Tuning Parameters for Random Forest Using DoE

Proud Jiao, Xiaocong Xuan, Yuetong Li

June 5, 2022

1 Introduction

Random forest is one of the most successful algorithms for tackling classification problems. The algorithm implements specialized statistical methods and techniques and involves several tuning parameters that affect its performance.

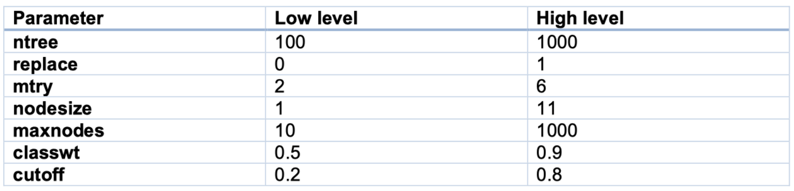
By conducting a cost-efficient experimental design and performing data analysis on the collected data, we hoped to 1) find the most important tuning parameters of random forest that drive the so-called cross-validation accuracy 2) find a model that maximizes cross-validation accuracy.

2 Methodology

2.1 Design of Experiment

2.1.1 Task Overview

There were 7 factors that we aimed to investigate simultaneously in this experiment. Figure 1 lists these factors along with the low and high levels for their settings that were determined. Due to budgetary constraints, the maximum number of tests in the whole experimentation process is 35. That is, we can test a maximum of 35 combinations of the settings of the tuning parameters. The goal is to find influential parameters and build a good model.



Parameter	**Low level**	**High level**
ntree	100	1000
replace	0	1
mtry	2	6
nodesize	1	11
maxnodes	10	1000
classwt	0.5	0.9
cutoff	0.2	0.8

Figure 1: Experimental Factors

2.1.2 Design Proposals

Ideally, a complete replicate of such design would require $2^7 = 128$ runs, which outgrows our resources. With limited runs, we propose a 2^{7-2} fractional design which only requires 32 runs. We would use the remaining 3 runs to conduct confirmation experiments. The specific design matrix for 32-run factorial design in coded form is shown in the Appendix Section ”Question 1”.

Our second proposal is a 32-run optimal design. The advantage of an optimal design is more flexible in the number of runs. It saves the cost by ignoring the less important higher-order interactions. Therefore, if we are interested in main effects and two-factor interactions, then we only need to estimate 1 intercept + 7 main effects + 7×6 / 2 two-factor interaction effects. This means that an experiment with $1 + 7 + 21 = 29$ observations can do the job! However, since we have spare runs up to 35 runs. We plan to use a 32-run optimal design, which will make stronger inferences than the 29-run. Among several metrics for an optimal design, we propose an I-optimal design because it will make more accurate predictions while keeping variance inflation factors (VIF) low. It is instructive to compare

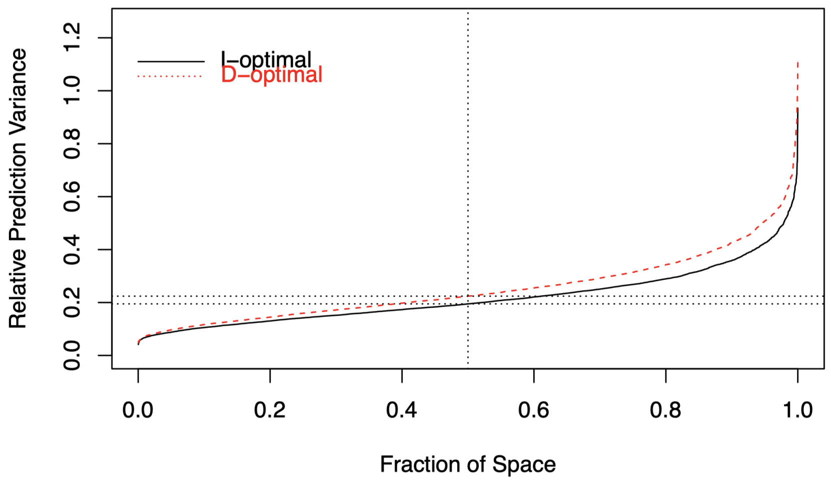


Figure 2: FDS

the I-optimal design versus a D-optimal design of the same size and for the same model. We use the Fraction of Design Space (FDS) plot to evaluate the performance of optimal designs. The lower the curve the better.

The FDS plot (Figure 2) above shows that the I-optimal design is better than the D-optimal design since the black curve is under the red curve. Therefore, the I-optimal design provides smaller (relative) prediction variances than the D-optimal design across the entire experimental region. The specific design matrix for 32-run I-optimal design in coded form is shown in the Appendix Section "Question 1".

2.1.3 Performance of Designs

To measure the performance of a design, we check on aliasing and multicollinearity between the effects of investigation. The color map (Figure 3) of the fractional design is shown below. There is no alias that exists between the main effects. This is good for the estimation of the significance of the main effects. However, the alias does exist between three particular two-factor interactions. We won't be able to estimate those 3 two-factor interactions with the alias.

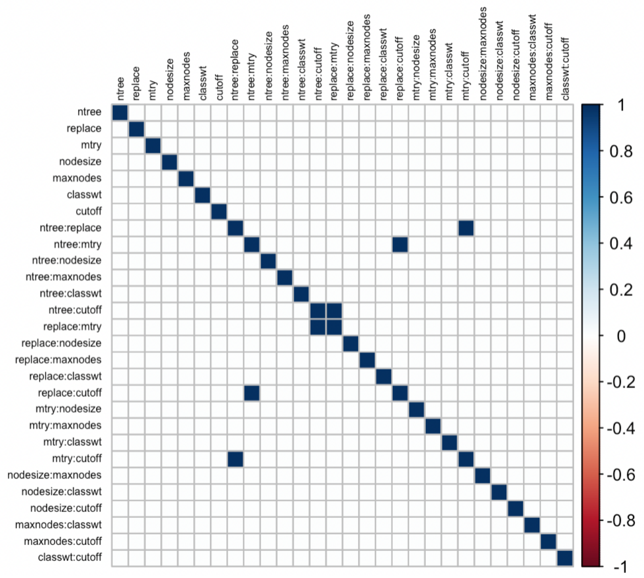


Figure 3: Color Map on Correlations for Factorial Design

Next, The color map (Figure 4) of the I-optimal design is shown below. For the optimal design, minor partial alias exists between main effects and interactions between two effects. Specifically, the off-diagonal elements are light red and blue, which means the correlations are not severe. The optimal design estimate well for main effects and all two-parameter interactions. We also further inspect the estimation properties of the I-optimal design using the VIFs. The VIFs are all around 1. Hence, each correlation is not significant. We get the same result from the color map. The specific VIFs for 32-run I-optimal design in coded form is shown in the Appendix Section "Question 2".

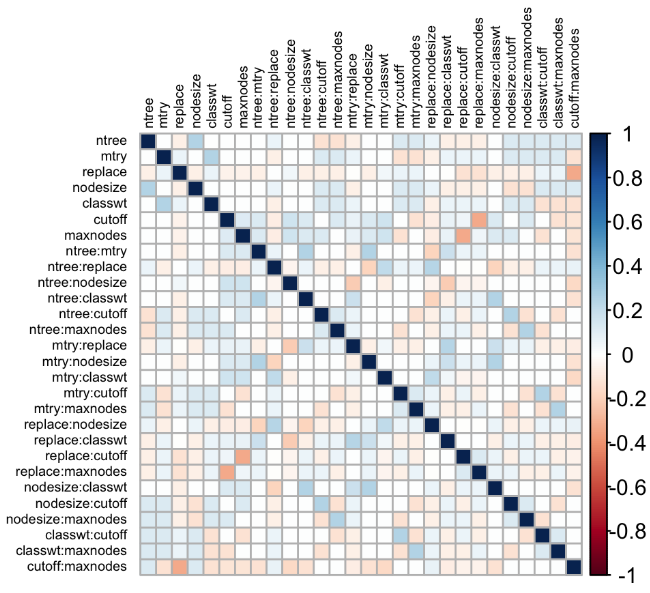


Figure 4: Color Map on Correlations for I-optimal Design

2.1.4 Recommended Design

The fractional factorial design includes alias terms that are inestimable. We have no reason to ignore the aliased interaction terms, so we should otherwise prefer to use the I-optimal design, from which we can estimate the main effects and all two-parameter interactions. In this case, we do not assume any previous knowledge of the aliased interaction. Hence, we recommend the 32-run I-optimal design.

