红酒类别预测

杨帆 2020103661 统计学院流行病与卫生统计学

目录

1 作业

数据说明：

该数据集是 2009 年针对葡萄牙某款红酒的评测数据，其中前十一项指标均来自理化试验，而第十二项红酒质量评分指标是基于主观评价。

作业要求：

1、描述性统计分析

2、建立二分类模型预测"普通红酒"（v12=3，4，5）和"高质量红酒"（v12=6，7，8）

3、建立多分类模型预测红酒类别（v12）

主要事项：

1、如果有需要，进行适当的数据变换

2、至少使用三种预测模型，使用交叉验证方法进行模型选择，在测试集比较模型效果

```r
# 加载包
suppressPackageStartupMessages(library(vctrs))
suppressPackageStartupMessages(library(rlang))
suppressPackageStartupMessages(library(tidyverse))
suppressPackageStartupMessages(library(tidymodels))
suppressPackageStartupMessages(library(themis))
suppressPackageStartupMessages(library(glmnet))
suppressPackageStartupMessages(library(kernlab))
suppressPackageStartupMessages(library(flextable))
```

```r
# 读取数据
red_wine = read_csv("red_wine_quality_data.csv", col_types = cols()) %>% glimpse()
```

```
## Rows: 1,599
## Columns: 12
## $ `fixed acidity`        <dbl> 7.4, 7.8, 7.8, 11.2, 7.4, 7.4, 7.9, 7.3, 7.8...
## $ `volatile acidity`     <dbl> 0.700, 0.880, 0.760, 0.280, 0.700, 0.660, 0....
## $ `citric acid`          <dbl> 0.00, 0.00, 0.04, 0.56, 0.00, 0.00, 0.06, 0....
## $ `residual sugar`       <dbl> 1.9, 2.6, 2.3, 1.9, 1.9, 1.8, 1.6, 1.2, 2.0,...
```

```
## $ chlorides              <dbl> 0.076, 0.098, 0.092, 0.075, 0.076, 0.075, 0....
## $ `free sulfur dioxide`  <dbl> 11, 25, 15, 17, 11, 13, 15, 15, 9, 17, 15, 1...
## $ `total sulfur dioxide` <dbl> 34, 67, 54, 60, 34, 40, 59, 21, 18, 102, 65,...
## $ density                <dbl> 0.9978, 0.9968, 0.9970, 0.9980, 0.9978, 0.99...
## $ pH                     <dbl> 3.51, 3.20, 3.26, 3.16, 3.51, 3.51, 3.30, 3....
## $ sulphates              <dbl> 0.56, 0.68, 0.65, 0.58, 0.56, 0.56, 0.46, 0....
## $ alcohol                <dbl> 9.4, 9.8, 9.8, 9.8, 9.4, 9.4, 9.4, 10.0, 9.5...
## $ quality                <dbl> 5, 5, 5, 6, 5, 5, 5, 7, 7, 5, 5, 5, 5, 5, 5,...
```

```
# knitr::kable(head(red_wine),
#              caption = 'red_wine_quality_data',align='c')
```

2 1、描述性统计分析

```
# 检查缺失值
apply(red_wine, 2, function(x) any(is.na(x)))
```

```
##        fixed acidity     volatile acidity          citric acid
##                FALSE                FALSE                FALSE
##       residual sugar            chlorides  free sulfur dioxide
##                FALSE                FALSE                FALSE
## total sulfur dioxide              density                   pH
##                FALSE                FALSE                FALSE
##            sulphates              alcohol              quality
##                FALSE                FALSE                FALSE
```

可以看到该数据集没有缺失值。

```
# summary
summary(red_wine[,-ncol(red_wine)])
```

```
##  fixed acidity   volatile acidity  citric acid    residual sugar
```

```
##  Min.   : 4.60   Min.   :0.1200   Min.   :0.000   Min.   : 0.900
##  1st Qu.: 7.10   1st Qu.:0.3900   1st Qu.:0.090   1st Qu.: 1.900
##  Median : 7.90   Median :0.5200   Median :0.260   Median : 2.200
##  Mean   : 8.32   Mean   :0.5278   Mean   :0.271   Mean   : 2.539
##  3rd Qu.: 9.20   3rd Qu.:0.6400   3rd Qu.:0.420   3rd Qu.: 2.600
##  Max.   :15.90   Max.   :1.5800   Max.   :1.000   Max.   :15.500
##    chlorides        free sulfur dioxide total sulfur dioxide    density
##  Min.   :0.01200   Min.   : 1.00       Min.   :  6.00       Min.   :0.9901
##  1st Qu.:0.07000   1st Qu.: 7.00       1st Qu.: 22.00       1st Qu.:0.9956
##  Median :0.07900   Median :14.00       Median : 38.00       Median :0.9968
##  Mean   :0.08747   Mean   :15.87       Mean   : 46.47       Mean   :0.9967
##  3rd Qu.:0.09000   3rd Qu.:21.00       3rd Qu.: 62.00       3rd Qu.:0.9978
##  Max.   :0.61100   Max.   :72.00       Max.   :289.00       Max.   :1.0037
##        pH          sulphates         alcohol
##  Min.   :2.740   Min.   :0.3300   Min.   : 8.40
##  1st Qu.:3.210   1st Qu.:0.5500   1st Qu.: 9.50
##  Median :3.310   Median :0.6200   Median :10.20
##  Mean   :3.311   Mean   :0.6581   Mean   :10.42
##  3rd Qu.:3.400   3rd Qu.:0.7300   3rd Qu.:11.10
##  Max.   :4.010   Max.   :2.0000   Max.   :14.90
```

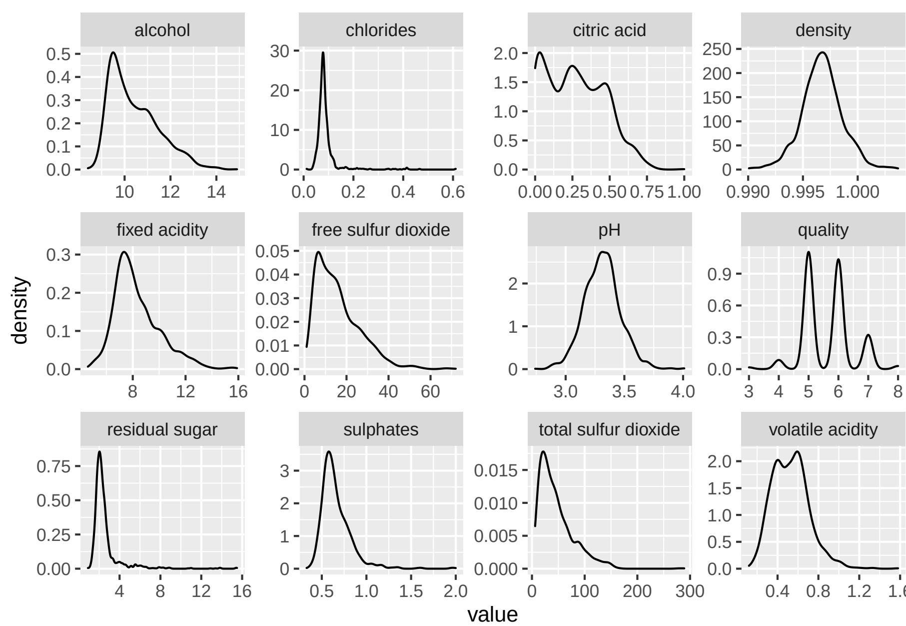
```r
# 红酒质量频数统计
red_wine %>%
  count(quality) %>%
  mutate(prop = n/sum(n))
```

```
## # A tibble: 6 x 3
##   quality     n    prop
##     <dbl> <int>   <dbl>
## 1       3    10 0.00625
## 2       4    53 0.0331
## 3       5   681 0.426
## 4       6   638 0.399
## 5       7   199 0.124
```

```
## 6         8     18 0.0113
```

可以看到类别 3、4、8 占比较低，不到 4%

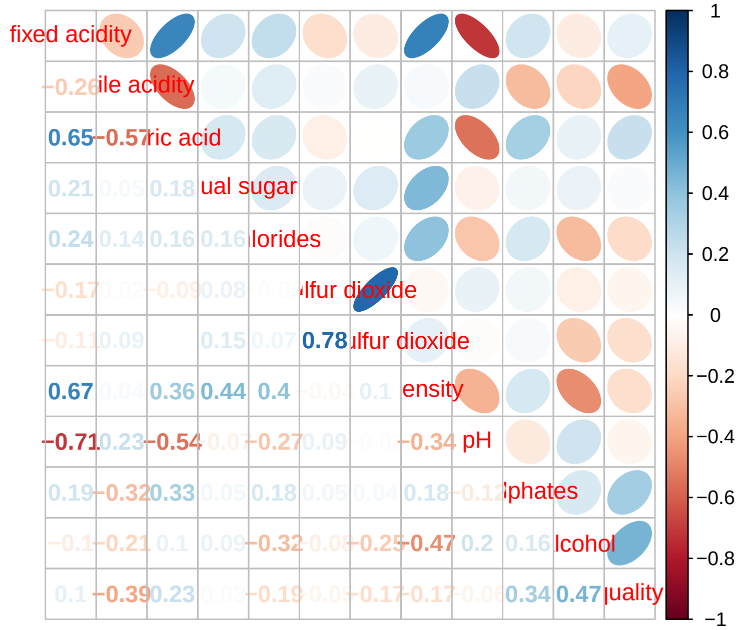
```
# 绘制直方图
red_wine %>%
  recipe(~.) %>%
  prep() %>%
  juice() %>%
  gather(Predictor, value)%>%
  ggplot(aes(value))+
  geom_density()+
  # geom_histogram()
  facet_wrap(~Predictor, scales = "free")
```



可以看到密度（density）、PH 接近正态分布，其他变量分布呈现偏态或者双峰。

下面将变量通过 Box_Cox 变换进行相关性分析

```r
# 变量相关系数矩阵
red_wine %>%
  recipe(~.) %>%
  step_BoxCox(all_predictors(),-quality) %>%
  prep() %>%
  juice() %>%
  cor() %>%
  corrplot::corrplot.mixed(upper = "ellipse")
```



可以看到红酒质量（quality）与酒精浓度（alcohol）、硫酸盐（sulphates）、柠檬酸（citric acid）、非挥发性酸（fixed acidity）呈正相关关系，但相关性不是很高，与红酒质量红酒质量（quality）相关性最高的是酒精浓度（alcohol），为 0.47；

红酒质量（quality）与密度（density）、游离二氧化硫（free sulfur dioxide）、氯化物（chlorides）、挥发性酸度（volatile acidity）呈负相关，相关性同样不是很高，其与挥发性酸度（volatile acidity）相关性最高，为-0.39。

此外，解释变量中，非挥发性酸（fixed acidity）与柠檬酸（citric acid）、总二氧化硫（total sulfur dioxide）呈正相关，相关系数分别为 0.65 和 0.67；

与 PH 呈负相关，为-0.71.

