Decision Tree Implementation

Import dataset and check the data dimension

Firstly, import the new data generated before, with a dataset size of 14347 and a dimension of 48.

```
# import new dataset
data <- read.csv("data/new_dataset.csv")
```

```
# check data dims
dim(data)
```

```
## [1] 14347    48
```

Preprocessing

There is one more pre-processing step that I had done, which is to combine the columns named "reserved_room_type" and "assigned_room_type" into one column called "is_same_type". If they are of the same type, then this row will be recorded as 1 in "is_same_type", otherwise be recorded as 0. According to the experiment, the final accuracy improved around 1.5%.

```
# combine 2 columns
data$is_same_type <- ifelse(data$reserved_room_type == data$assigned_room_type, 1, 0)
data <- subset(data, select = -c(reserved_room_type, assigned_room_type))
```

Build decision tree

Devide the dataset into train and test set

The first step is to divide the dataset into train and test set, with the division of 7:3, and the model I chose for individual part is decision tree. In this part, different model parameters are generated, which could be a convenient for checking parameters in parameter tuning part.

```
## import necessary libraries
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
library(mlr)
```

```
## Loading required package: ParamHelpers
```

```
## Warning message: 'mlr' is in 'maintenance-only' mode since July 2019.
## Future development will only happen in 'mlr3'
## (<https://mlr3.mlr-org.com>). Due to the focus on 'mlr3' there might be
## uncaught bugs meanwhile in {mlr} - please consider switching.


##
## Attaching package: 'mlr'


## The following object is masked from 'package:caret':
##
##     train
```

```
## data partition into train-test with 7:3
data$is_canceled <- factor(data$is_canceled)

idx <- sample(c(TRUE, FALSE), nrow(data), replace=TRUE, prob=c(0.7,0.3))
train <- data[idx, ]
test <- data[!idx, ]
```

build the decision tree model

```
## define the trainning and testing tasks by using mlr
train.task <- makeClassifTask(
  data=train,
  target="is_canceled")
test.task <- makeClassifTask(
  data = test,
  target = "is_canceled")

# Create a classification tree
lrn <- makeLearner("classif.rpart")

# get information about hyper-parameters
getParamSet(lrn)
```

```
##                    Type len  Def   Constr Req Tunable Trafo
## minsplit        integer   -   20 1 to Inf   -    TRUE     -
## minbucket       integer   -    - 1 to Inf   -    TRUE     -
## cp              numeric   - 0.01   0 to 1   -    TRUE     -
## maxcompete      integer   -    4 0 to Inf   -    TRUE     -
## maxsurrogate    integer   -    5 0 to Inf   -    TRUE     -
## usesurrogate   discrete   -    2    0,1,2   -    TRUE     -
## surrogatestyle discrete   -    0      0,1   -    TRUE     -
## maxdepth        integer   -   30  1 to 30   -    TRUE     -
## xval            integer   -   10 0 to Inf   -   FALSE     -
## parms           untyped   -    -        -   -    TRUE     -
```

Train the initial model

```r
# train the decision tree model
model <- train(lrn, train.task)
```

Use K-fold for validation

In task 3-2, the cross validation with 5 k-folds is built here, the accuracy for test dataset is recorded below, with the final aggregated result.

```r
# set the cross validation with 5 folds
cv_task <- makeResampleDesc("CV", iters = 5)

# use the setted k-fold for validation
cv_results <- resample(learner = lrn,
                       task = train.task,
                       resampling = cv_task,
                       measures = list(acc))
```

```
## Resampling: cross-validation

## Measures:             acc

## [Resample] iter 1:    0.7921236

## [Resample] iter 2:    0.7896311

## [Resample] iter 3:    0.7820449

## [Resample] iter 4:    0.7795511

## [Resample] iter 5:    0.7856431

##

## Aggregated Result: acc.test.mean=0.7857988

##
```

```r
cv_results
```

```
## Resample Result
## Task: train
## Learner: classif.rpart
## Aggr perf: acc.test.mean=0.7857988
## Runtime: 0.324528
```

Parameter tuning

Parameter tuning by grid search

In decision tree, there are many hyper-parameters, such as max-depth, min-criterion and so on. However, I only tuned 3 hyper-parameters in this task that are especially important for decision tree, protecting them from over-fitting and controlling the model complexity as well, namely min-split, min-bucket and cp. The importance and meaning of these hyper-parameters will be introduced below.

