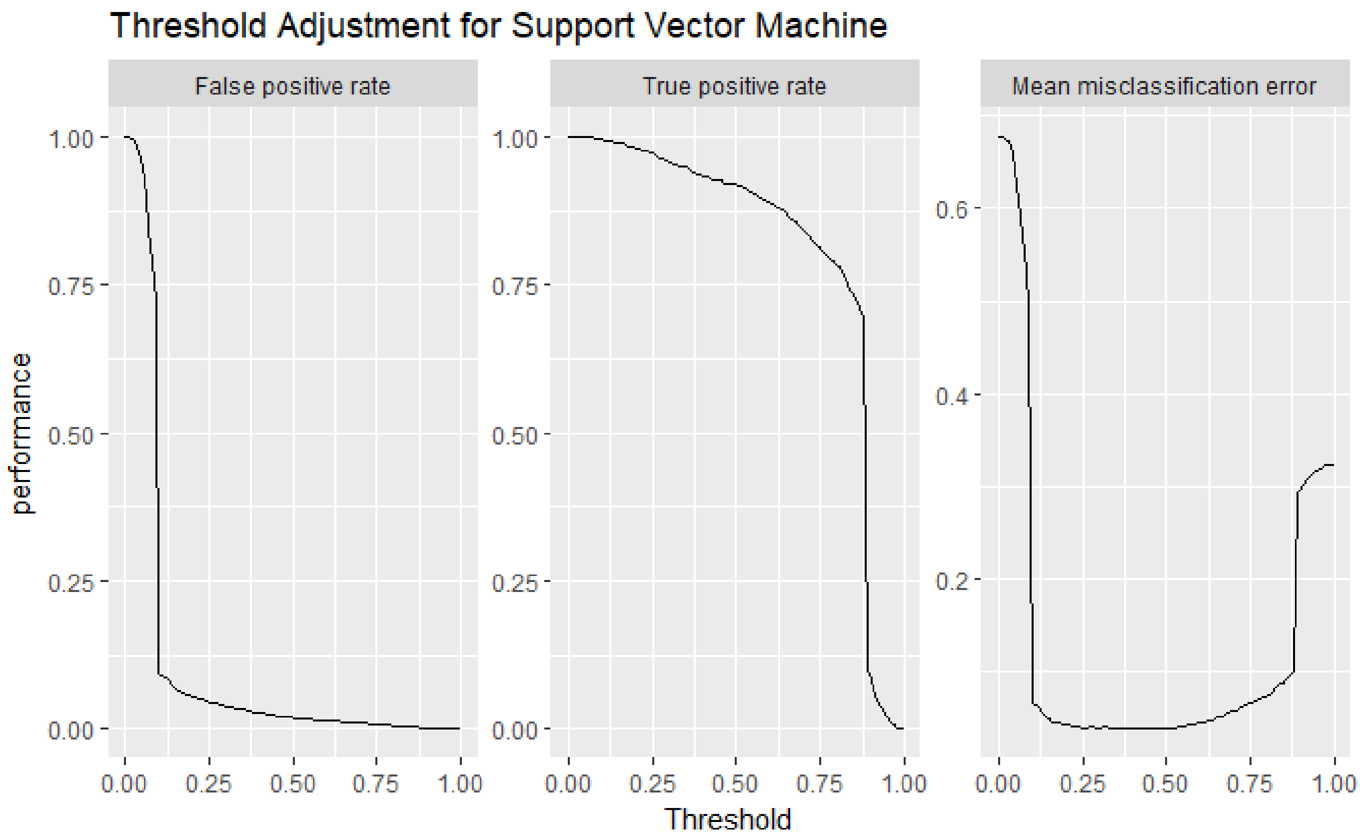


Hide

```
# Get Threshold value of KNN
knn_threshold<- dt_knn_thresholds$data$threshold[ which.min(dt_knn_thresholds$data$mmce) ]
knn_threshold
```

```
[1] 0.4545455
```