相关性较大的有游离二氧化硫（free sulfur dioxide）和总二氧化硫（total sulfur dioxide），相关系数为 0.78；挥发性酸度（volatile acidity）与柠檬酸（citric acid）呈负相关，为-0.57；柠檬酸（citric acid）与 PH 呈负相关，相关系数为-0.54。

3　2、预测“普通红酒”（v12=3，4，5）和“高质量红酒”（v12=6，7，8）

```
dat_rw = red_wine
# 创建分类标签
dat_rw$class = ifelse(dat_rw$quality %in% c(3,4,5),0,1)
dat_rw$quality = NULL

glimpse(dat_rw)
```

```
## Rows: 1,599
## Columns: 12
## $ `fixed acidity`        <dbl> 7.4, 7.8, 7.8, 11.2, 7.4, 7.4, 7.9, 7.3, 7.8...
## $ `volatile acidity`     <dbl> 0.700, 0.880, 0.760, 0.280, 0.700, 0.660, 0....
## $ `citric acid`          <dbl> 0.00, 0.00, 0.04, 0.56, 0.00, 0.00, 0.06, 0....
## $ `residual sugar`       <dbl> 1.9, 2.6, 2.3, 1.9, 1.9, 1.8, 1.6, 1.2, 2.0,...
## $ chlorides              <dbl> 0.076, 0.098, 0.092, 0.075, 0.076, 0.075, 0....
## $ `free sulfur dioxide`  <dbl> 11, 25, 15, 17, 11, 13, 15, 15, 9, 17, 15, 1...
## $ `total sulfur dioxide` <dbl> 34, 67, 54, 60, 34, 40, 59, 21, 18, 102, 65,...
## $ density                <dbl> 0.9978, 0.9968, 0.9970, 0.9980, 0.9978, 0.99...
## $ pH                     <dbl> 3.51, 3.20, 3.26, 3.16, 3.51, 3.51, 3.30, 3....
## $ sulphates              <dbl> 0.56, 0.68, 0.65, 0.58, 0.56, 0.56, 0.46, 0....
## $ alcohol                <dbl> 9.4, 9.8, 9.8, 9.8, 9.4, 9.4, 9.4, 10.0, 9.5...
## $ class                  <dbl> 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,...
```

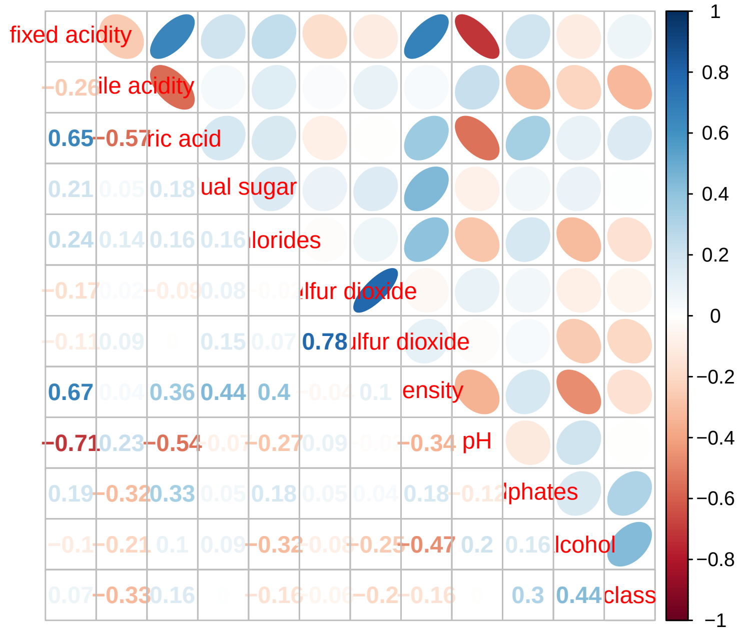
3.1　描述统计

```r
# 因变量分布情况
dat_rw %>%
  count(class) %>%
  mutate(prop = n/sum(n))
```

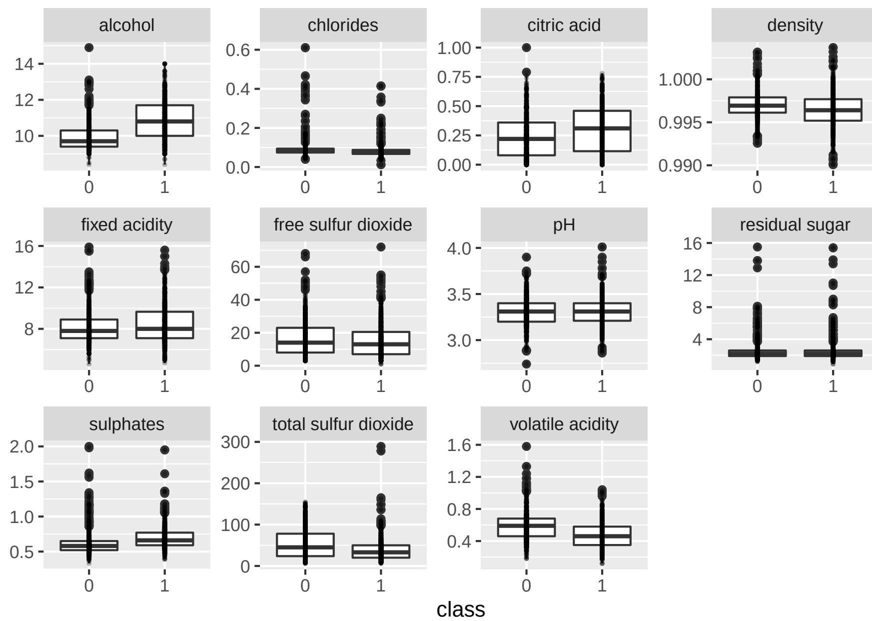
```
## # A tibble: 2 x 3
##   class     n  prop
##   <dbl> <int> <dbl>
## 1     0   744 0.465
## 2     1   855 0.535
```

因变量分布较为均衡

```r
# 相关系数矩阵
dat_rw %>%
  recipe(~.) %>%
  step_BoxCox(all_predictors(),-class) %>%
  prep() %>%
  juice() %>%
  cor() %>%
  corrplot::corrplot.mixed(upper = "ellipse")
```




```r
# 绘制各个变量箱线图
dat_rw$class = as.factor(dat_rw$class)
dat_rw %>%
  recipe(class ~ .) %>%
  prep() %>%
  juice() %>%
  gather(Predictor, value, -class)%>%
  ggplot(aes(x = class, y = value)) +
  geom_boxplot() +
  geom_point(alpha = 0.3, cex = .5) +
  facet_wrap(~Predictor, scales = "free") +
  ylab("")
```



可以看到酒精 (alcohol)、柠檬酸 (citric acid)、密度 (density)、硫酸盐 (sulphates)、挥发性酸度 (volatile acidity) 对因变量有较大影响。

```
# 划分训练集、测试集
df = dat_rw
set.seed(2020)
df_split = initial_split(df, prop=0.75, strata = class)

df_train = training(df_split)
df_test = testing(df_split)

# 训练集、测试机因变量分布
df_train %>%
  count(class) %>%
  mutate(prop = n/sum(n))
```

```
## # A tibble: 2 x 3
##   class     n  prop
##   <fct> <int> <dbl>
```

```
## 1 0        558 0.465
## 2 1        642 0.535
```

```
df_test %>%
  count(class) %>%
  mutate(prop = n/sum(n))
```

```
## # A tibble: 2 x 3
##   class      n  prop
##   <fct> <int> <dbl>
## 1 0       186 0.466
## 2 1       213 0.534
```

训练集、测试集因变量分布接近，且较为均衡。

```
# 将训练数据划分为 10 折
df_vfold<-vfold_cv(df_train,v=10,repeats=1)
df_vfold
```

```
## #  10-fold cross-validation
## # A tibble: 10 x 2
##    splits             id
##    <list>             <chr>
##  1 <split [1.1K/120]> Fold01
##  2 <split [1.1K/120]> Fold02
##  3 <split [1.1K/120]> Fold03
##  4 <split [1.1K/120]> Fold04
##  5 <split [1.1K/120]> Fold05
##  6 <split [1.1K/120]> Fold06
##  7 <split [1.1K/120]> Fold07
##  8 <split [1.1K/120]> Fold08
##  9 <split [1.1K/120]> Fold09
## 10 <split [1.1K/120]> Fold10
```

下面分别采用 logistic 回归、随机森林、boosted tree 进行分类

3.2 logistic 回归

```
# 定义 recipe
lr_recipe = df_train %>%
  recipe(class ~ .)
```

```
# 定义模型
lr_model <-
  logistic_reg(penalty = tune(), mixture = 1) %>%
  set_engine("glmnet")
```

```
# 定义工作流
lr_wfl =
  workflow() %>%
  add_recipe(lr_recipe) %>%
  add_model(lr_model)
lr_wfl
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: logistic_reg()
##
## -- Preprocessor ----------------------------------------------------------------
## 0 Recipe Steps
##
## -- Model -----------------------------------------------------------------------
## Logistic Regression Model Specification (classification)
##
```

```
## Main Arguments:
##   penalty = tune()
##   mixture = 1
##
## Computational engine: glmnet
```

```r
# 创建调节参数的格点集
lr_reg_grid <- tibble(penalty = 10^seq(-4, -1, length.out = 30))

## 最小的 5 个 lambda
lr_reg_grid %>% top_n(-5)
```

```
## Selecting by penalty

## # A tibble: 5 x 1
##    penalty
##      <dbl>
## 1 0.0001
## 2 0.000127
## 3 0.000161
## 4 0.000204
## 5 0.000259
```