1. minsplit: This represents the minimum amount of observation data on a node. when the data amount is greater than or equal to this number, further splitting will occur, while nodes below this number will not have further split. A smaller minsplit can cause the data to be finely divided, resulting in more complex trees, but it might lead to overfitting. Accordingly, a larger minsplit will limit the growth of the tree to a certain degree, making the model simpler and may have better generalization ability on unprecedented data.
2. minbucket: This represents the minimum amount of observation data that a leaf node should contain if it wants to split. Similarly, this also helps to control the size of leaf nodes and prevent overfitting of the model.
3. cp: This represents the Complexity Parameter, which is commonly used to control the degree of pruning of trees. By pruning some branches to make the tree model simpler, it helps to reduce the complexity of the model and improve its generalization ability. Usually, the smaller the value of cp, the easier it is for the tree to prune, resulting in a simpler tree structure that helps prevent overfitting.

```
# set parameters by using grid search
ps <- makeParamSet(
makeDiscreteParam("minsplit", values = seq(1, 10, 2)),
makeDiscreteParam("minbucket", values = seq(4, 20, 2)),
makeDiscreteParam("cp", values = seq(0.0001, 0.0015, 0.0001))
)
```

start parameter tunning with 5 fold

```
# set up cross validation
control.grid <- makeTuneControlGrid()

# tune the parameters
set.seed(123)
tuned.model <- tuneParams(lrn, task = train.task,
                          resampling = cv_task,
                          control = control.grid,
                          par.set = ps,
                          measures = list(acc),
                          show.info = FALSE)
```

show the best result

```
tuned.model$x
```

```
## $minsplit
## [1] 1
##
## $minbucket
## [1] 6
##
## $cp
## [1] 0.0011
```

show the table of results with different parameters

show the different model performance under different parameters, which are around 600 different models with corresponding accuracy.

```
# show the table of grid search result
library(data.table)
opt.grid <- as.data.table(tuned.model$opt.path)
opt.grid
```

```
##      minsplit minbucket     cp acc.test.mean dob eol error.message exec.time
##   1:        1         4  1e-04     0.8006577   1  NA          <NA>     0.544
##   2:        3         4  1e-04     0.8006577   2  NA          <NA>     0.457
##   3:        5         4  1e-04     0.8006577   3  NA          <NA>     0.456
##   4:        7         4  1e-04     0.8006577   4  NA          <NA>     0.455
##   5:        9         4  1e-04     0.8006576   5  NA          <NA>     0.545
##  ---
## 671:        1        20 0.0015     0.8210997 671  NA          <NA>     0.371
## 672:        3        20 0.0015     0.8210997 672  NA          <NA>     0.453
## 673:        5        20 0.0015     0.8210997 673  NA          <NA>     0.365
## 674:        7        20 0.0015     0.8210997 674  NA          <NA>     0.368
## 675:        9        20 0.0015     0.8210997 675  NA          <NA>     0.369
```

In order to show the specific change of accuracy clearly, a specific set of parameters are chosen here, which are under the best minbucket and minsplit. Therefore, the model performance could be shown here with the change of cp.