2.1.5 Alternative Design By Commercial Software

We compare our design to an alternative 22-run design produced by a commercial software. The specific alternative design in coded form is shown in the Appendix Section "Question 4".

The alternatively designed matrix included three levels for 6 tuning parameters and two levels for 1 tuning parameter. Normally, a design that includes three-level interactions will allow the production of quadratic models and identification of significant quadratic terms, and thus provide more precise predictions. However, the alternative design only recommends 22 runs instead of utilizing more runs. Hence, using 22 runs means there will be a serious aliasing issue and we will not be able to estimate for most terms. While the alternative design might allow 35-22 = 13 runs for confirmation experiments. We do not need as many as 13 runs for confirmation experiment.

Here we constructed a correlation plot (Figure 5) for estimating the main effects and two-factor interaction terms. There is minor partial aliasing among main effects, no aliasing between main effects and two-way interactions, and mild heavy aliasing among interaction terms. (Existence of many dark spots indicates heavy partial aliasing among two-factor interaction terms.) Furthermore, we cannot construct a VIF table for main effects and two-factor interactions due to the existence of serious aliasing. Since we need a model that estimates well for both main effects and two-factor interactions. We recommend a 32-run I-optimal design.

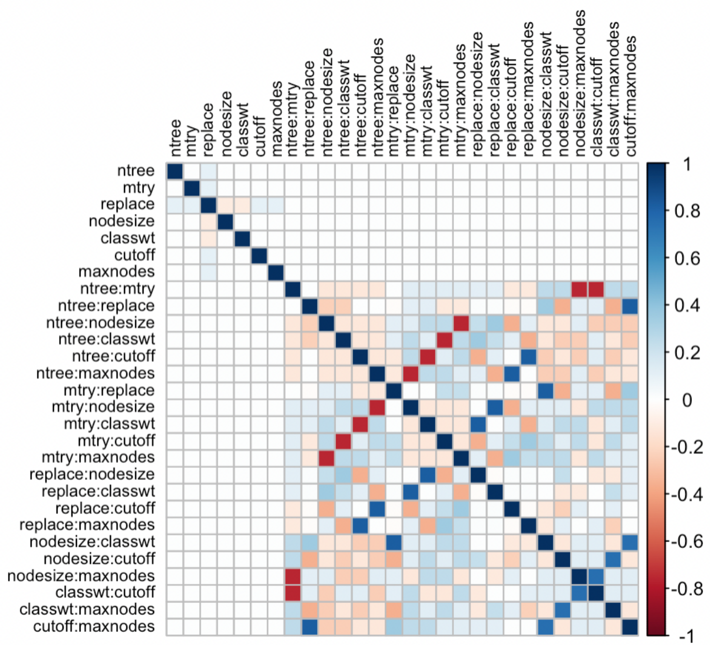


Figure 5: Color Map on Correlations for given Design

2.2 Data Analysis

2.2.1 Data Collection

After confirming to use I-optimal design, we convert the coded level to the actual level. Then we collect the cross-validation data by using $cv.rf$ function with datasets named "hearts". The collected data from the 32-run I-optimal design is shown in the Appendix Section "Question 5".

2.2.2 Model Fitting

In order to find the most important tuning parameters of random forest that drive the so-called cross-validation accuracy, we first fit the main effect and the interactions between two factors into a linear model. The summary of the model shows that only three variables are significant (p-value $\leq$ 0.05), which are "classwt", "cuttoff" and "classwt:maxnodes". Then we create a simplified model $cv = 0.615910\beta_0 - 0.060241 \times classwt + 0.026423 \times cutoff + 0.025019 \times classwt : maxnodes$ by only fitting the significant variables mentioned above and the summary table shows all of them are significant. The constructions of the null model and simplified model are shown in the Appendix Section "Question 6".

2.2.3 Model Validation

In order to check whether the model assumptions have been fulfilled, we plotted the normal probability plots (Figure 6 - left) and conducted residuals analysis for the fitted model(Figure 6 - right).

From the normal QQ-plot, most of the points follow the normal distribution, but a few points deviations from the straight line of the normal QQ plot. Hence, the normality assumption of the residuals was not perfectly fulfilled. Given more runs, we are able to investigate the possibility of a model that includes quadratic terms, in which the Q-Q plot may contain fewer points departing from the straight line. Furthermore, the constant variance assumption of the residuals seemed to be satisfied. The Residual vs. Predicted plot does not show a significant pattern, so we can conclude that the variance of the residuals is constant with different levels. For clarification, we do not construct Residuals vs. Run Order plot because the nature by which we generated our data (using $cv.rf$) guarantees randomization and independence of the individual observations. On another note, even though the normality assumption is not perfect enough, our sample size of 32 will allow the central limit theorem to guarantee a good approximation of p-values for the t-tests of the model parameters. Thus our model inference is unaffected.

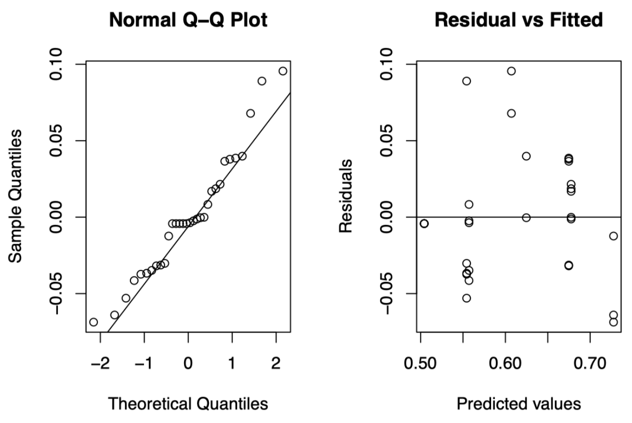


Figure 6: check the model assumptions

The adjusted R-square value is also a good metric for measuring model accuracy. Ideally, the adjusted R-squared value should explain as much percent of the variance of the data as possible. We had an adjusted R^2 of 0.753. This is sufficient yet not optimal. We suspect a model with quadratic terms will have a better fit to the model and thus produce higher R^2. To this end, further investigation is needed.

2.2.4 Confirmation Experiment

We conducted 3 additional runs to validate our previous conclusion by randomly choosing the factor settings from the rest of not used runs in the full factorial design. The randomly factor settings are shown in the Appendix Section "Question 6". Then we use our model to predict the cross-validation accuracy and compare our prediction with the value output by *cv.rf* function. The values of actual response and the prediction are shown in the Appendix Section "Question 6". The absolute difference between the actual response and our prediction value are all smaller than 0.05, so the predictions are good. Hence, we can conclude that our constructed model is able to predict the value of cross-validation accurately.

3 Conclusions

Our model included the most important tuning parameters of random forest and is able to predict cross-validation values with the desired accuracy. Due to the limited 35-run, we constructed a 32-run I-optimal design with the main effects and the interactions between two effects. Our design works well because this design does not contain any significant correlation by evaluating the VIFs. Furthermore, our model also works well. Furthermore, our constructed model for predicting the accuracy of cross-validation indicated that the model assumptions have been fulfilled. When we ran 3 additional tests for confirmation experiments using the random test combinations, we were able to know that our predictions and interpretations of the cross-validation were accurate.

One thing to note is that we only used one dataset, i.e. the heart datasets, for finding the best parameters for the random forest. This means that our results for the bet parameters do not generalize to all datasets but merely the one provided to us. For future experiments, we would recommend setting 'classwt' with low level, 'maxnodes' with low level, and 'cutoff' with high level to maximize the accuracy of cross-validation. Finding optimal settings by *optim*() is shown in the Appendix Section "Question 6". If the budgetary constraints allow the maximum number of tests to be larger than 35, our recommendation is to propose a design with 40 runs. Specifically, 36 runs for a design that includes main effects, two-factor interactions, and quadratic effects. and 4 runs for confirmation experiments. This will let us have a greater capability to identify the important active factors and construct a better model.