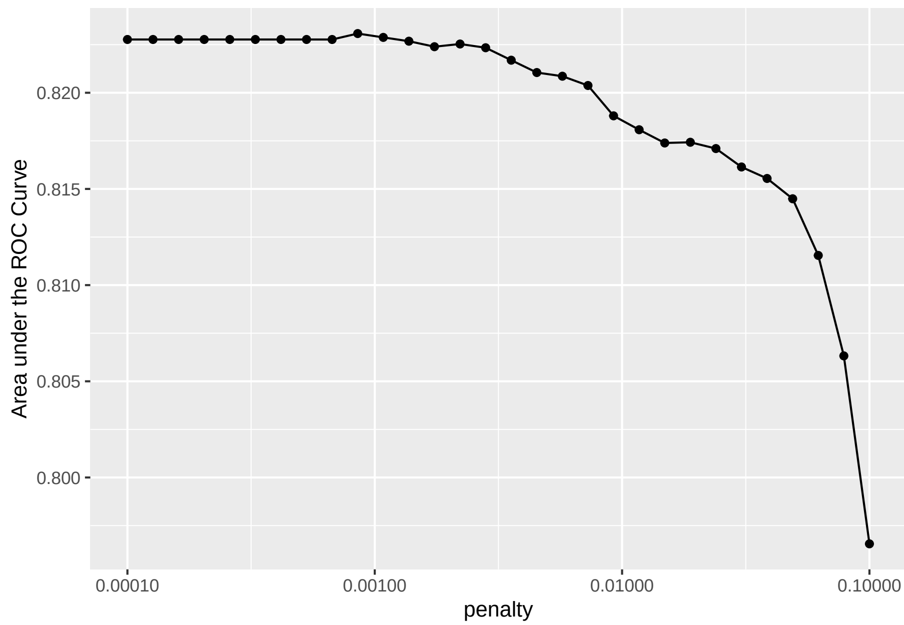
```r
## 最大的 5 个 lambda
lr_reg_grid %>% top_n(5)
```

```
## Selecting by penalty

## # A tibble: 5 x 1
##   penalty
##     <dbl>
## 1  0.0386
## 2  0.0489
## 3  0.0621
```

```
## 4  0.0788
## 5  0.1
```

```
doParallel::registerDoParallel()
# 训练模型及调参
lr_tune =
  lr_wfl %>%
  tune_grid(df_vfold,
            grid = lr_reg_grid,
            control = control_grid(save_pred = TRUE),
            metrics = metric_set(roc_auc))
```

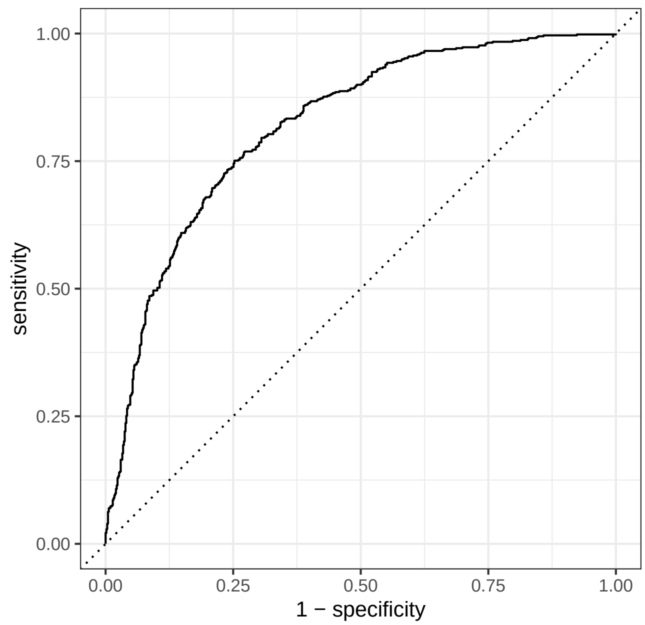
```
# 可视化不同惩罚参数下的 AUC
lr_tune %>%
  collect_metrics() %>%
  filter(.metric=='roc_auc') %>%
  ggplot(aes(x = penalty, y = mean)) +
  geom_point() +
  geom_line() +
  ylab("Area under the ROC Curve") +
  scale_x_log10(labels = scales::label_number())
```




```
# 根据 AUC 值选出最好的惩罚参数
lr_best = select_best(lr_tune,metric = "roc_auc")
lr_best
```

```
## # A tibble: 1 x 2
##    penalty .config
##      <dbl> <chr>
## 1 0.000853 Preprocessor1_Model10
```

```
# 交叉验证最好的模型 ROC 曲线
lr_auc <-
  lr_tune %>%
  collect_predictions(parameters = lr_best) %>%
  roc_curve(class, .pred_0) %>%
  mutate(model = "Logistic Regression")

autoplot(lr_auc)
```




```r
# 选出最好的惩罚函数在训练集建模
lr_wfl_final =
  lr_wfl %>%
  finalize_workflow(lr_best) %>%
  fit(data = df_train)



lr_train_probs = lr_wfl_final %>%
  predict(df_train, type = "prob") %>%
  bind_cols(df_train %>% dplyr::select(class)) %>%
  bind_cols(predict(lr_wfl_final, df_train))

# 混淆矩阵
conf_mat(lr_train_probs, class, .pred_class)
```

```
##           Truth
## Prediction   0   1
##          0 410 149
```

```
##          1 148 493
```

```r
# AUC
lr_train_AUC = roc_auc(lr_train_probs, class, .pred_0)
lr_train_AUC
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.827
```

```r
# 准确率
lr_train_accu = accuracy(lr_train_probs,class,.pred_class)
lr_train_accu
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.752
```

```r
# 召回率
lr_train_rec = recall(lr_train_probs,class,.pred_class)
lr_train_rec
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 recall  binary         0.735
```

```r
# 精确率
lr_train_prec = precision(lr_train_probs,class,.pred_class)
lr_train_prec
```

```
## # A tibble: 1 x 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 precision binary         0.733
```

```
lr_train_metric =
  bind_rows(lr_train_accu,lr_train_AUC,
            lr_train_rec,lr_train_prec) %>%
  select(.metric,.estimate)
```

```
# 在测试集预测并评估模型性能
lr_test_probs = lr_wfl_final %>%
  predict(df_test, type = "prob") %>%
  bind_cols(df_test %>% dplyr::select(class)) %>%
  bind_cols(predict(lr_wfl_final, df_test))

# 混淆矩阵
conf_mat(lr_test_probs, class, .pred_class)
```

```
##           Truth
## Prediction   0   1
##          0 138  58
##          1  48 155
```

```
# AUC
lr_test_AUC = roc_auc(lr_test_probs, class, .pred_0)
lr_test_AUC
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.807
```

```r
# 准确率
lr_test_accu = accuracy(lr_test_probs,class,.pred_class)
lr_test_accu
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.734
```

```r
# 召回率
lr_test_rec = recall(lr_test_probs,class,.pred_class)
lr_test_rec
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 recall  binary         0.742
```

```r
# 精确率
lr_test_prec = precision(lr_test_probs,class,.pred_class)
lr_test_prec
```

```
## # A tibble: 1 x 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 precision binary         0.704
```

```r
lr_test_metric =
  bind_rows(lr_test_accu,lr_test_AUC,
            lr_test_rec,lr_test_prec) %>%
  select(.metric,.estimate)

lr_metric = inner_join(lr_train_metric, lr_test_metric,
```

```
                       by = ".metric")

lr_ROC = roc_curve(lr_test_probs, class, .pred_0) %>%
  mutate(model = "Logistic Regression")
# autoplot(lr_ROC)
```

3.3 随机森林

```
# 定义 recipe
rf_recipe = df_train %>%
  mutate(class = as.factor(class)) %>%
  recipe(class ~ .)
```

```
# 定义模型
rf_model = rand_forest(mtry=tune(),min_n = tune(), trees = 1000)%>%
           set_mode("classification")%>%
           set_engine("ranger")
```

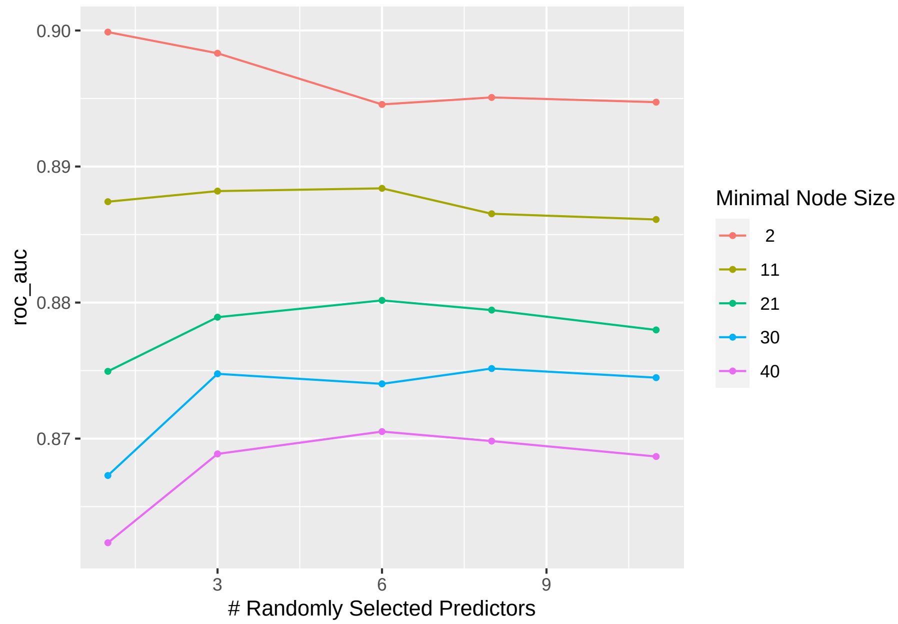
```
# 使用工作流将预处理和模型结合起来
rf_wfl =
  workflow() %>%
  add_recipe(rf_recipe) %>%
  add_model(rf_model)
rf_wfl
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: rand_forest()
##
## -- Preprocessor ----------------------------------------------------------------
## 0 Recipe Steps
##
```

```
## -- Model -------------------------------------------------------------------
## Random Forest Model Specification (classification)
##
## Main Arguments:
##   mtry = tune()
##   trees = 1000
##   min_n = tune()
##
## Computational engine: ranger
```

```
# 创建调节参数的格点集
rf_grid = grid_regular(finalize(mtry(), x = df_train[, -1]),
                       min_n(),
                       levels = 5)
```