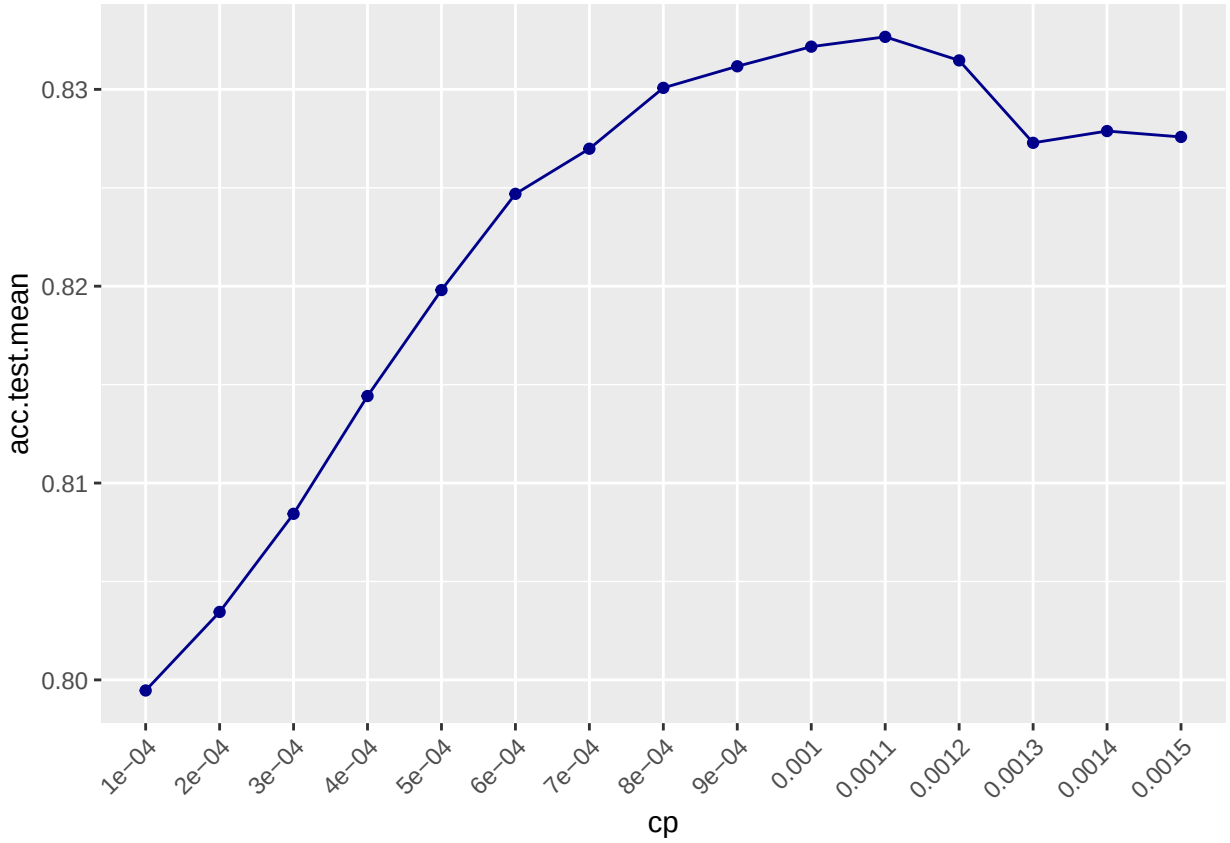
```
# show the result with a specific minsplit and minbucket
optimal.cp <- opt.grid[minbucket == tuned.model$x$minbucket
        & minsplit == tuned.model$x$minsplit]
print(optimal.cp)
```

```
##     minsplit minbucket     cp acc.test.mean dob eol error.message exec.time
##  1:        1         6  1e-04     0.7994625   6  NA          <NA>     0.439
##  2:        1         6  2e-04     0.8034516  51  NA          <NA>     0.437
##  3:        1         6  3e-04     0.8084369  96  NA          <NA>     0.436
##  4:        1         6  4e-04     0.8144198 141  NA          <NA>     0.427
##  5:        1         6  5e-04     0.8198043 186  NA          <NA>     0.424
##  6:        1         6  6e-04     0.8246907 231  NA          <NA>     0.419
##  7:        1         6  7e-04     0.8269844 276  NA          <NA>     0.418
##  8:        1         6  8e-04     0.8300756 321  NA          <NA>     0.413
##  9:        1         6  9e-04     0.8311716 366  NA          <NA>     0.407
## 10:        1         6  0.001     0.8321691 411  NA          <NA>     0.486
```

```
## 11:         1         6 0.0011     0.8326677 456  NA          <NA>     0.406
## 12:         1         6 0.0012     0.8314710 501  NA          <NA>     0.398
## 13:         1         6 0.0013     0.8272834 546  NA          <NA>     0.393
## 14:         1         6 0.0014     0.8278820 591  NA          <NA>     0.391
## 15:         1         6 0.0015     0.8275830 636  NA          <NA>     0.394
```

Moreover, the plots of model performance with the change of cp, but under the certain minsplit and minbucket could also be seen.

```
# plot the acc with different cp under certain minbucket and minsplit
ggplot(data=as.data.frame(optimal.cp),
       aes(x=cp, y=acc.test.mean, group=1)) +
  geom_line(colour = "darkblue")+
  geom_point(colour = "darkblue")+
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



Report the best model

Use the best model to show the confusion matrix

The confusion matrix can be used as one of the indicators for evaluating a model, as it allows for the calculation of Accuracy, Precision, Recall, F1 score. These results are presented in the following code.