Appendix For Final Project

Proud Jiao, Xiaocong Xuan, Yuetong Li

Question 1. Propose a fractional factorial design for the problem. In addition, propose an experimental design constructed using the optimal design approach.

We have 7 parameters to investigate. However, a complete replicate of such design would require $2^7 = 128$ runs, which outgrows our resources. Due to budgetary constraints, we can try creating a 2^{7-2} fractional design with 32 runs. The fractional design generated by FrF2 library is demonstrated below:

```r
library(FrF2)
# Fractional factorial design
factors <- list(ntree = c(100, 1000),
                mtry = c(2, 6),
                replace = c(0, 1),
                nodesize = c(1,11),
                classwt = c(0.5, 0.9),
                cutoff = c(0.2, 0.8),
                maxnodes = c(10, 1000))
factorial_design <- FrF2(nruns = 32,
                  nfactors = 7,
                  alias.info = 2,
                  randomize = F,
                  factor.names = factors)
summary(factorial_design)
```

```
## Call:
## FrF2(nruns = 32, nfactors = 7, alias.info = 2, randomize = F,
##     factor.names = factors)
##
## Experimental design of type  FrF2
## 32  runs
##
## Factor settings (scale ends):
##   ntree mtry replace nodesize classwt cutoff maxnodes
## 1   100    2       0        1     0.5    0.2       10
## 2  1000    6       1       11     0.9    0.8     1000
##
## Design generating information:
## $legend
## [1] A=ntree    B=mtry     C=replace  D=nodesize E=classwt  F=cutoff   G=maxnodes
##
## $generators
## [1] F=ABC  G=ABDE
##
##
```

```
## Alias structure:
## $fi2
## [1] AB=CF AC=BF AF=BC
##
##
## The design itself:
##    ntree mtry replace nodesize classwt cutoff maxnodes
## 1    100    2       0        1     0.5    0.2     1000
## 2   1000    2       0        1     0.5    0.8       10
## 3    100    6       0        1     0.5    0.8       10
## 4   1000    6       0        1     0.5    0.2     1000
## 5    100    2       1        1     0.5    0.8     1000
## 6   1000    2       1        1     0.5    0.2       10
## 7    100    6       1        1     0.5    0.2       10
## 8   1000    6       1        1     0.5    0.8     1000
## 9    100    2       0       11     0.5    0.2       10
## 10  1000    2       0       11     0.5    0.8     1000
## 11   100    6       0       11     0.5    0.8     1000
## 12  1000    6       0       11     0.5    0.2       10
## 13   100    2       1       11     0.5    0.8       10
## 14  1000    2       1       11     0.5    0.2     1000
## 15   100    6       1       11     0.5    0.2     1000
## 16  1000    6       1       11     0.5    0.8       10
## 17   100    2       0        1     0.9    0.2       10
## 18  1000    2       0        1     0.9    0.8     1000
## 19   100    6       0        1     0.9    0.8     1000
## 20  1000    6       0        1     0.9    0.2       10
## 21   100    2       1        1     0.9    0.8       10
## 22  1000    2       1        1     0.9    0.2     1000
## 23   100    6       1        1     0.9    0.2     1000
## 24  1000    6       1        1     0.9    0.8       10
## 25   100    2       0       11     0.9    0.2     1000
## 26  1000    2       0       11     0.9    0.8       10
## 27   100    6       0       11     0.9    0.8       10
## 28  1000    6       0       11     0.9    0.2     1000
## 29   100    2       1       11     0.9    0.8     1000
## 30  1000    2       1       11     0.9    0.2       10
## 31   100    6       1       11     0.9    0.2       10
## 32  1000    6       1       11     0.9    0.8     1000
## class=design, type= FrF2
```

Alternatively, we could have generated a D-optimal design. The advantage of a optimal design is that is more flexible in the number of runs. It saves the cost by ignoring the less important higher-order interactions. Therefore, if we are only interested in main effects and two-factor interactions, then we only need to estimate:

- 1 intercept
- 7 main effects
- 7 × 6 / 2 = 21 two-factor interaction effects

An experiment with 1 + 7 + 21 = 29 observations can do the job!

However, since we have spare runs up to 35 runs. We will use a 32-run D-optimal design, which will make stronger inferences than a 29-run optimal design. We will leave 3 runs for confirmation experiments. The 32-run optimal design in coded form is as follows and also compare them:

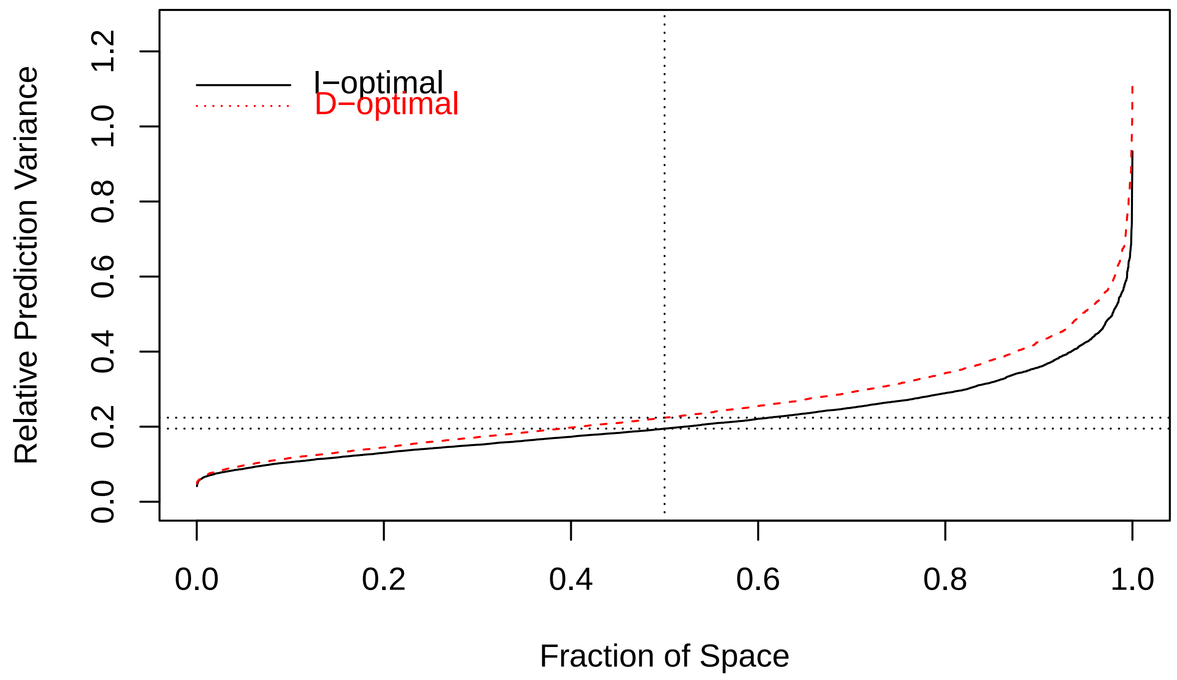
```
# Constructing a D-optimal design
library(AlgDesign)
candidate.set <- gen.factorial(levels=2,
                               nVars = 7,
                               varNames = names(factors))
optimal_design <- optFederov(~(ntree + mtry + replace + nodesize + classwt + cutoff + maxnodes)^2, candi
optimal_design$design
```

```
##     ntree mtry replace nodesize classwt cutoff maxnodes
## 4       1    1      -1       -1      -1     -1       -1
## 5      -1   -1       1       -1      -1     -1       -1
## 9      -1   -1      -1        1      -1     -1       -1
## 14      1   -1       1        1      -1     -1       -1
## 15     -1    1       1        1      -1     -1       -1
## 17     -1   -1      -1       -1       1     -1       -1
## 24      1    1       1       -1       1     -1       -1
## 26      1   -1      -1        1       1     -1       -1
## 27     -1    1      -1        1       1     -1       -1
## 29     -1   -1       1        1       1     -1       -1
## 34      1   -1      -1       -1      -1      1       -1
## 35     -1    1      -1       -1      -1      1       -1
## 44      1    1      -1        1      -1      1       -1
## 45     -1   -1       1        1      -1      1       -1
## 55     -1    1       1       -1       1      1       -1
## 57     -1   -1      -1        1       1      1       -1
## 64      1    1       1        1       1      1       -1
## 66      1   -1      -1       -1      -1     -1        1
## 67     -1    1      -1       -1      -1     -1        1
## 76      1    1      -1        1      -1     -1        1
## 77     -1   -1       1        1      -1     -1        1
## 87     -1    1       1       -1       1     -1        1
## 89     -1   -1      -1        1       1     -1        1
## 96      1    1       1        1       1     -1        1
## 101    -1   -1       1       -1      -1      1        1
## 104     1    1       1       -1      -1      1        1
## 106     1   -1      -1        1      -1      1        1
## 107    -1    1      -1        1      -1      1        1
## 113    -1   -1      -1       -1       1      1        1
## 116     1    1      -1       -1       1      1        1
## 118     1   -1       1       -1       1      1        1
## 127    -1    1       1        1       1      1        1
```

```
# Constructing an I-optimal design
optimal_design_I <- optFederov(~(ntree + mtry + replace + nodesize + classwt + cutoff + maxnodes)^2, can
print.data.frame(optimal_design_I$design)
```

```
##     ntree mtry replace nodesize classwt cutoff maxnodes
## 2       1   -1      -1       -1      -1     -1       -1
## 3      -1    1      -1       -1      -1     -1       -1
## 12      1    1      -1        1      -1     -1       -1
## 14      1   -1       1        1      -1     -1       -1
## 15     -1    1       1        1      -1     -1       -1
## 20      1    1      -1       -1       1     -1       -1
## 21     -1   -1      1       -1      1     -1       -1
## 26      1   -1     -1        1      1     -1       -1
## 27     -1    1     -1        1      1     -1       -1
## 32      1    1      1        1      1     -1       -1
## 38      1   -1      1       -1     -1      1       -1
## 40      1    1      1       -1     -1      1       -1
## 41     -1   -1     -1        1     -1      1       -1
## 50      1   -1     -1       -1      1      1       -1
## 51     -1    1     -1       -1      1      1       -1
## 60      1    1     -1        1      1      1       -1
## 62      1   -1      1        1      1      1       -1
## 63     -1    1      1        1      1      1       -1
## 72      1    1      1       -1     -1     -1        1
## 73     -1   -1     -1        1     -1     -1        1
## 81     -1   -1     -1       -1      1     -1        1
## 86      1   -1      1       -1      1     -1        1
## 87     -1    1      1       -1      1     -1        1
## 92      1    1     -1        1      1     -1        1
## 93     -1   -1      1        1      1     -1        1
## 100     1    1     -1       -1     -1      1        1
## 101    -1   -1      1       -1     -1      1        1
## 106     1   -1     -1        1     -1      1        1
## 107    -1    1     -1        1     -1      1        1
## 112     1    1      1        1     -1      1        1
## 120     1    1      1       -1      1      1        1
## 121    -1   -1     -1        1      1      1        1
```

```
I.opt <- optimal_design_I$design

# We compare the D-optimal design and I-optimal design using the FDS plot.
library(Vdgraph)
D.opt <- optimal_design$design
Compare2FDS(I.opt, D.opt, "I-optimal", "D-optimal", mod = 1)
```