3.4.2 Suport Vector Machine



Hide

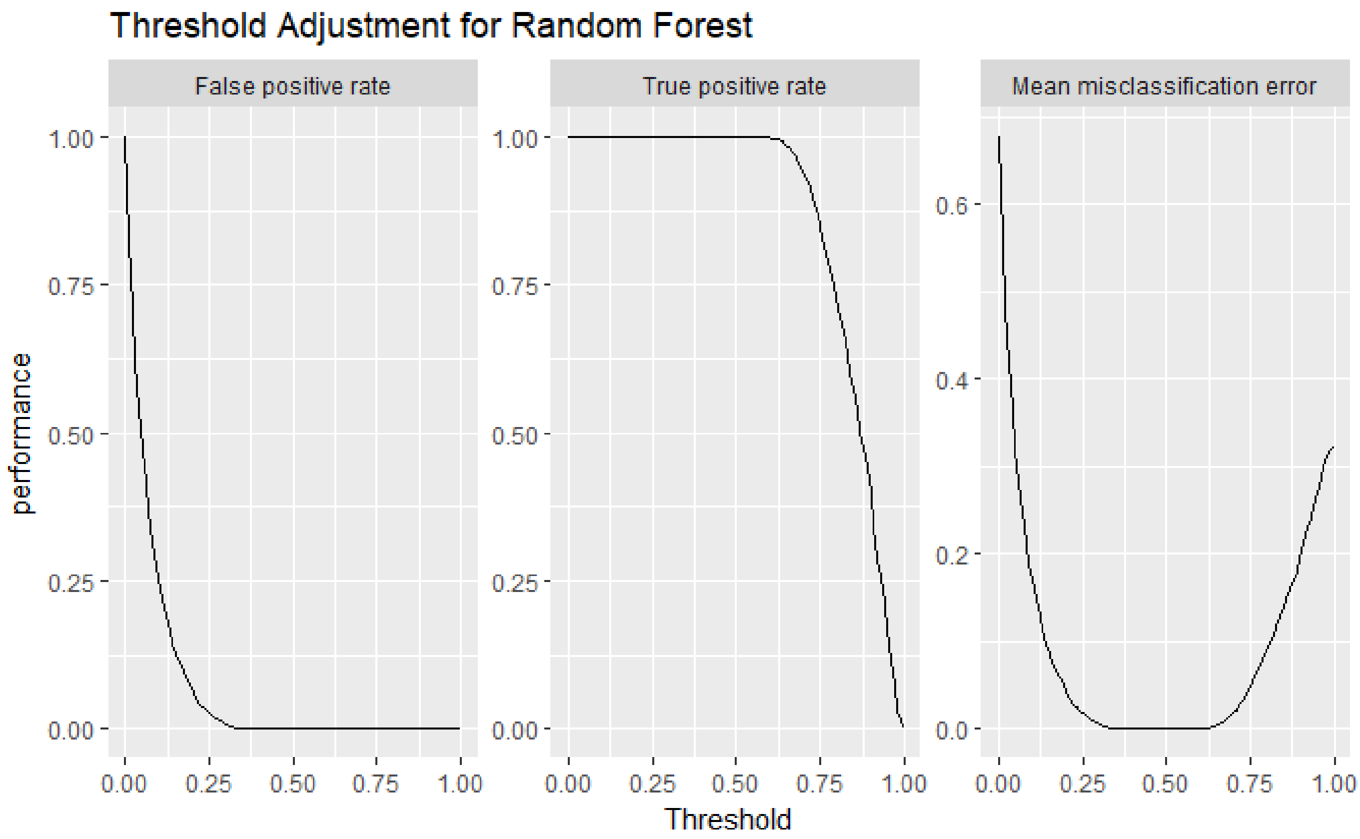
```
# Get Threshold value of SVM:
svm_threshold<- dt_svm_thresholds$data$threshold[ which.min(dt_svm_thresholds$data$mmce) ]
svm_threshold
```

```
[1] 0.3535354
```