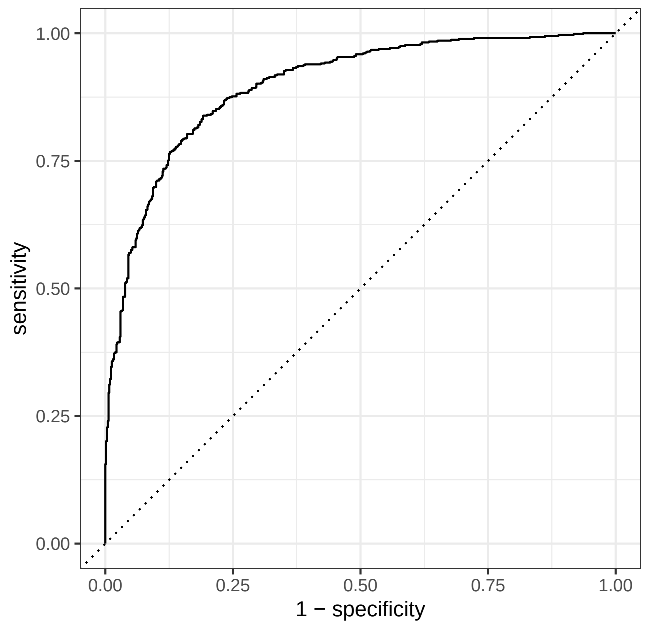
```
# 训练模型及调参
set.seed(2020)
rf_tune =
  rf_wfl %>%
  tune_grid(df_vfold,
            grid = rf_grid,
            control = control_grid(save_pred = TRUE),
            metrics = metric_set(roc_auc))

autoplot(rf_tune)
```




```r
# 根据交叉验证选出最好的超参数
rf_best = select_best(rf_tune)
rf_best
```

```
## # A tibble: 1 x 3
##    mtry min_n .config
##   <int> <int> <chr>
## 1     1     2 Preprocessor1_Model01
```

```r
# 交叉验证最好的模型 ROC 曲线
rf_auc =
  rf_tune %>%
  collect_predictions(parameters = rf_best) %>%
  roc_curve(class, .pred_0) %>%
  mutate(model = "Random Forest")

autoplot(rf_auc)
```




```
# 选出最好的惩罚函数在训练集建模
rf_wfl_final =
  rf_wfl %>%
  finalize_workflow(rf_best) %>%
  fit(data = df_train)

rf_train_probs = rf_wfl_final %>%
  predict(df_train, type = "prob") %>%
  bind_cols(df_train %>% dplyr::select(class)) %>%
  bind_cols(predict(rf_wfl_final, df_train))

# 混淆矩阵
conf_mat(rf_train_probs, class, .pred_class)
```

```
##           Truth
## Prediction   0   1
##          0 558   0
##          1   0 642
```

```r
# AUC
rf_train_AUC = roc_auc(rf_train_probs, class, .pred_0)
rf_train_AUC
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary             1
```

```r
# 准确率
rf_train_accu = accuracy(rf_train_probs,class,.pred_class)
rf_train_accu
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary             1
```

```r
# 召回率
rf_train_rec = recall(rf_train_probs,class,.pred_class)
rf_train_rec
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 recall  binary             1
```

```r
# 精确率
rf_train_prec = precision(rf_train_probs,class,.pred_class)
rf_train_prec
```

```
## # A tibble: 1 x 3
##   .metric   .estimator .estimate
```

```
##    <chr>     <chr>          <dbl>
## 1 precision binary             1
```

```
rf_train_metric =
  bind_rows(rf_train_accu,rf_train_AUC,
            rf_train_rec,rf_train_prec) %>%
  select(.metric,.estimate)
```

```
# 在测试集预测并评估模型性能
rf_test_probs = rf_wfl_final %>%
  predict(df_test, type = "prob") %>%
  bind_cols(df_test %>% dplyr::select(class)) %>%
  bind_cols(predict(rf_wfl_final, df_test))

# 混淆矩阵
conf_mat(rf_test_probs, class, .pred_class)
```

```
##             Truth
## Prediction   0   1
##          0 149  43
##          1  37 170
```

```
# AUC
rf_test_AUC = roc_auc(rf_test_probs, class, .pred_0)
rf_test_AUC
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.878
```

```
# 准确率
rf_test_accu = accuracy(rf_test_probs,class,.pred_class)
rf_test_accu
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.799
```

```
# 召回率
rf_test_rec = recall(rf_test_probs,class,.pred_class)
rf_test_rec
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 recall  binary         0.801
```

```
# 精确率
rf_test_prec = precision(rf_test_probs,class,.pred_class)
rf_test_prec
```

```
## # A tibble: 1 x 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 precision binary         0.776
```

```
rf_test_metric =
  bind_rows(rf_test_accu,rf_test_AUC,
            rf_test_rec,rf_test_prec) %>%
  select(.metric,.estimate)

rf_metric = inner_join(rf_train_metric, rf_test_metric,
                       by=".metric")

rf_ROC = roc_curve(rf_test_probs, class, .pred_0) %>%
  mutate(model = "Random Forest")
# autoplot(rf_ROC)
```

3.4 Boosted Trees

```
# 定义 recipe
C5_recipe = df_train %>%
  recipe(class ~ .)
```

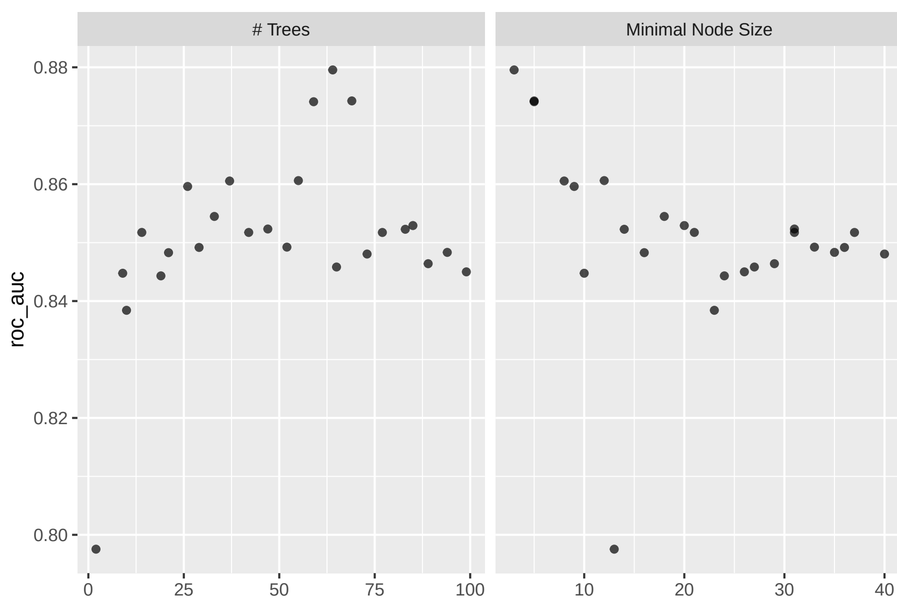
```
# 定义模型
C5_model <-
  boost_tree(trees = tune(), min_n = tune()) %>%
  set_engine("C5.0") %>%
  set_mode("classification")
```

```
# 定义工作流
C5_wfl =
  workflow() %>%
  add_recipe(C5_recipe) %>%
  add_model(C5_model)
C5_wfl
```

```
## == Workflow ===================================================================
## Preprocessor: Recipe
## Model: boost_tree()
##
## -- Preprocessor ---------------------------------------------------------------
## 0 Recipe Steps
##
## -- Model ----------------------------------------------------------------------
## Boosted Tree Model Specification (classification)
##
## Main Arguments:
##   trees = tune()
##   min_n = tune()
##
```

```
## Computational engine: C5.0
```

```r
set.seed(2020)
C5_tune =
  C5_wfl %>%
  tune_grid(df_vfold,
            grid = 25,
            control = control_grid(save_pred = TRUE),
            metrics = metric_set(roc_auc))


autoplot(C5_tune)
```

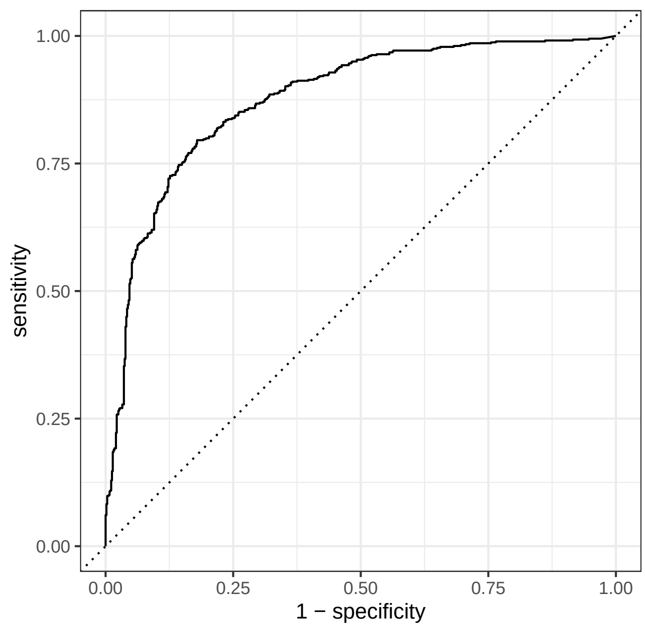



```r
# 根据交叉验证选出最好的超参数
C5_best = select_best(C5_tune)
C5_best
```

```
## # A tibble: 1 x 3
```

```
##   trees min_n .config
##   <int> <int> <chr>
## 1    64     3 Preprocessor1_Model05
```

```r
# 交叉验证最好的模型 ROC 曲线
C5_auc =
  C5_tune %>%
  collect_predictions(parameters = C5_best) %>%
  roc_curve(class, .pred_0) %>%
  mutate(model = "Boosted Trees")

autoplot(C5_auc)
```




```r
# 选出最好的惩罚函数在训练集建模
C5_wfl_final =
  C5_wfl %>%
  finalize_workflow(C5_best) %>%
  fit(data = df_train)
```

```r
C5_train_probs = C5_wfl_final %>%
  predict(df_train, type = "prob") %>%
  bind_cols(df_train %>% dplyr::select(class)) %>%
  bind_cols(predict(C5_wfl_final, df_train))

# 混淆矩阵
conf_mat(C5_train_probs, class, .pred_class)
```

```
##           Truth
## Prediction   0   1
##          0 525  26
##          1  33 616
```

```r
# AUC
C5_train_AUC = roc_auc(C5_train_probs, class, .pred_0)
C5_train_AUC
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.992
```

```r
# 准确率
C5_train_accu = accuracy(C5_train_probs,class,.pred_class)
C5_train_accu
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.951
```

```r
# 召回率
C5_train_rec = recall(C5_train_probs,class,.pred_class)
C5_train_rec
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 recall  binary         0.941
```

```r
# 精确率
C5_train_prec = precision(C5_train_probs,class,.pred_class)
C5_train_prec
```

```
## # A tibble: 1 x 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 precision binary         0.953
```

```r
C5_train_metric =
  bind_rows(C5_train_accu,C5_train_AUC,
            C5_train_rec,C5_train_prec) %>%
  select(.metric,.estimate)
```

```r
# 在测试集预测并评估模型性能
C5_test_probs = C5_wfl_final %>%
  predict(df_test, type = "prob") %>%
  bind_cols(df_test %>% dplyr::select(class)) %>%
  bind_cols(predict(C5_wfl_final, df_test))



# 混淆矩阵
conf_mat(C5_test_probs, class, .pred_class)
```

```
##            Truth
## Prediction   0   1
##          0 142  43
##          1  44 170
```

```
# AUC
C5_test_AUC = roc_auc(C5_test_probs, class, .pred_0)
C5_test_AUC
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.864
```

```
# 准确率
C5_test_accu = accuracy(C5_test_probs,class,.pred_class)
C5_test_accu
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.782
```

```
# 召回率
C5_test_rec = recall(C5_test_probs,class,.pred_class)
C5_test_rec
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 recall  binary         0.763
```

```r
# 精确率
C5_test_prec = precision(C5_test_probs,class,.pred_class)
C5_test_prec
```

```
## # A tibble: 1 x 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 precision binary         0.768
```

```r
C5_test_metric =
  bind_rows(C5_test_AUC,C5_test_accu,
            C5_test_rec,C5_test_prec) %>%
  select(.metric, .estimate)

C5_metric = inner_join(C5_train_metric,C5_test_metric,
                       by = ".metric")

C5_ROC = roc_curve(C5_test_probs, class, .pred_0) %>%
  mutate(model = "Boosted Trees")
# autoplot(C5_ROC)
```