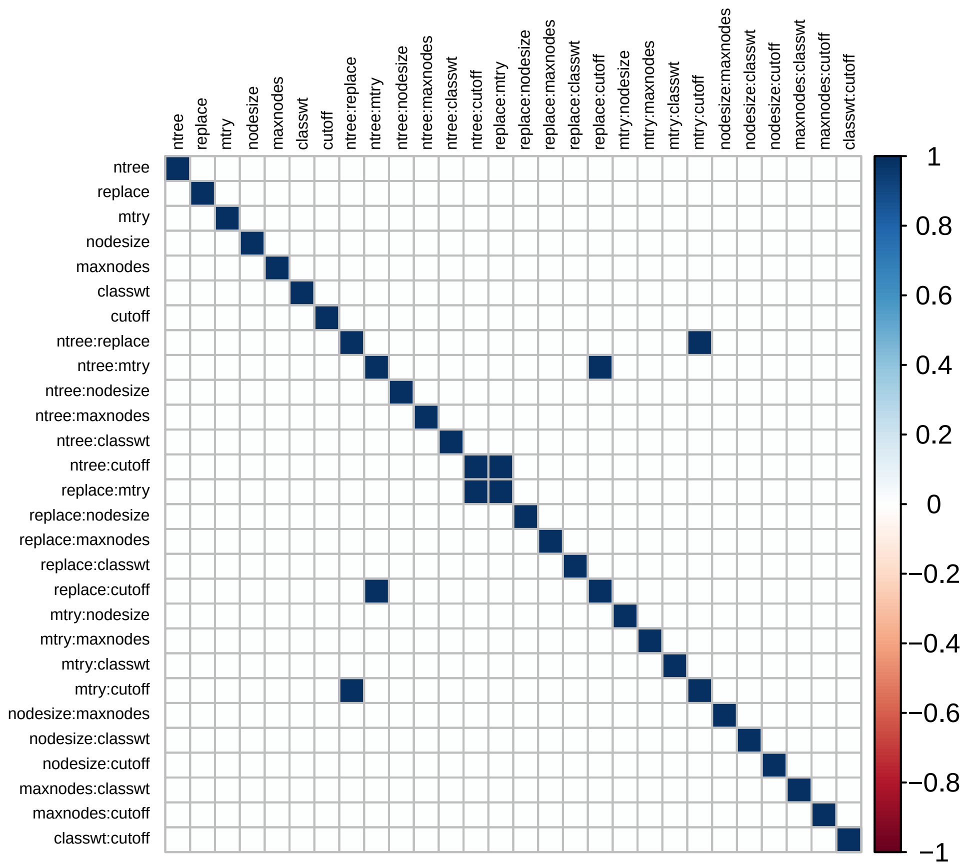
Question 2. Compare the optimal design with the fractional factorial design in practical and statistical terms. For instance, what is the performance of the designs for studying the main effects of the tuning parameters only? Can they estimate all two-parameter interactions? Why or why not? How do they compare in terms of multicollinearity?

To measure the performance of a design, we check on aliasing and collinearity between the effects of investigation, in this case, main effects and interaction terms.

```
library(corrplot)
```