3.4.3 Random Forest

Hide

```
rf_tunePredict <- predict(rf_tuneWrapper, task)
dt_rf_thresholds <- generateThreshVsPerfData(rf_tunePredict, measures = list(fpr, tpr, mmce))
mlr::plotThreshVsPerf(dt_rf_thresholds) + labs(title = 'Threshold Adjustment for Random Fores
t', x = 'Threshold')
```



Hide

```
# Get threshold for RF
rf_threshold<- dt_rf_thresholds$data$threshold[ which.min(dt_rf_thresholds$data$mmce) ]
rf_threshold
```

```
[1] 0.3636364
```

4 Evaluation

Making prediction on test data for each classifier

```
[1] "===== Predict test data using KNN ========="
Prediction: 1352 observations
predict.type: prob
threshold: <=5=0.45,>=5=0.55
time: 0.20
   truth   prob.<=5  prob.>=5 response
1    >=5 0.08908340 0.9109166      >=5
2    >=5 0.03434164 0.9656584      >=5
12   >=5 0.29859661 0.7014034      >=5
14   >=5 0.00000000 1.0000000      >=5
16   >=5 0.25270199 0.7472980      >=5
17   <=5 0.65160903 0.3483910      <=5
... (#rows: 1352, #cols: 4)
[1] "===== Predict test data using SVM ========="
Prediction: 1352 observations
predict.type: prob
threshold: <=5=0.35,>=5=0.65
time: 0.30
   truth  prob.>=5   prob.<=5 response
1    >=5 0.9037137 0.09628633      >=5
2    >=5 0.9036612 0.09633884      >=5
12   >=5 0.6489929 0.35100713      >=5
14   >=5 0.9476377 0.05236232      >=5
16   >=5 0.7202245 0.27977545      >=5
17   <=5 0.1300488 0.86995116      <=5
... (#rows: 1352, #cols: 4)
[1] "===== Predict test data using RF ========="
Prediction: 1352 observations
predict.type: prob
threshold: <=5=0.36,>=5=0.64
time: 0.09
   truth prob.<=5 prob.>=5 response
1    >=5    0.272    0.728      >=5
2    >=5    0.174    0.826      >=5
12   >=5    0.194    0.806      >=5
14   >=5    0.090    0.910      >=5
16   >=5    0.410    0.590      <=5
17   <=5    0.700    0.300      <=5
... (#rows: 1352, #cols: 4)
```

Using the parameters and threshold levels, we calculated the ROC measures for each classifier. The Confusion Matrix and ROC measures of KNN classifer is as follow:

```
[1] "=========== Confusion Matrix==============="
Relative confusion matrix (normalized by row/column):
        predicted
true     <=5       >=5       -err.-
  <=5    0.64/0.67 0.36/0.17 0.36
  >=5    0.15/0.33 0.85/0.83 0.15
  -err.-      0.33      0.17 0.22


Absolute confusion matrix:
        predicted
true     <=5 >=5 -err.-
  <=5    278 159    159
  >=5    140 775    140
  -err.- 140 159    299
[1] "========= ROC Measures ========="
     predicted
true  <=5       >=5
  <=5 278       159       tpr: 0.64 fnr: 0.36
  >=5 140       775       fpr: 0.15 tnr: 0.85
      ppv: 0.67 for: 0.17 lrp: 4.16 acc: 0.78
      fdr: 0.33 npv: 0.83 lrm: 0.43 dor: 9.68


Abbreviations:
tpr - True positive rate (Sensitivity, Recall)
fpr - False positive rate (Fall-out)
fnr - False negative rate (Miss rate)
tnr - True negative rate (Specificity)
ppv - Positive predictive value (Precision)
for - False omission rate
lrp - Positive likelihood ratio (LR+)
fdr - False discovery rate
npv - Negative predictive value
acc - Accuracy
lrm - Negative likelihood ratio (LR-)
dor - Diagnostic odds ratio
```