```
# get best parameter and train the model
tree.tuned <- setHyperPars(learner = lrn, par.vals = tuned.model$x)

best.model <- train(tree.tuned, task = train.task)

# get prediction result and confusion matrix
start_time <- Sys.time()
tree.pred <- predict(best.model, test.task)
end_time <- Sys.time()
predictions  <- tree.pred$data
confusion.matrix <- table(predictions$response, predictions$truth)

# show the confusion matrix
print(confusion.matrix)
```

```
##
##        0    1
##   0 2725  486
##   1  237  871
```

get each evaluating indicator under the best performance

```
# calculate evaluating indicator
TN <- confusion.matrix[1, 1]
FP <- confusion.matrix[1, 2]
FN <- confusion.matrix[2, 1]
TP <- confusion.matrix[2, 2]

acc <- (TP + TN) / sum(confusion.matrix)
precision <- TP / (TP + FP)
recall <- TP / (TP + FN)
F1 <- 2 * (precision * recall) / (precision + recall)

# show the evaluation results of the best model
cat("Accuracy, ", acc)
```

```
## Accuracy,  0.8326001
```

```
cat("Presion, :", precision)
```

```
## Presion, : 0.641857
```

```
cat("Recall, :", recall)
```

```
## Recall, : 0.7861011
```

```
cat("F1 score, :", F1)
```

```
## F1 score, : 0.7066937
```

```
cat("Run time, :", round(end_time - start_time, digits=3))
```

```
## Run time, : 0.006
```

Model Explanation

Decision tree is a basic model for classification, and the final output is the categories contained in train dataset. At each node, the decision tree divides the dataset by selecting the best features. For decision trees used for classification tasks, commonly used partitioning standards including Information gain, etc, which will recursively select the best features and partition the dataset until the termination conditions are met at each node. The termination condition can be to reach the maximum depth and meet the requirements of min split number, etc. When stopping partitioning, each leaf node has a category prediction value.

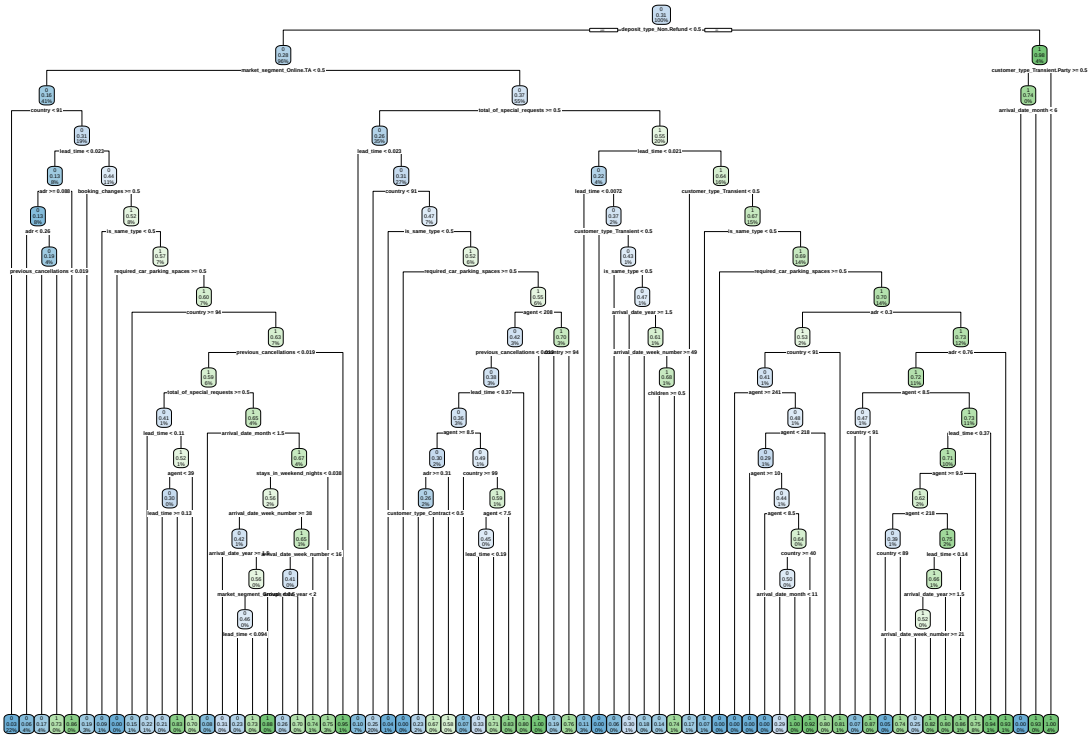
The i will use one example from the given dataset to explain how decision tree works. The following code can plot how the best model mentioned above works, which is to visualize the best decision tree model trained above.