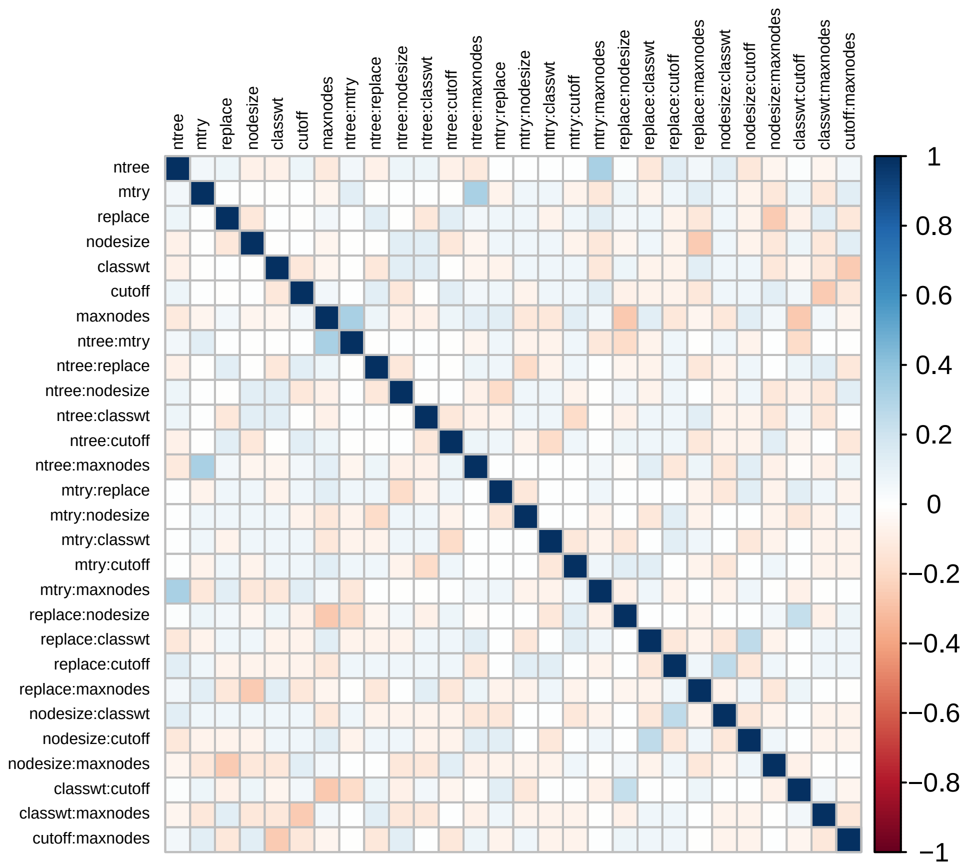
```
## corrplot 0.92 loaded
```

```
# Fractional design:
# Create the model matrix including main effects and two-factor interactions.
X.one <- model.matrix(~(ntree + replace + mtry + nodesize + maxnodes + classwt + cutoff)^2-1, data.frame

# Create color map on pairwise correlations.
contrast.vectors.correlations.one <- cor(X.one)
corrplot(contrast.vectors.correlations.one, type = "full", addgrid.col = "gray", tl.col = "black", tl.s
```



For the fractional design, no alias exists between main effects. This is good for the estimation of significance of the main effects. However, alias does exist between two-factor interactions. We won't be able to estimate those two-factor interactions with alias.

```
# I-optimal design:
# Create the model matrix including main effects and two-factor interactions.
X.opt <- model.matrix(~(ntree + mtry + replace + nodesize + classwt + cutoff + maxnodes)^2-1, data.fram
# Create color map on pairwise correlations.
contrast.vectors.correlations.opt <- cor(X.opt)

X.opt <- model.matrix(~(ntree + mtry + replace + nodesize + classwt + cutoff + maxnodes)^2, data.frame(
corrplot(contrast.vectors.correlations.opt, type = "full", addgrid.col = "gray", tl.col = "black", tl.s
```



The color map shows that there is some partial aliasing in the design. We can further inspect the estimation properties of the I-optimal design using the VIFs.

```
XtX <- t(X.opt)%*%X.opt
inv.XtX <- solve(XtX)
var.eff <- diag(inv.XtX)
cat("\n Variance inflation factors \n")
```

```
##
##  Variance inflation factors
```

```
print(nrow(I.opt)*var.eff)
```

```
##       (Intercept)             ntree              mtry           replace
##          1.214030          1.312708          1.299472          1.309002
##          nodesize           classwt            cutoff          maxnodes
##          1.309002          1.309002          1.309002          1.458211
##        ntree:mtry     ntree:replace    ntree:nodesize     ntree:classwt
##          1.289039          1.272499          1.272499          1.272499
##      ntree:cutoff    ntree:maxnodes      mtry:replace     mtry:nodesize
##          1.272499          1.312708          1.229556          1.229556
##      mtry:classwt       mtry:cutoff     mtry:maxnodes  replace:nodesize
##          1.229556          1.229556          1.299472          1.306483
##   replace:classwt    replace:cutoff  replace:maxnodes  nodesize:classwt
##          1.330672          1.330672          1.309002          1.330672
##   nodesize:cutoff nodesize:maxnodes    classwt:cutoff  classwt:maxnodes
##          1.330672          1.309002          1.306483          1.309002
##   cutoff:maxnodes
##          1.309002
```

Question 4. Using a commercial software, the TAs and I came up with the experimental design shown in Table 2. How does your recommended design in the previous question compare with this one?

```
# Alternative 22-run design produced by a commercial software (Table 2)
# Transform to the coded level
ntree <- c(100, 550, 1000, 1000, 1000, 100, 1000, 100, 100, 100,
           100, 1000, 100, 550, 100, 1000, 1000, 1000, 100, 1000,
           550, 550)
c_ntree <- c(-1, 0, 1, 1, 1, -1, 1, -1, -1, -1,
             -1, 1, -1, 0, -1, 1, 1, 1, -1, 1,
             0, 0)

replace <- c(1, 0, 1, 1, 0, 0, 0, 0, 1, 0,
             0, 1, 1, 0, 0, 0, 0, 1, 1, 1,
             1, 1)
c_replace <- c(1, -1, 1, 1, -1, -1, -1, -1, 1, -1,
             -1, 1, 1, -1, -1, -1, -1, 1, 1, 1,
             1, 1)

mtry <- c(2, 2, 4, 6, 6, 2, 2, 2, 6, 6,
          4, 2, 6, 4, 6, 6, 2, 6, 2, 2,
          4, 6)
c_mtry <- c(-1, -1, 0, 1, 1, -1, -1, -1, 1, 1,
          0, -1, 1, 0, 1, 1, -1, 1, -1, -1,
          0, 1)

nodesize <- c(11, 1, 1, 1, 1, 1, 6, 11, 1, 1,
              11, 11, 6, 6, 11, 11, 11, 11, 1, 1,
              6, 11)
c_nodesize <- c(1, -1, -1, -1, -1, -1, 0, 1, -1, -1,
              1, 1, 0, 0, 1, 1, 1, 1, -1, -1,
              0, 1)


maxnodes <- c(10, 10, 10, 1000, 1000, 1000, 10, 10, 10, 10,
              1000, 1000, 1000, 505, 505, 10, 1000, 10, 1000, 505,
              505, 1000)
c_maxnodes <- c(-1, -1, -1, 1, 1, 1, -1, -1, -1, -1,
              1, 1, 1, 0, 0, -1, 1, -1, 1, 0,
              0, 1)

classwt <- c(0.5, 0.5, 0.5, 0.5, 0.9, 0.5, 0.9, 0.9, 0.7, 0.9,
             0.9, 0.5, 0.5, 0.7, 0.5, 0.5, 0.7, 0.9, 0.9, 0.9,
             0.7, 0.9)
c_classwt <- c(-1, -1, -1, -1, 1, -1, 1, 1, 0, 1,
             1, -1, -1, 0, -1, -1, 0, 1, 1, 1,
             0, 1)

cutoff <- c(0.8, 0.2, 0.2, 0.8, 0.2, 0.8, 0.8, 0.2, 0.8, 0.5,
            0.8, 0.5, 0.2, 0.5, 0.2, 0.8, 0.2, 0.2, 0.2, 0.8,
            0.5, 0.8)
c_cutoff <- c(1, -1, -1, 1, -1, 1, 1, -1, 1, 0,
            1, 0, -1, 0, -1, 1, -1, -1, -1, 1,
            0, 1)

alter <- cbind(ntree, mtry, replace, nodesize, classwt, cutoff, maxnodes)
alter <- data.frame(alter)

c_alter <- cbind(c_ntree, c_mtry, c_replace, c_nodesize,
                 c_classwt, c_cutoff, c_maxnodes)
c_alter <- data.frame(c_alter)

alter; c_alter
```

```
##    ntree mtry replace nodesize classwt cutoff maxnodes
## 1    100    2       1       11     0.5    0.8       10
## 2    550    2       0        1     0.5    0.2       10
## 3   1000    4       1        1     0.5    0.2       10
## 4   1000    6       1        1     0.5    0.8     1000
## 5   1000    6       0        1     0.9    0.2     1000
## 6    100    2       0        1     0.5    0.8     1000
## 7   1000    2       0        6     0.9    0.8       10
## 8    100    2       0       11     0.9    0.2       10
## 9    100    6       1        1     0.7    0.8       10
## 10   100    6       0        1     0.9    0.5       10
## 11   100    4       0       11     0.9    0.8     1000
## 12  1000    2       1       11     0.5    0.5     1000
## 13   100    6       1        6     0.5    0.2     1000
## 14   550    4       0        6     0.7    0.5      505
## 15   100    6       0       11     0.5    0.2      505
## 16  1000    6       0       11     0.5    0.8       10
## 17  1000    2       0       11     0.7    0.2     1000
## 18  1000    6       1       11     0.9    0.2       10
## 19   100    2       1        1     0.9    0.2     1000
## 20  1000    2       1        1     0.9    0.8      505
## 21   550    4       1        6     0.7    0.5      505
## 22   550    6       1       11     0.9    0.8     1000
```

```
##    c_ntree c_mtry c_replace c_nodesize c_classwt c_cutoff c_maxnodes
## 1       -1     -1         1          1        -1        1         -1
## 2        0     -1        -1         -1        -1       -1         -1
## 3        1      0         1         -1        -1       -1         -1
## 4        1      1         1         -1        -1        1          1
## 5        1      1        -1         -1         1       -1          1
## 6       -1     -1        -1         -1        -1        1          1
## 7        1     -1        -1          0         1        1         -1
## 8       -1     -1        -1          1         1       -1         -1
## 9       -1      1         1         -1         0        1         -1
## 10      -1      1        -1         -1         1        0         -1
## 11      -1      0        -1          1         1        1          1
## 12       1     -1         1          1        -1        0          1
## 13      -1      1         1          0        -1       -1          1
## 14       0      0        -1          0         0        0          0
## 15      -1      1        -1          1        -1       -1          0
## 16       1      1        -1          1        -1        1         -1
## 17       1     -1        -1          1         0       -1          1
## 18       1      1         1          1         1       -1         -1
## 19      -1     -1         1         -1         1       -1          1
## 20       1     -1         1         -1         1        1          0
## 21       0      0         1          0         0        0          0
## 22       0      1         1          1         1        1          1
```