The Confusion Matrix and ROC measures of SVM classifer is as follow:

```
[1] "=========== Confusion Matrix==============="
Relative confusion matrix (normalized by row/column):
        predicted
true     <=5       >=5       -err.-
  <=5    0.62/0.74 0.38/0.17 0.38
  >=5    0.10/0.26 0.90/0.83 0.10
  -err.-      0.26      0.17 0.19


Absolute confusion matrix:
        predicted
true     <=5 >=5 -err.-
  <=5    270 167    167
  >=5     96 819     96
  -err.-  96 167    263
[1] "========= ROC Measures ========="
     predicted
true  <=5       >=5
  <=5 270       167       tpr: 0.62 fnr: 0.38
  >=5 96        819       fpr: 0.1  tnr: 0.9
      ppv: 0.74 for: 0.17 lrp: 5.89 acc: 0.81
      fdr: 0.26 npv: 0.83 lrm: 0.43 dor: 13.79


Abbreviations:
tpr - True positive rate (Sensitivity, Recall)
fpr - False positive rate (Fall-out)
fnr - False negative rate (Miss rate)
tnr - True negative rate (Specificity)
ppv - Positive predictive value (Precision)
for - False omission rate
lrp - Positive likelihood ratio (LR+)
fdr - False discovery rate
npv - Negative predictive value
acc - Accuracy
lrm - Negative likelihood ratio (LR-)
dor - Diagnostic odds ratio
```

The Confusion Matrix and ROC measures of RF classifer is as follow:

```
[1] "=========== Confusion Matrix================"
Relative confusion matrix (normalized by row/column):
        predicted
true     <=5       >=5       -err.-
  <=5    0.63/0.77 0.37/0.16 0.37
  >=5    0.09/0.23 0.91/0.84 0.09
  -err.-      0.23      0.16 0.18


Absolute confusion matrix:
        predicted
true     <=5 >=5 -err.-
  <=5    277 160    160
  >=5     81 834     81
  -err.-  81 160    241
[1] "========= ROC Measures =========="
     predicted
true  <=5       >=5
  <=5 277       160       tpr: 0.63 fnr: 0.37
  >=5 81        834       fpr: 0.09 tnr: 0.91
      ppv: 0.77 for: 0.16 lrp: 7.16 acc: 0.82
      fdr: 0.23 npv: 0.84 lrm: 0.4  dor: 17.83


Abbreviations:
tpr - True positive rate (Sensitivity, Recall)
fpr - False positive rate (Fall-out)
fnr - False negative rate (Miss rate)
tnr - True negative rate (Specificity)
ppv - Positive predictive value (Precision)
for - False omission rate
lrp - Positive likelihood ratio (LR+)
fdr - False discovery rate
npv - Negative predictive value
acc - Accuracy
lrm - Negative likelihood ratio (LR-)
dor - Diagnostic odds ratio
```

It is obviously to see that RandomForest classier gave higher accuracy rate rather than KNN and SVM.

5 Discussion

Three models gave the accuracy more than 79%. But RandomForest produced the better performance. All three classifiers did perform high accuracy in predicting the quality of wine larger than 5. However, three models cannot deal with imbalance issues in this dataset.

In addition, the time execution for SVM is longer than KNN and RF. It might imply that SVM may be not suitable for large dataset.

6 Conclusion

Among three classifiers, the Random Forest produces the best performance in predicting whether physicochemical tests give the quality of wine larger than 5 grade of score between 0 (very bad) and 10 (very excellent) which were evaluated by wine experts. We split the dataset into training and test sets with ratio 7:3.

Based on that, we determined the optimal value of the selected hyperparameters of each classifier and the probability threshold. In the future work, we will consider another solution to deal with imbalance issues from this dataset.

7 References