```
# draw the plots of decision tree with one example
library(rpart.plot)
```

```
## Loading required package: rpart
```

```
tree.rpart <- getLearnerModel(best.model)
rpart.plot(tree.rpart, roundint = FALSE)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



Firstly, we should select one row from the dataset, the detailed information of this data is shown below.

First Node: It could be observed that the first node for classification is deposite_type_None.Refund, those that are less than 0.5 should go left while those more than 0.5 should go left. The attribute deposite_type_None.Refund of selected dataset is 1, therefore it should go right for further classification.

Second Node: The next classification standard is market_segment_Online.TA. The selected data has 0 for this attribute, so according to the decision tree, it should go left for further division.

Third Node: The following criteria for division is country, it could be observed that its country id is 92, so according to the plot, it should still go left.

Fourth Node: The following criteria for division is lead_time, which is 0.0063593 in the selected row, therefore, according to the ploted tree, it belongs to the group whose lead_time is less than 0.026, therefore, it should go left for further division.

Fifth Node: The next division criteria is adr, for those that are more than 0.088 should go left while for those that are less than 0.088 should go right. The selected data for this attribute is 0.2409639, therefore it should go left and now can be classified as 0, which is will not be canceled.

However, if its adr is less than 0.088, then it should go right, where lied other division criteria such as is_same_type, booking changes and so on.

```
# select one example from testset
test_sample <- data[30,]
print(t(test_sample))
```

```
##                              30
## hotel_City.Hotel             "1"
```

```
## hotel_Resort.Hotel             "0"
## is_canceled                    "0"
## lead_time                      "0.0063593"
## arrival_date_year              "2"
## arrival_date_month             "8"
## arrival_date_week_number       "35"
## arrival_date_day_of_month      "25"
## stays_in_weekend_nights        "0"
## stays_in_week_nights           "0.125"
## adults                         "2"
## children                       "0"
## babies                         "0"
## meal_BB                        "1"
## meal_FB                        "0"
## meal_HB                        "0"
## meal_SC                        "0"
## country                        "92"
## market_segment_Aviation        "0"
## market_segment_Complementary   "0"
## market_segment_Corporate       "0"
## market_segment_Direct          "0"
## market_segment_Groups          "1"
## market_segment_Offline.TA.TO   "0"
## market_segment_Online.TA       "0"
## distribution_channel_Corporate "0"
## distribution_channel_Direct    "0"
## distribution_channel_GDS       "0"
## distribution_channel_TA.TO     "1"
## is_repeated_guest              "0"
## previous_cancellations         "0"
## previous_bookings_not_canceled "0"
## booking_changes                "0"
## deposit_type_No.Deposit        "1"
## deposit_type_Non.Refund        "0"
## deposit_type_Refundable        "0"
## agent                          "1"
## company                        "0"
## days_in_waiting_list           "0"
## customer_type_Contract         "0"
## customer_type_Group            "0"
## customer_type_Transient        "0"
## customer_type_Transient.Party  "1"
## adr                            "0.2409639"
## required_car_parking_spaces    "0"
## total_of_special_requests      "0"
## is_same_type                   "1"
```

Check the prediction result

We have already classified the selected data according to the plotted tree, which is classified as 0. Therefore, we could verify the result by using the decision tree model. It could be observed that the predicted label is 0, while the real label is also 0. Therefore, the prediction is accurate.

```r
# use model to show the result
test_sample$is_canceled <- factor(test_sample$is_canceled)
sample.task <- makeClassifTask(
  data = test_sample,
  target = "is_canceled")
predict_sample <- predict(best.model, sample.task)
print(predict_sample)
```

```
## Prediction: 1 observations
## predict.type: response
## threshold:
## time: 0.00
##    id truth response
## 30  1     0        0
```