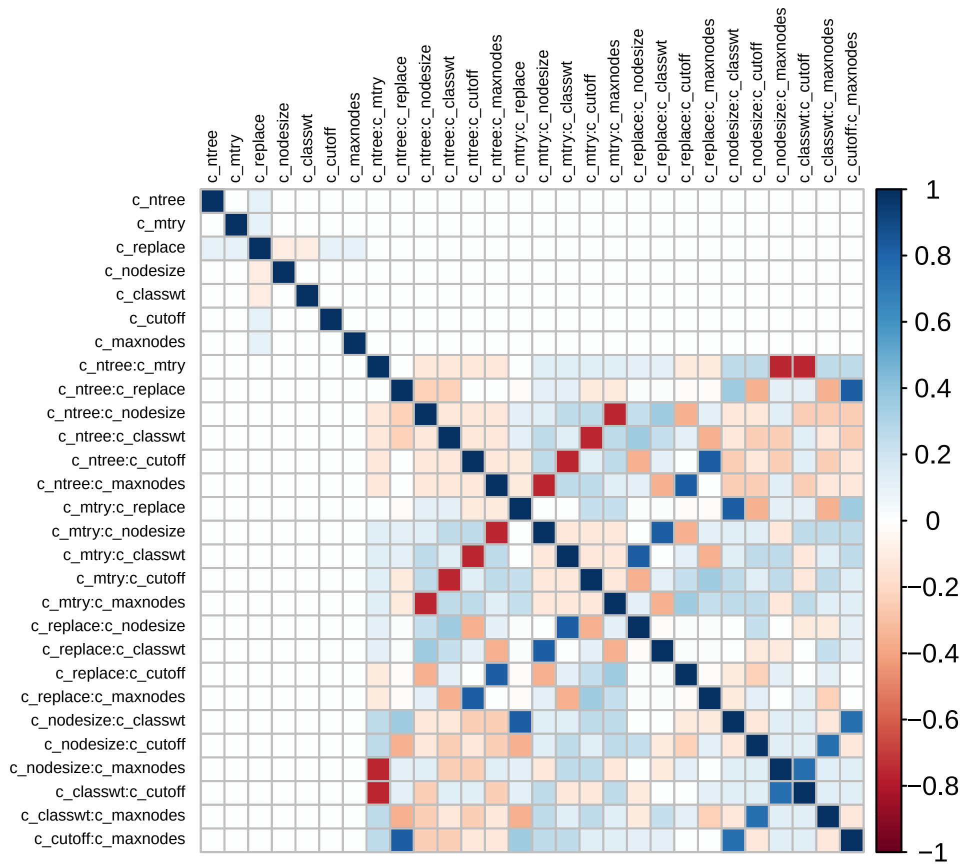
The alternatively designed matrix included three levels for most tuning parameters.

Normally, a design that include three-level interactions will allow the production of quadratic models and identification of significant quadratic terms, and thus provide more precise predictions.

However, the alternative design only recommends 22 runs instead of utilizing more runs. A full quadratic model would require 1+2*6+7*(7-1)/2 + 1 = 35 degree of freedom, i.e. 35 runs. Using 22 runs means there will be serious aliasing issue and we will not be able to estimate for most terms.

While the alternative design might allow 35-22 = 13 runs for confirmation experiments. We do not need as many as 13 runs for confirmation experiment

```
# Create color map on pairwise correlations for 22-run design.
X.alt <- model.matrix(~.^2-1, data.frame(c_alter))
contrast.vectors.correlations.alt <- cor(X.alt)
corrplot(contrast.vectors.correlations.alt, type = "full", addgrid.col = "gray", tl.col = "black", tl.s
```