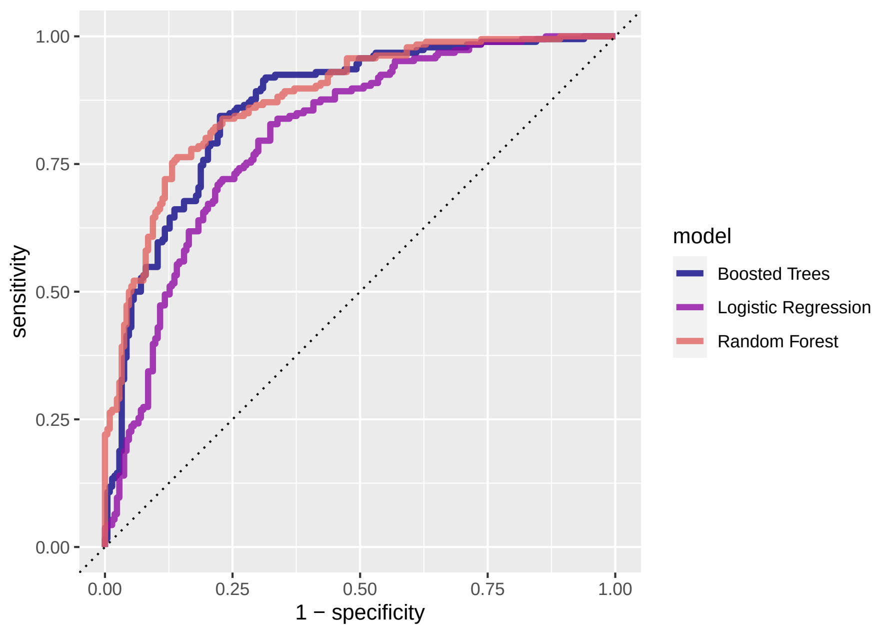
```r
# 三个机器学习模型在训练集、测试集的评估指标
bind_metric =
  inner_join(lr_metric, inner_join(rf_metric,C5_metric,
                                   by = ".metric"),
             by = ".metric") %>%
  dplyr::rename(metric = .metric,
                LR_train = .estimate.x,LR_test = .estimate.y,
                RF_train = .estimate.x.x,RF_test = .estimate.y.x,
                BT_train = .estimate.x.y,BT_test = .estimate.y.y)

knitr::kable(bind_metric)
```

metric	LR_train	LR_test	RF_train	RF_test	BT_train	BT_test
accuracy	0.7525000	0.7343358	1	0.7994987	0.9508333	0.7819549
roc_auc	0.8266143	0.8065273	1	0.8775809	0.9915196	0.8640517
recall	0.7347670	0.7419355	1	0.8010753	0.9408602	0.7634409
precision	0.7334526	0.7040816	1	0.7760417	0.9528131	0.7675676

```
# typology = tibble(
#   col_keys = c(".metric",".estimate.x",".estimate.y",".estimate.x.x",
#                ".estimate.y.x",".estimate.x.y",".estimate.y.y"),
#
#   type = c("metric",
#            "Logistic Regression","Logistic Regression",
#            "Random Forest","Random Forest",
#            "Boosted Tree","Boosted Tree"),
#
#   what = c("metric"," 训练集"," 测试集"," 训练集", " 测试集"," 训练集",
#            " 测试集")
#   )
#
# bind_metric %>%
#   flextable() %>%
#   set_header_df(mapping = typology, key = "col_keys") %>%
#   merge_h(part = "header") %>%
#   merge_v(part = "header") %>%
#   theme_booktabs() %>%
#   autofit() %>%
#   fix_border_issues()
```

从表中可以看出，随机森林表现在正确率、AUC、召回率、精确率最好，logistic 回归在训练集、测试集表现差不多，而随机森林和 boosted tree 在训练集和测试集差别稍大，但总体后两个模型要比 logistic 回归表现更好。

```
# 三个机器学习模型在测试集 ROC 曲线
bind_rows(lr_ROC,rf_ROC,C5_ROC) %>%
  ggplot(aes(x = 1 - specificity, y = sensitivity, col = model)) +
  geom_path(lwd = 1.5, alpha = 0.8) +
  geom_abline(lty = 3) +
  coord_equal() +
  scale_color_viridis_d(option = "plasma", end = .6)
```



ROC 曲线同样表明随机森林预测效果更好。

4 3、建立多分类模型预测红酒类别（v12=3，4，5）

```
dat_rw2 = red_wine
dat_rw2 = dat_rw2 %>%
  filter(quality<6) %>%
  glimpse()
```

```
## Rows: 744
## Columns: 12
## $ `fixed acidity`        <dbl> 7.4, 7.8, 7.8, 7.4, 7.4, 7.9, 7.5, 6.7, 7.5,...
## $ `volatile acidity`     <dbl> 0.700, 0.880, 0.760, 0.700, 0.660, 0.600, 0....
## $ `citric acid`          <dbl> 0.00, 0.00, 0.04, 0.00, 0.00, 0.06, 0.36, 0....
## $ `residual sugar`       <dbl> 1.9, 2.6, 2.3, 1.9, 1.8, 1.6, 6.1, 1.8, 6.1,...
## $ chlorides              <dbl> 0.076, 0.098, 0.092, 0.076, 0.075, 0.069, 0....
## $ `free sulfur dioxide`  <dbl> 11, 25, 15, 11, 13, 15, 17, 15, 17, 16, 9, 5...
## $ `total sulfur dioxide` <dbl> 34, 67, 54, 34, 40, 59, 102, 65, 102, 59, 29...
## $ density                <dbl> 0.9978, 0.9968, 0.9970, 0.9978, 0.9978, 0.99...
## $ pH                     <dbl> 3.51, 3.20, 3.26, 3.51, 3.51, 3.30, 3.35, 3....
## $ sulphates              <dbl> 0.56, 0.68, 0.65, 0.56, 0.56, 0.46, 0.80, 0....
## $ alcohol                <dbl> 9.4, 9.8, 9.8, 9.4, 9.4, 9.4, 10.5, 9.2, 10....
## $ quality                <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 4,...
```

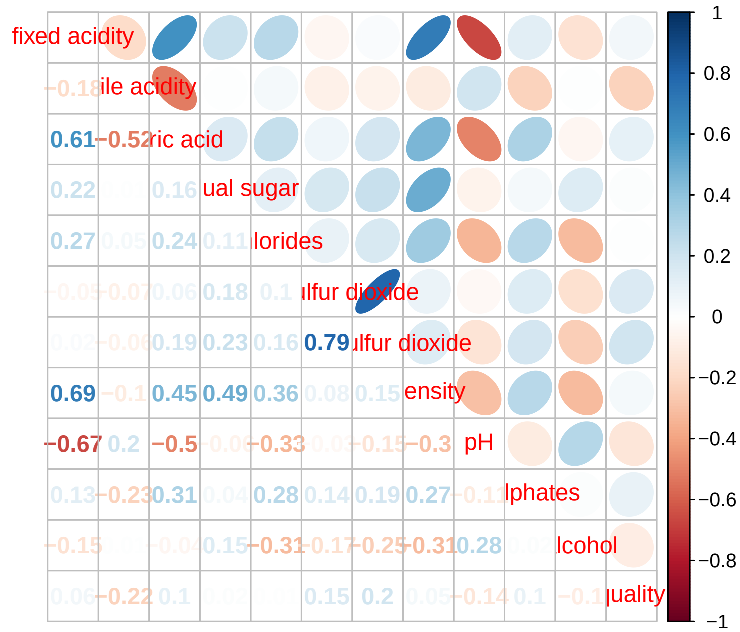
4.1 描述统计

```r
# 因变量分布情况
dat_rw2 %>%
  count(quality) %>%
  mutate(prop = n/sum(n))
```

```
## # A tibble: 3 x 3
##   quality     n   prop
##     <dbl> <int>  <dbl>
## 1       3    10 0.0134
## 2       4    53 0.0712
## 3       5   681 0.915
```

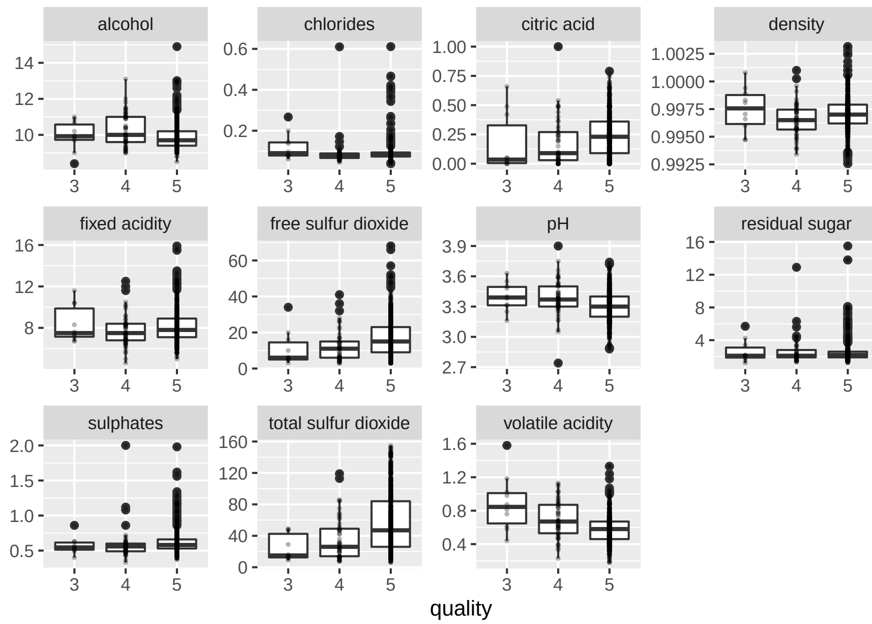
可以看到因变量的分布不平衡，即 quality=5 的占 90% 以上，quality=4 的占 7%，而 quality=3 的只有 1%。

```
# 相关系数矩阵
dat_rw2 %>%
  recipe(quality~.) %>%
  step_BoxCox(all_predictors()) %>%
  prep() %>%
  juice() %>%
  cor() %>%
  corrplot::corrplot.mixed(upper = "ellipse")
```




```
# 绘制各个变量箱线图
dat_rw2$quality = as.factor(dat_rw2$quality)
dat_rw2 %>%
  recipe( ~ .) %>%
  prep() %>%
  juice() %>%
  gather(Predictor, value, -quality)%>%
  ggplot(aes(x = quality, y = value)) +
  geom_boxplot() +
```

```
  geom_point(alpha = 0.3, cex = .5) +
  facet_wrap(~Predictor, scales = "free") +
  ylab("")
```