Model Comparison

In addition to the decision tree I chose, the other two models our group chose were SVM and logistic regression In order to have our model in the same configuration environment, we used the same computer. Finally, the results of different models are shown in the table below:

Model	Accuracy	Precision	Recall	F1-Score	Running time
Decision Tree	0.827	0.683	0.746	0.713	0.016
SVM	0.822	0.851	0.900	0.875	1.356
Logistic Regression	0.797	0.730	0.541	0.622	0.004

Table 1: Test results for different models

From the table, it can be seen that although the model I have chosen has the highest accuracy, there are still shortcomings for other evaluation performance cases. That is to say, it can be intuitively seen from the table that the Precision of the decision tree is the lowest among the three models, and the Recall and F1 scores are in the middle of the three models.

After confirming that the model has not been overfitted, I believe that the high accuracy of the model, but the precision, recall, and F1 score can be partially attributed to the category imbalance of the dataset. In this hotel reservation dataset, 8966 customers will cancel their reservations, while only 5381 customers will not cancel their reservations, which is almost twice the number of customers who will cancel their reservations. Due to the fact that the sample size of most categories far exceeds that of a few categories, the model may perform well in most categories, as it tends to categorize itself in most categories when predicting samples, while the prediction performance is poor in categories with small sample sizes. Therefore, the prediction performance of the model on a few categories may be poor, as the model has relatively little training data on these categories and cannot fully learn the features and patterns of a few categories. This can lead to this situation.

Finally, there is the running time. The decision tree of the model I selected has a faster running time than logistic regression and SVM. I think the reasons are as follows: The computational complexity of decision trees is relatively low because they typically only require binary comparisons, unlike SVM, which require complex calculations. However, compared with logistic regression, the decision tree usually involves more binary comparisons at each node with a lot of division of the dataset, which has a more computational cost considering the large volumes and features of our dataset. In summary, in the case of high-dimensional data, decision trees may exhibit shorter runtime compared to SVM due to their low computational complexity.

Advantages and Disadvantages of Decision Tree and improvements

Advantages

The advantage of decision tree is as follows:

1. The decision tree algorithm has strong interpretability, and its principles are easy to understand and visually analyze
2. Can handle both continuous data and discrete features simultaneously, without the need for additional feature engineering
3. Decision trees usually have a certain degree of robustness to outliers, and outliers only have a significant impact when the model is overfitting
4. It can handle missing values, so the decision tree does not require additional preprocessing to solve the problem of missing values
5. Decision trees are a non parametric model, so they do not need to make assumptions about the distribution of data, so they are suitable for various types of data
6. The decision tree model is nonlinear, which means it can be widely applied to more complex problems

Disadvantages

The disadvantage of decision tree is as follows:

1. Decision trees are prone to overfitting if certain parameters are not restricted
2. Even if the data undergoes minor changes, the generated model can still be significantly different
3. Decision trees are sensitive to noise present in the data
4. In the case of imbalanced categories in the dataset, decision trees have poor classification performance for data with fewer categories
5. For large datasets, decision trees have high computational costs and complexity
6. Decision trees are a greedy algorithm that is prone to falling into local optima rather than finding the overall optimal solution

Future improvements

1. For the parameter tuning section, only three parameters are adjusted, namely cp, minbucket, and minsplit. You can try adjusting other parameters, such as the method of calculating cross entropy and max depth, to achieve better model performance.
2. According to the dataset, it can be inferred that the labels in the dataset are imbalanced. There are a total of 9889 pieces of data for cancelled categories, while there are 4458 pieces of data for unclaimed categories. Due to the potential poor performance of decision trees in handling imbalanced datasets and their tendency towards the category with a large amount of data, future efforts can focus on improving the performance of decision trees in handling imbalanced data.
3. Compared to a single decision tree, an ensemble model can be used for random forest prediction. Because decision trees are sensitive to data, it is mentioned in the shortcomings of the model that small data changes may also cause significant differences in the model. So using random forests can help improve the generalization ability of the model and reduce the risk of overfitting, potentially improving the accuracy of the model.