Question 5. Collect data using your recommended design in Question 3.

```
# We convert the coded level to the actual level
I.design <- optimal_design_I$design

I.design$ntree[I.design$ntree == 1] <- 1000
I.design$ntree[I.design$ntree == -1] <- 100

I.design$mtry[I.design$mtry == 1] <- 6
I.design$mtry[I.design$mtry == -1] <- 2

I.design$replace[I.design$replace == -1] <- 0

I.design$nodesize[I.design$nodesize == 1] <- 11
I.design$nodesize[I.design$nodesize == -1] <- 1

I.design$classwt[I.design$classwt == 1] <- 0.9
I.design$classwt[I.design$classwt == -1] <- 0.5

I.design$cutoff[I.design$cutoff == 1] <- 0.8
I.design$cutoff[I.design$cutoff == -1] <- 0.2

I.design$maxnodes[I.design$maxnodes == 1] <- 1000
I.design$maxnodes[I.design$maxnodes == -1] <- 10
```

```r
I.design
```

```
##     ntree mtry replace nodesize classwt cutoff maxnodes
## 2    1000    2       0        1     0.5    0.2       10
## 3     100    6       0        1     0.5    0.2       10
## 12   1000    6       0       11     0.5    0.2       10
## 14   1000    2       1       11     0.5    0.2       10
## 15    100    6       1       11     0.5    0.2       10
## 20   1000    6       0        1     0.9    0.2       10
## 21    100    2       1        1     0.9    0.2       10
## 26   1000    2       0       11     0.9    0.2       10
## 27    100    6       0       11     0.9    0.2       10
## 32   1000    6       1       11     0.9    0.2       10
## 38   1000    2       1        1     0.5    0.8       10
## 40   1000    6       1        1     0.5    0.8       10
## 41    100    2       0       11     0.5    0.8       10
## 50   1000    2       0        1     0.9    0.8       10
## 51    100    6       0        1     0.9    0.8       10
## 60   1000    6       0       11     0.9    0.8       10
## 62   1000    2       1       11     0.9    0.8       10
## 63    100    6       1       11     0.9    0.8       10
## 72   1000    6       1        1     0.5    0.2     1000
## 73    100    2       0       11     0.5    0.2     1000
## 81    100    2       0        1     0.9    0.2     1000
## 86   1000    2       1        1     0.9    0.2     1000
## 87    100    6       1        1     0.9    0.2     1000
## 92   1000    6       0       11     0.9    0.2     1000
## 93    100    2       1       11     0.9    0.2     1000
## 100  1000    6       0        1     0.5    0.8     1000
## 101   100    2       1        1     0.5    0.8     1000
## 106  1000    2       0       11     0.5    0.8     1000
## 107   100    6       0       11     0.5    0.8     1000
## 112  1000    6       1       11     0.5    0.8     1000
## 120  1000    6       1        1     0.9    0.8     1000
## 121   100    2       0       11     0.9    0.8     1000
```

```r
load(file = "heart.RData")
source("CrossValidation_RF.R")
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```r
cv.rf(I.design, y, X)
```

```
## Collecting response on test combination  1
## Collecting response on test combination  2
## Collecting response on test combination  3
## Collecting response on test combination  4
## Collecting response on test combination  5
## Collecting response on test combination  6
## Collecting response on test combination  7
## Collecting response on test combination  8
## Collecting response on test combination  9
## Collecting response on test combination  10
## Collecting response on test combination  11
## Collecting response on test combination  12
## Collecting response on test combination  13
## Collecting response on test combination  14
## Collecting response on test combination  15
## Collecting response on test combination  16
## Collecting response on test combination  17
## Collecting response on test combination  18
## Collecting response on test combination  19
## Collecting response on test combination  20
## Collecting response on test combination  21
## Collecting response on test combination  22
## Collecting response on test combination  23
## Collecting response on test combination  24
## Collecting response on test combination  25
## Collecting response on test combination  26
## Collecting response on test combination  27
## Collecting response on test combination  28
## Collecting response on test combination  29
## Collecting response on test combination  30
## Collecting response on test combination  31
## Collecting response on test combination  32

##     ntree mtry replace nodesize classwt cutoff maxnodes        CV
## 2    1000    2       0        1     0.5    0.2       10 0.6434052
## 3     100    6       0        1     0.5    0.2       10 0.7126352
## 12   1000    6       0       11     0.5    0.2       10 0.7133776
## 14   1000    2       1       11     0.5    0.2       10 0.6428887
## 15    100    6       1       11     0.5    0.2       10 0.7112933
## 20   1000    6       0        1     0.9    0.2       10 0.5000044
## 21    100    2       1        1     0.9    0.2       10 0.4999999
## 26   1000    2       0       11     0.9    0.2       10 0.5000000
## 27    100    6       0       11     0.9    0.2       10 0.5000045
## 32   1000    6       1       11     0.9    0.2       10 0.5000000
## 38   1000    2       1        1     0.5    0.8       10 0.6635865
## 40   1000    6       1        1     0.5    0.8       10 0.7152352
## 41    100    2       0       11     0.5    0.8       10 0.6589157
## 50   1000    2       0        1     0.9    0.8       10 0.5156135
## 51    100    6       0        1     0.9    0.8       10 0.5546583
## 60   1000    6       0       11     0.9    0.8       10 0.5534090
## 62   1000    2       1       11     0.9    0.8       10 0.5222387
## 63    100    6       1       11     0.9    0.8       10 0.5653842
## 72   1000    6       1        1     0.5    0.2     1000 0.6243382
## 73    100    2       0       11     0.5    0.2     1000 0.6645824
## 81    100    2       0        1     0.9    0.2     1000 0.5175735
## 86   1000    2       1        1     0.9    0.2     1000 0.5169417
## 87    100    6       1        1     0.9    0.2     1000 0.6432756
## 92   1000    6       0       11     0.9    0.2     1000 0.5240403
## 93    100    2       1       11     0.9    0.2     1000 0.5012533
## 100  1000    6       0        1     0.5    0.8     1000 0.6774713
## 101   100    2       1        1     0.5    0.8     1000 0.6763149
## 106  1000    2       0       11     0.5    0.8     1000 0.6961511
## 107   100    6       0       11     0.5    0.8     1000 0.6990668
## 112  1000    6       1       11     0.5    0.8     1000 0.6944489
## 120  1000    6       1        1     0.9    0.8     1000 0.6749914
## 121   100    2       0       11     0.9    0.8     1000 0.7026534
```

```
# Collect data
data_cv <- c(0.6434052, 0.7126352, 0.7133776, 0.6428887, 0.7112933,
             0.5000044, 0.4999999, 0.5000000, 0.5000045, 0.5000000,
             0.6635865, 0.7152352, 0.6589157, 0.5156135, 0.5546583,
             0.5534090, 0.5222387, 0.5653842, 0.6243382, 0.6645824,
             0.5175735, 0.5169417, 0.6432756, 0.5240403, 0.5012533,
             0.6774713, 0.6763149, 0.6961511, 0.6990668, 0.6944489,
             0.6749914, 0.7026534)
```

Question 6. Conduct a detailed data analysis. What are the influential tuning parameters? What is the final model that links the tuning parameters to the cross-validation accuracy? Does the final model provide a good fit to the data?