可以看到不同变量在不同红酒类别分布差别很大，quality=5 的类别最多，异常值也相对更多，有些变量中位数在 quality 三个类别基本相同，而柠檬酸 (citric acid)、密度 (density)、游离二氧化硫 (free sulfur dioxide)、总二氧化硫（total sulfur dioxide)、挥发性酸度 (volatile acidity) 在 quality 三个类别相对差别较大。

```
# 划分训练集、测试集
df = dat_rw2 %>%
  mutate(quality = as.factor(quality))

set.seed(2020)
df_split = initial_split(df, prop=0.8, strata = quality)

df_train = training(df_split)
df_test = testing(df_split)
```

```
# 训练集、测试集因变量分布
df_train %>%
  count(quality) %>%
  mutate(prop = n/sum(n))
```

```
## # A tibble: 3 x 3
##   quality     n   prop
##   <fct>   <int>  <dbl>
## 1 3           6 0.0101
## 2 4          39 0.0654
## 3 5         551 0.924
```

```
df_test %>%
  count(quality) %>%
  mutate(prop = n/sum(n))
```

```
## # A tibble: 3 x 3
##   quality     n   prop
##   <fct>   <int>  <dbl>
## 1 3           4 0.0270
## 2 4          14 0.0946
## 3 5         130 0.878
```

可以看到训练集类别严重不平衡，下面采用 smote 算法平衡数据

```
df_train = df_train %>%
  mutate(quality = as.factor(quality)) %>%
  recipe(quality ~ .) %>%
  step_smote(quality) %>% # 解决数据不平衡
  prep() %>%
  juice()

df_train %>%
```

```r
  count(quality) %>%
  mutate(prop = n/sum(n))
```

```
## # A tibble: 3 x 3
##   quality     n  prop
##   <fct>   <int> <dbl>
## 1 3         551 0.333
## 2 4         551 0.333
## 3 5         551 0.333
```

这时样本三类分别占 33.3%，类别达到平衡

```r
# 将训练数据划分为 10 折
df_vfold<-vfold_cv(df_train,v=10,repeats=1)
df_vfold
```

```
## #  10-fold cross-validation
## # A tibble: 10 x 2
##    splits             id
##    <list>             <chr>
##  1 <split [1.5K/166]> Fold01
##  2 <split [1.5K/166]> Fold02
##  3 <split [1.5K/166]> Fold03
##  4 <split [1.5K/165]> Fold04
##  5 <split [1.5K/165]> Fold05
##  6 <split [1.5K/165]> Fold06
##  7 <split [1.5K/165]> Fold07
##  8 <split [1.5K/165]> Fold08
##  9 <split [1.5K/165]> Fold09
## 10 <split [1.5K/165]> Fold10
```

下面分别采用随机森林、boosted trees、SVM 进行分类

4.2 随机森林

```
df_train_rec = df_train %>%
  recipe(quality ~ .)
```

```
# 定义模型
rf_model = rand_forest(mtry=tune(),min_n = tune(), trees = 1000)%>%
           set_mode("classification")%>%
           set_engine("ranger")
```

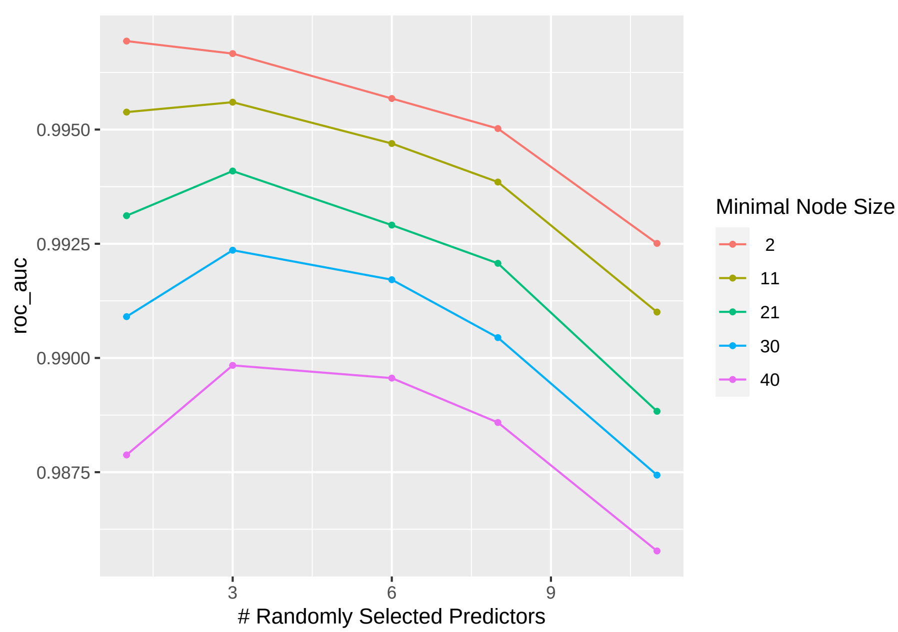
```
# 使用工作流将预处理和模型结合起来
rf_wfl =
  workflow() %>%
  add_recipe(df_train_rec) %>%
  add_model(rf_model)

rf_wfl
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: rand_forest()
##
## -- Preprocessor ----------------------------------------------------------------
## 0 Recipe Steps
##
## -- Model -----------------------------------------------------------------------
## Random Forest Model Specification (classification)
##
## Main Arguments:
##   mtry = tune()
##   trees = 1000
##   min_n = tune()
##
```

```
## Computational engine: ranger
```

```r
# 创建调节参数的格点集
rf_grid = grid_regular(finalize(mtry(), x = df_train[, -1]),
                       min_n(),
                       levels = 5)
```

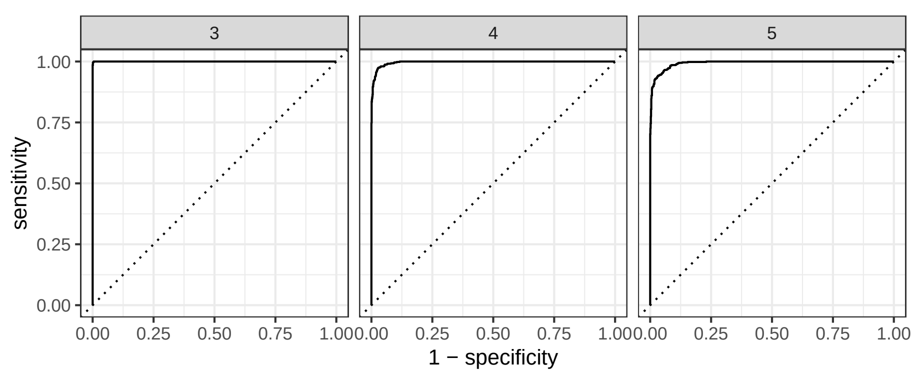
```r
# 训练模型及调参
set.seed(2020)
rf_tune =
  rf_wfl %>%
  tune_grid(df_vfold,
            grid = rf_grid,
            control = control_grid(save_pred = TRUE),
            metrics = metric_set(roc_auc))

autoplot(rf_tune)
```



```r
# 根据交叉验证选出最好的超参数
rf_best = select_best(rf_tune)
rf_best
```

```
## # A tibble: 1 x 3
##    mtry min_n .config
##   <int> <int> <chr>
## 1     1     2 Preprocessor1_Model01
```

```r
# 交叉验证最好的模型 ROC 曲线
rf_auc =
  rf_tune %>%
  collect_predictions(parameters = rf_best) %>%
  roc_curve(quality, .pred_3:.pred_5) %>%
  mutate(model = "Random Forest")

autoplot(rf_auc)
```



```
# 选出最好的惩罚函数在训练集建模
rf_wfl_final =
  rf_wfl %>%
  finalize_workflow(rf_best) %>%
  fit(data = df_train)

rf_train_probs = rf_wfl_final %>%
  predict(df_train, type = "prob") %>%
  bind_cols(df_train %>% dplyr::select(quality)) %>%
  bind_cols(predict(rf_wfl_final, df_train))

# 混淆矩阵
conf_mat(rf_train_probs, quality, .pred_class)
```

```
##             Truth
## Prediction    3   4   5
##            3 551   0   0
##            4   0 551   0
##            5   0   0 551
```

```
# AUC
rf_train_AUC = roc_auc(rf_train_probs, quality, .pred_3:.pred_5)
rf_train_AUC
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till          1
```

```
# 准确率
rf_train_accu = accuracy(rf_train_probs,quality,.pred_class)
rf_train_accu
```

```
## # A tibble: 1 x 3
```

```
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass         1
```

```
# 召回率
rf_train_rec = recall(rf_train_probs,quality,.pred_class)
rf_train_rec
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 recall  macro              1
```

```
# 精确率
rf_train_prec = precision(rf_train_probs,quality,.pred_class)
rf_train_prec
```

```
## # A tibble: 1 x 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 precision macro              1
```

```
rf_train_metric =
  bind_rows(rf_train_accu,rf_train_AUC,
            rf_train_rec,rf_train_prec) %>%
  select(.metric,.estimate)
```

```
# 在测试机预测并评估模型性能
rf_test_probs = rf_wfl_final %>%
  predict(df_test, type = "prob") %>%
  bind_cols(df_test %>% dplyr::select(quality)) %>%
  bind_cols(predict(rf_wfl_final, df_test))
```

```r
# 混淆矩阵
conf_mat(rf_test_probs, quality, .pred_class)
```

```
##           Truth
## Prediction   3   4   5
##          3   1   3   0
##          4   2   2  11
##          5   1   9 119
```

```r
# AUC
rf_test_AUC = roc_auc(rf_test_probs, quality, .pred_3:.pred_5)
rf_test_AUC
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.647
```

```r
# 准确率
rf_test_accu = accuracy(rf_test_probs,quality,.pred_class)
rf_test_accu
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.824
```

```r
# 召回率
rf_test_rec = recall(rf_test_probs,quality,.pred_class)
rf_test_rec
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
```

```
##    <chr>    <chr>          <dbl>
## 1 recall   macro          0.436
```

```
# 精确率
rf_test_prec = precision(rf_test_probs,quality,.pred_class)
rf_test_prec
```

```
## # A tibble: 1 x 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 precision macro          0.435
```

```
rf_test_metric =
  bind_rows(rf_test_accu,rf_test_AUC,
            rf_test_rec,rf_test_prec) %>%
  select(.metric,.estimate)

rf_metric = inner_join(rf_train_metric, rf_test_metric,
                       by=".metric")


rf_ROC = roc_curve(rf_test_probs, quality, .pred_3:.pred_5) %>%
  mutate(model = "Random Forest")
# autoplot(rf_ROC)
```

4.3 Boosted Trees

```
df_train_rec = df_train %>%
  recipe(quality ~ .)
```

```
# 定义模型
C5_model <-
```

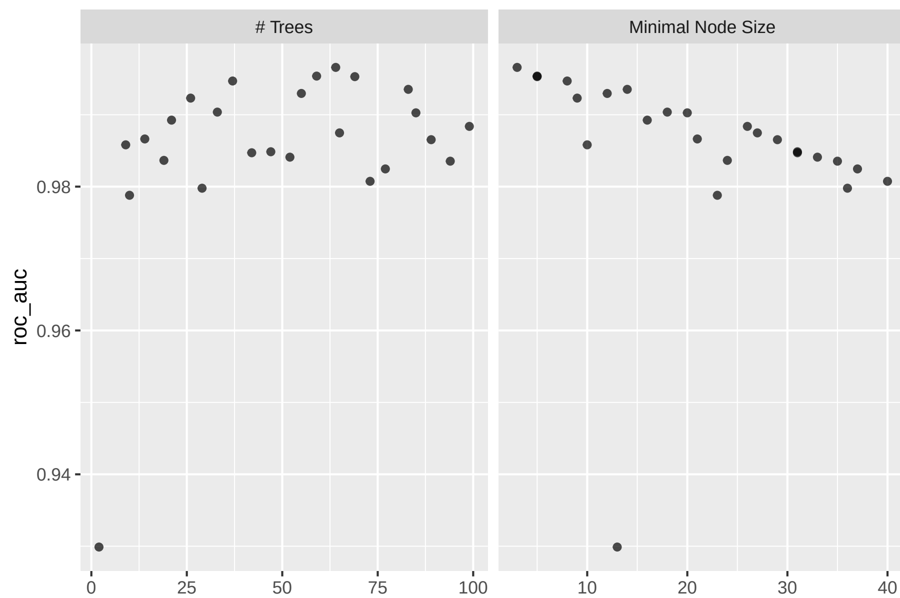
```r
  boost_tree(trees = tune(), min_n = tune()) %>%
  set_engine("C5.0") %>%
  set_mode("classification")
```

```r
# 定义工作流
C5_wfl =
  workflow() %>%
  add_recipe(df_train_rec) %>%
  add_model(C5_model)
C5_wfl
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: boost_tree()
##
## -- Preprocessor ----------------------------------------------------------------
## 0 Recipe Steps
##
## -- Model -----------------------------------------------------------------------
## Boosted Tree Model Specification (classification)
##
## Main Arguments:
##   trees = tune()
##   min_n = tune()
##
## Computational engine: C5.0
```

```r
set.seed(2020)
C5_tune =
  C5_wfl %>%
  tune_grid(df_vfold,
            grid = 25,
            control = control_grid(save_pred = TRUE),
```

```
            metrics = metric_set(roc_auc))


autoplot(C5_tune)
```

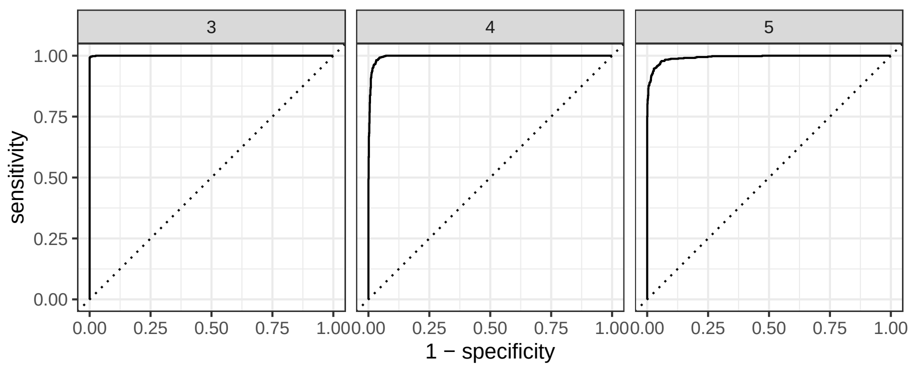



```
# 根据交叉验证选出最好的超参数
C5_best = select_best(C5_tune)
C5_best
```

```
## # A tibble: 1 x 3
##   trees min_n .config
##   <int> <int> <chr>
## 1    64     3 Preprocessor1_Model05
```

```
# 交叉验证最好的模型 ROC 曲线
C5_auc =
  C5_tune %>%
  collect_predictions(parameters = C5_best) %>%
```

```
  roc_curve(quality, .pred_3:.pred_5) %>%
  mutate(model = "Boosted Trees")

autoplot(C5_auc)
```




```
# 选出最好的惩罚函数在训练集建模
C5_wfl_final =
  C5_wfl %>%
  finalize_workflow(C5_best) %>%
  fit(data = df_train)

C5_train_probs = C5_wfl_final %>%
  predict(df_train, type = "prob") %>%
  bind_cols(df_train %>% dplyr::select(quality)) %>%
  bind_cols(predict(C5_wfl_final, df_train))

# 混淆矩阵
conf_mat(C5_train_probs, quality, .pred_class)
```

```
##            Truth
## Prediction   3   4   5
##          3 551   0   0
##          4   0 551   0
##          5   0   0 551
```

```
# AUC
C5_train_AUC = roc_auc(C5_train_probs, quality, .pred_3:.pred_5)
C5_train_AUC
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till          1
```

```
# 准确率
C5_train_accu = accuracy(C5_train_probs,quality,.pred_class)
C5_train_accu
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass         1
```

```
# 召回率
C5_train_rec = recall(C5_train_probs,quality,.pred_class)
C5_train_rec
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 recall  macro              1
```

```
# 精确率
C5_train_prec = precision(C5_train_probs,quality,.pred_class)
C5_train_prec
```

```
## # A tibble: 1 x 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 precision macro              1
```

```
C5_train_metric =
  bind_rows(C5_train_accu,C5_train_AUC,
            C5_train_rec,C5_train_prec) %>%
  select(.metric,.estimate)
```

```
# 在测试集预测并评估模型性能
C5_test_probs = C5_wfl_final %>%
  predict(df_test, type = "prob") %>%
  bind_cols(df_test %>% dplyr::select(quality)) %>%
  bind_cols(predict(C5_wfl_final, df_test))

# 混淆矩阵
conf_mat(C5_test_probs, quality, .pred_class)
```

```
##           Truth
## Prediction   3   4   5
##          3   0   3   1
##          4   2   3  15
##          5   2   8 114
```

```
# AUC
C5_test_AUC = roc_auc(C5_test_probs, quality, .pred_3:.pred_5)
C5_test_AUC
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.660
```

```
# 准确率
C5_test_accu = accuracy(C5_test_probs,quality,.pred_class)
C5_test_accu
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.791
```

```
# 召回率
C5_test_rec = recall(C5_test_probs,quality,.pred_class)
C5_test_rec
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 recall  macro          0.364
```

```
# 精确率
C5_test_prec = precision(C5_test_probs,quality,.pred_class)
C5_test_prec
```

```
## # A tibble: 1 x 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 precision macro          0.356
```

```
C5_test_metric =
  bind_rows(C5_test_AUC,C5_test_accu,
            C5_test_rec,C5_test_prec) %>%
  select(.metric, .estimate)

C5_metric = inner_join(C5_train_metric,C5_test_metric,
                       by = ".metric")



C5_ROC = roc_curve(C5_test_probs, quality, .pred_3:.pred_5) %>%
  mutate(model = "Boosted Trees")
# autoplot(C5_ROC)
```

4.4 支持向量机

```
# 预处理
svm_rec =
  recipe(quality ~ ., data = df_train) %>%
  step_BoxCox(all_predictors())%>%
  step_normalize(all_predictors())

svm_prep = prep(svm_rec)

# 测试集预处理
test_normalized = bake(svm_prep, new_data = df_test, all_predictors())
```

```
# 定义模型
set.seed(2020)
svm_model =
  svm_rbf(cost = tune(),rbf_sigma = tune()) %>%
  set_mode("classification") %>%
```

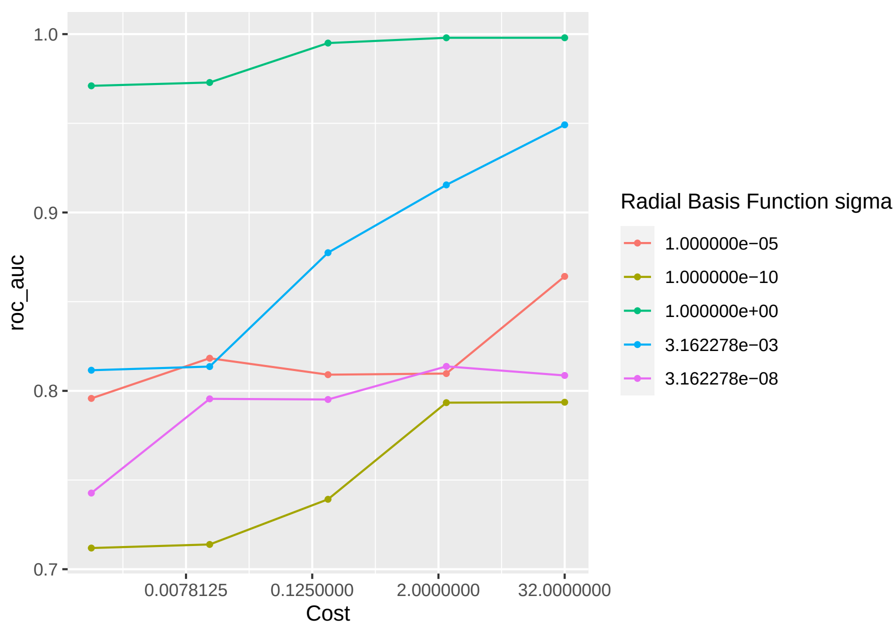
```
  set_engine("kernlab")
```

```
# 定义工作流
svm_wfl =
  workflow() %>%
  add_recipe(svm_rec) %>%
  add_model(svm_model)
svm_wfl
```

```
## == Workflow ===================================================================
## Preprocessor: Recipe
## Model: svm_rbf()
##
## -- Preprocessor ---------------------------------------------------------------
## 2 Recipe Steps
##
## * step_BoxCox()
## * step_normalize()
##
## -- Model ----------------------------------------------------------------------
## Radial Basis Function Support Vector Machine Specification (classification)
##
## Main Arguments:
##   cost = tune()
##   rbf_sigma = tune()
##
## Computational engine: kernlab
```

```
# 创建调节参数的格点集
svm_grid = grid_regular(cost(),
                        rbf_sigma(),
                        levels = 5)
```

```
set.seed(2020)
svm_tune =
  svm_wfl %>%
  tune_grid(df_vfold,
            grid = svm_grid,
            control = control_grid(save_pred = TRUE),
            metrics = metric_set(roc_auc))


autoplot(svm_tune)
```