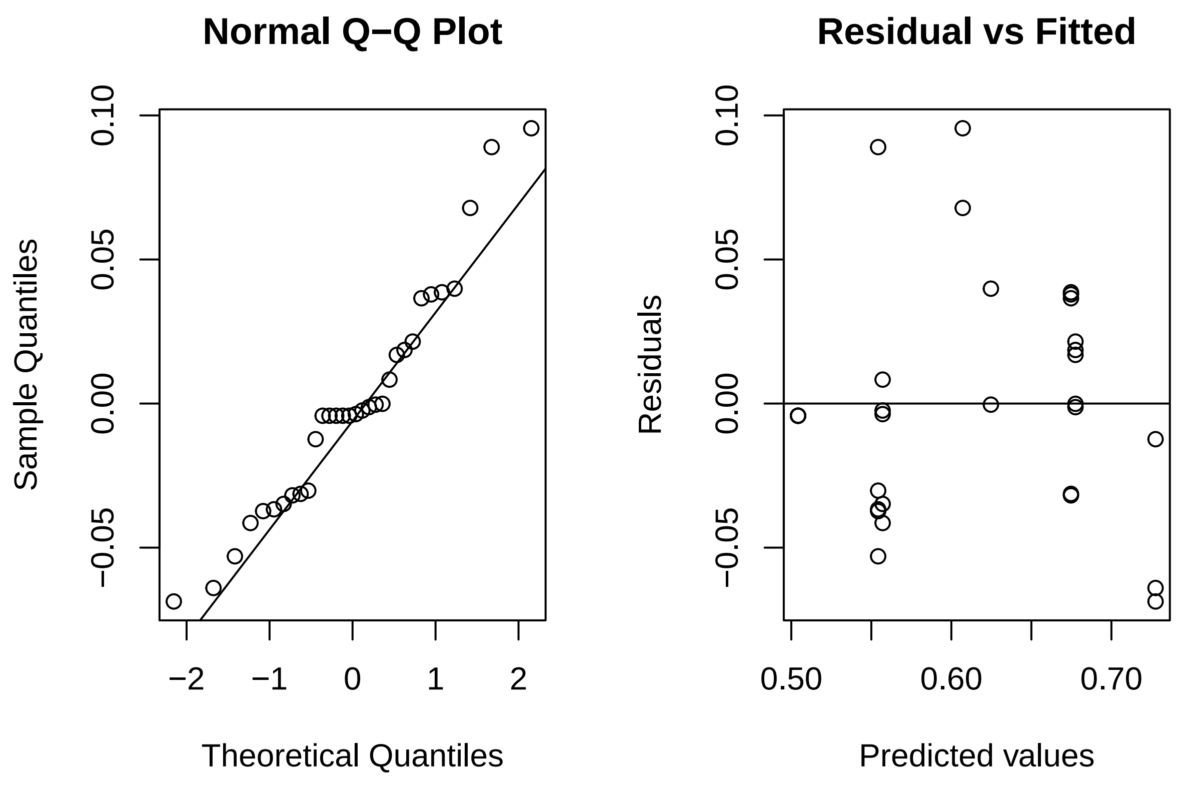
```
# Null model: the main effect and the interactions between two factors
new_data <- cbind(optimal_design_I$design, data_cv)
null_model <- lm(data_cv~.^2, data=new_data)
summary(null_model)
```

```
##
## Call:
## lm.default(formula = data_cv ~ .^2, data = new_data)
##
## Residuals:
##          2          3         12         14         15         20         21
## -0.0042317  0.0023032  0.0016071 -0.0030002  0.0010717  0.0016071 -0.0006428
##         26         27         32         38         40         41         50
##  0.0051427 -0.0054641  0.0016071  0.0106069 -0.0077141 -0.0006428 -0.0030002
##         51         60         62         63         72         73         81
##  0.0010717  0.0016071 -0.0042317  0.0023032 -0.0143749  0.0166249 -0.0143749
##         86         87         92         93        100        101        106
##  0.0148571  0.0148571 -0.0032142 -0.0143749  0.0166249 -0.0032142 -0.0161428
##        107        112        120        121
## -0.0161428  0.0166249 -0.0143749  0.0166249
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)          0.616171   0.006383  96.535 2.45e-06 ***
## ntree               -0.008991   0.006637  -1.355  0.26853
## mtry                 0.019578   0.006604   2.965  0.05932 .
## replace              0.003576   0.006628   0.540  0.62700
## nodesize            -0.002525   0.006628  -0.381  0.72857
## classwt             -0.051941   0.006628  -7.837  0.00433 **
## cutoff               0.028511   0.006628   4.302  0.02312 *
## maxnodes             0.013451   0.006995   1.923  0.15022
## ntree:mtry        -0.004855   0.006577  -0.738  0.51389
## ntree:replace     -0.003407   0.006535  -0.521  0.63814
## ntree:nodesize     0.004799   0.006535   0.734  0.51592
## ntree:classwt     -0.001601   0.006535  -0.245  0.82226
## ntree:cutoff       0.003117   0.006535   0.477  0.66598
## ntree:maxnodes    -0.007972   0.006637  -1.201  0.31589
## mtry:replace       0.009391   0.006424   1.462  0.23990
## mtry:nodesize     -0.007590   0.006424  -1.182  0.32249
## mtry:classwt      -0.002233   0.006424  -0.348  0.75110
## mtry:cutoff       -0.003182   0.006424  -0.495  0.65436
## mtry:maxnodes     -0.004229   0.006604  -0.640  0.56746
## replace:nodesize  -0.015035   0.006621  -2.271  0.10786
## replace:classwt    0.002810   0.006682   0.421  0.70238
## replace:cutoff     0.001912   0.006682   0.286  0.79336
## replace:maxnodes  -0.003425   0.006628  -0.517  0.64098
## nodesize:classwt  -0.004176   0.006682  -0.625  0.57636
## nodesize:cutoff    0.001854   0.006682   0.277  0.79946
## nodesize:maxnodes  0.002539   0.006628   0.383  0.72718
## classwt:cutoff     0.016939   0.006621   2.558  0.08335 .
## classwt:maxnodes   0.025838   0.006628   3.898  0.02995 *
## cutoff:maxnodes    0.019773   0.006628   2.983  0.05844 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.03277 on 3 degrees of freedom
## Multiple R-squared:  0.9854, Adjusted R-squared:  0.8489
## F-statistic:  7.22 on 28 and 3 DF,  p-value: 0.06386
```

```
# Simplified model: only include significant parameters
alter_model <- lm(data_cv ~ classwt+cutoff+classwt:maxnodes, data=new_data)
summary(alter_model)
```

```
##
## Call:
## lm.default(formula = data_cv ~ classwt + cutoff + classwt:maxnodes,
##     data = new_data)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.068678 -0.031472 -0.003943  0.019325  0.095542
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       0.615910   0.007452  82.654  < 2e-16 ***
## classwt          -0.060241   0.007572  -7.956 1.15e-08 ***
## cutoff            0.026423   0.007773   3.399  0.00205 **
## classwt:maxnodes  0.025019   0.007773   3.219  0.00325 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.04189 on 28 degrees of freedom
## Multiple R-squared:  0.7769, Adjusted R-squared:  0.753
## F-statistic: 32.51 on 3 and 28 DF,  p-value: 2.915e-09
```

```r
# Check normality
par(mfrow=c(1,2))
res <- alter_model$residuals
qqnorm(res); qqline(res)
# Check constant variance
plot(alter_model$fitted.values, res, xlab = 'Predicted values', ylab = 'Residuals', main = "Residual vs
abline(h = 0)
```



```r
# Recommened level
obj_func <- function(x){
  predicted <- 0.615910-0.060241*x[1]+0.026423*x[3]+0.025019*x[1]*x[2]
  -predicted
}

optim(par = c(0, 0, 0), fn = obj_func, lower = -1, upper = 1, method = "L-BFGS-B")
```

```
## $par
## [1] -1 -1  1
##
## $value
## [1] -0.727593
##
## $counts
## function gradient
##      127      127
```

```
##
## $convergence
## [1] 0
##
## $message
## [1] "CONVERGENCE: NORM OF PROJECTED GRADIENT <= PGTOL"
```

We would recommend setting 'classwt' with low level, 'maxnodes' with low level, and 'cutoff' with high level to maximize the accuracy of cross-validation.

```
# Confirmation Experiment
factors <- list(ntree = c(100, 1000),
                mtry = c(2, 6),
                replace = c(0, 1),
                nodesize = c(1,11),
                classwt = c(0.5, 0.9),
                cutoff = c(0.2, 0.8),
                maxnodes = c(10, 1000))
full_factorial_design <- FrF2(nruns = 2^7,
                  nfactors = 7,
                  randomize = F,
                  factor.names = factors)

a <- full_factorial_design
b <- I.design

x <- sapply(a, is.factor)
a[ , x] <- as.data.frame(apply(a[ , x], 2, as.numeric))

x <- sapply(b, is.factor)
b[ , x] <- as.data.frame(apply(b[ , x], 2, as.numeric))

# the rest of 128-32=96 did not use designs
library(dplyr)
D <- setdiff(a,b)

# randomly factor settings for three designs
set.seed(123)
D <- D[sample(1:nrow(D), 3), ]
D
```

```
##    ntree mtry replace nodesize classwt cutoff maxnodes
## 31  1000    6       0       11     0.5    0.8       10
## 79  1000    6       0       11     0.5    0.8     1000
## 51   100    2       1        1     0.5    0.2     1000
```

```
load(file = "heart.RData")
source("CrossValidation_RF.R")
cv.rf(D, y, X)
```

```
## Collecting response on test combination  1
## Collecting response on test combination  2
## Collecting response on test combination  3
##    ntree mtry replace nodesize classwt cutoff maxnodes        CV
## 31  1000    6       0       11     0.5    0.8       10 0.7167379
## 79  1000    6       0       11     0.5    0.8     1000 0.7016937
## 51   100    2       1        1     0.5    0.2     1000 0.6349859
```

```
# Actual response
confirm_cv <- c(0.7167379, 0.7016937, 0.6349859)

# Transform to the coded level
p1 <- data.frame(ntree = 1, mtry =1, replace = -1,
           nodesize = 1, classwt = -1, cutoff = 1,
           maxnodes = -1)
p2 <- data.frame(ntree = 1, mtry =1, replace = -1,
           nodesize = 1, classwt = -1, cutoff = 1,
           maxnodes = 1)
p3 <- data.frame(ntree = -1, mtry =-1, replace = 1,
           nodesize = -1, classwt = -1, cutoff = -1,
           maxnodes = 1)

v1 <- predict(alter_model, p1)
v2 <- predict(alter_model, p2)
v3 <- predict(alter_model, p3)
predict_v <- c(v1,v2,v3)
predict_v <- as.numeric(predict_v)

# Predictions
predict_v
```

```
## [1] 0.7275936 0.6775549 0.6247089
```

```
# Judge the predictions
as.numeric(abs(confirm_cv - predict_v))
```

```
## [1] 0.01085566 0.02413884 0.01027697
```

Statement of Contribution

Proud Jiao

- Wrote up the report: Design Proposals, Performance of Designs, Recommended Design
- Wrote code for Question 1-3
- Came up with the design of the experiment with group members

Yuetong Li

- Wrote up the report: analysis of Alternative Design By Commercial Software, Data Collection, Model Fitting
- Wrote code for Question 4-5
- Collected data using our design and conduct an ideal linear regression model

Xiaocong Xuan

- Wrote up the report: Introduction, Model Validation, Conclusions
- Wrote code for Question 6
- Consolidated and organized the Appendix file