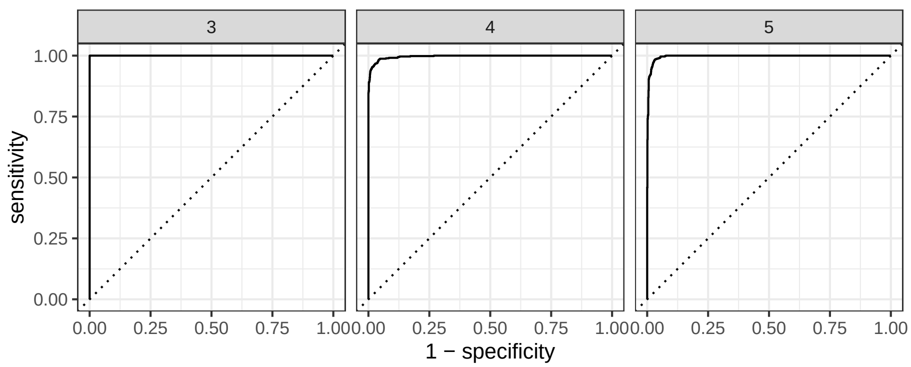



```
# 根据交叉验证选出最好的超参数
svm_best = select_best(svm_tune)
svm_best
```

```
## # A tibble: 1 x 3
##     cost rbf_sigma .config
##    <dbl>     <dbl> <chr>
```

```
## 1     32          1 Preprocessor1_Model25
```

```
# 交叉验证最好的模型 ROC 曲线
svm_auc =
  svm_tune %>%
  collect_predictions(parameters = svm_best) %>%
  roc_curve(quality, .pred_3:.pred_5) %>%
  mutate(model = "Boosted Trees")

autoplot(svm_auc)
```




```
# 选出最好的惩罚函数在训练集建模
svm_wfl_final =
  svm_wfl %>%
  finalize_workflow(svm_best) %>%
  fit(data = df_train)

svm_train_probs = svm_wfl_final %>%
```

```
  predict(df_train, type = "prob") %>%
  bind_cols(df_train %>% dplyr::select(quality)) %>%
  bind_cols(predict(C5_wfl_final, df_train))

# 混淆矩阵
conf_mat(svm_train_probs, quality, .pred_class)
```

```
##           Truth
## Prediction   3   4   5
##          3 551   0   0
##          4   0 551   0
##          5   0   0 551
```

```
# AUC
svm_train_AUC = roc_auc(svm_train_probs, quality, .pred_3:.pred_5)
svm_train_AUC
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till          1
```

```
# 准确率
svm_train_accu = accuracy(svm_train_probs,quality,.pred_class)
svm_train_accu
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass         1
```

```
# 召回率
svm_train_rec = recall(svm_train_probs,quality,.pred_class)
svm_train_rec
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 recall  macro              1
```

```
# 精确率
svm_train_prec = precision(svm_train_probs,quality,.pred_class)
svm_train_prec
```

```
## # A tibble: 1 x 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 precision macro              1
```

```
svm_train_metric =
  bind_rows(svm_train_accu,svm_train_AUC,
            svm_train_rec,svm_train_prec) %>%
  select(.metric,.estimate)
```

```
# 在测试集预测并评估模型性能
svm_test_probs = svm_wfl_final %>%
  predict(df_test, type = "prob") %>%
  bind_cols(df_test %>% dplyr::select(quality)) %>%
  bind_cols(predict(svm_wfl_final, df_test))

# 混淆矩阵
conf_mat(C5_test_probs, quality, .pred_class)
```

```
##          Truth
```

```
## Prediction   3   4   5
##          3   0   3   1
##          4   2   3  15
##          5   2   8 114
```

```r
# AUC
svm_test_AUC = roc_auc(svm_test_probs, quality, .pred_3:.pred_5)
svm_test_AUC
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.670
```

```r
# 准确率
svm_test_accu = accuracy(svm_test_probs,quality,.pred_class)
svm_test_accu
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.872
```

```r
# 召回率
svm_test_rec = recall(svm_test_probs,quality,.pred_class)
svm_test_rec
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 recall  macro          0.352
```

```
# 精确率
svm_test_prec = precision(svm_test_probs,quality,.pred_class)
```

```
## Warning: While computing multiclass `precision()`, some levels had no predicted even
## Precision is undefined in this case, and those levels will be removed from the avera
## Note that the following number of true events actually occured for each problematic
## '3': 4
```

```
svm_test_prec
```

```
## # A tibble: 1 x 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 precision macro          0.569
```

```
svm_test_metric =
  bind_rows(svm_test_AUC,svm_test_accu,
            svm_test_rec,svm_test_prec) %>%
  select(.metric, .estimate)

svm_metric = inner_join(svm_train_metric,svm_test_metric,
                        by = ".metric")



svm_ROC = roc_curve(svm_test_probs, quality, .pred_3:.pred_5) %>%
  mutate(model = "SVM")
# autoplot(svm_ROC)
```

```
# 三个机器学习模型在训练集、测试集的评估指标
bind_metric =
  inner_join(rf_metric, inner_join(C5_metric,svm_metric,
                                   by = ".metric"),
             by = ".metric") %>%
```

```
  dplyr::rename(metric = .metric,
                RF_train= .estimate.x, RF_test = .estimate.y,
                BT_train= .estimate.x.x, BT_test = .estimate.y.x,
                SVM_train = .estimate.x.y, SVM_test = .estimate.y.y)

knitr::kable(bind_metric)
```

metric	RF_train	RF_test	BT_train	BT_test	SVM_train	SVM_test
accuracy	1	0.8243243	1	0.7905405	1	0.8716216
roc_auc	1	0.6468178	1	0.6600504	1	0.6697344
recall	1	0.4360806	1	0.3637363	1	0.3520147
precision	1	0.4352713	1	0.3564516	1	0.5694444

```
# typology = tibble(
#   col_keys = c(".metric",".estimate.x",".estimate.y",".estimate.x.x",
#                ".estimate.y.x",".estimate.x.y",".estimate.y.y"),
#
#   type = c("metric",
#            "Random Forest","Random Forest",
#            "Boosted Tree","Boosted Tree",
#            "SVM","SVM"),
#
#   what = c("metric"," 训练集"," 测试集"," 训练集", " 测试集"," 训练集",
#            " 测试集")
#   )
#
# bind_metric %>%
#   flextable() %>%
#   set_header_df(mapping = typology, key = "col_keys") %>%
#   merge_h(part = "header") %>%
#   merge_v(part = "header") %>%
#   theme_booktabs() %>%
```

```
#   autofit() %>%
#   fix_border_issues()
```

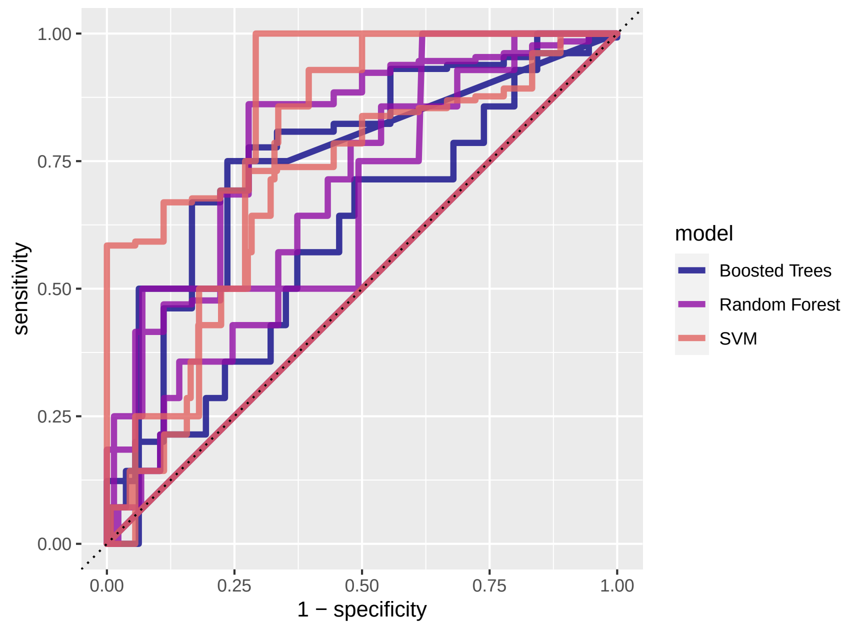
因为有一个类别只占 1%，一个类别不到 10%，可见，经过 smote 算法平衡训练集数据后，预测结果还是并不是很好。经过交叉验证后选出相对较优的超参数后，三个模型依然呈现过拟合的状态，Boosted Tree 的训练采用的是算法自己寻找 25 个参数组合，而且超参数过多，可能并没有找到相对较优的参数；Random Forest 和 SVM 给出超参数格点集，结果也呈现过拟合的状态。

正如描述统计所显示，与因变量相关的自变量并不多，而且相关性相对较弱，这是预测效果差一方面的原因。

从表中可以看出，三个模型的 AUC 相差不大，SVM 的 AUC 最大（0.67），其次 Boosted Tree（0.66），然后就是 Random Forest（0.65），如果注重召回率，随机森林相对较好，但是与其他模型差别不大，总体 SVM 效果最好（AUC 最大，且召回率虽然最低，但与其他两个模型相差不大，而准确率最高）。

最后给出三个模型的 ROC 曲线

```
bind_rows(rf_ROC,C5_ROC,svm_ROC) %>%
  ggplot(aes(x = 1 - specificity, y = sensitivity, col = model)) +
  geom_path(lwd = 1.5, alpha = 0.8) +
  geom_abline(lty = 3) +
  coord_equal() +
  scale_color_viridis_d(option = "plasma", end = .6)
```



1.00
0.75
0.50
0.25
0.00
sensitivity
0.00
0.25
0.50
0.75
1.00
1 − specificity
model
Boosted Trees
Random Forest
SVM