ISYE 6501 HW9

2019/10/16

**Group Members:**  
Jinyu, Huang | [jhuang472@gatech.edu](mailto:jhuang472@gatech.edu) | GTID: 903522245  
Chengqi, Huang | [chengqihuang@gatech.edu](mailto:chengqihuang@gatech.edu) | GTID: 903534690  
Yuefan, Hu | [yuefanhu@gatech.edu](mailto:yuefanhu@gatech.edu) | GTID:903543027  
Jingyu, Li | [alanli@gatech.edu](mailto:alanli@gatech.edu) | GTID: 903520148

# Question 12.1

**Describe a situation or problem from your job, everyday life, current events, etc., for which a design of experiments approach would be appropriate.**

Suppose we are the record company running the hip-hop show Rhythm and Flow, we may want to know which candidate of the champion can bring greater benefits to the company, so that the record company will be able to cheat and manipulate the results of the competition behind the scenes.

Since we may have many potential championship candidates, and other candidates may also bring profit to the company, we don’t want to lose value. In this case, we can use the Multi Armed Bandit approach. During the last episode before the final, the company can randomly ask an audience to buy virtual currency in order to show his or her support to one of the candidates. Then we can renew the score of this particular candidate, and ask another audience rating another candidate. First, every candidate has equal probability of selecting. But as we keep renewing each candidate’s score, the probability of appearance of an individual candidate changes. Finally we are able to have the result of who is likely to be the best. Moreover, if every candidate have an equal probability of appearing, people are less likely to buy virtual currency to support a less popular candidate. Thus we successfully prevent losing money.

# Question 12.2

**To determine the value of 10 different yes/no features to the market value of a house (large yard, solar roof, etc.), a real estate agent plans to survey 50 potential buyers, showing a fictitious house with different combinations of features. To reduce the survey size, the agent wants to show just 16 fictitious houses. Use R’s FrF2 function (in the FrF2 package) to find a fractional factorial design for this experiment: what set of features should each of the 16 fictitious houses have? Note: the output of FrF2 is “1” (include) or “-1” (don’t include) for each feature.**

We use the FrF2 function to choose 16 different combinations among 10 factors

#First, we should load the FrF2 Package  
library(FrF2)

## Loading required package: DoE.base

## Loading required package: grid

## Loading required package: conf.design

## Registered S3 method overwritten by 'DoE.base':  
## method from   
## factorize.factor conf.design

##   
## Attaching package: 'DoE.base'

## The following objects are masked from 'package:stats':  
##   
## aov, lm

## The following object is masked from 'package:graphics':  
##   
## plot.design

## The following object is masked from 'package:base':  
##   
## lengths

set.seed(123)  
#We choose 10 different factors that affects the value of a house  
featureNames=c("A","B","C","D","E","F","G","H","J","K")  
#We have 10 different features and we want to test 16 different combinations  
conbination <- FrF2(nruns = 16,nfactors = 10,factor.names = featureNames)  
#Finally we have the results: in the following chart, 1 means we choose that factor, and -1 refers to not included of a certain factor. We can use the following 16 combinations:  
conbination

## A B C D E F G H J K  
## 1 -1 1 1 1 -1 -1 1 -1 1 -1  
## 2 1 1 1 1 1 1 1 1 1 1  
## 3 -1 1 -1 -1 -1 1 -1 1 1 -1  
## 4 1 -1 1 1 -1 1 -1 1 -1 -1  
## 5 1 -1 -1 1 -1 -1 1 1 1 1  
## 6 1 -1 -1 -1 -1 -1 1 -1 -1 -1  
## 7 1 -1 1 -1 -1 1 -1 -1 1 1  
## 8 -1 -1 1 -1 1 -1 -1 1 1 -1  
## 9 1 1 -1 -1 1 -1 -1 -1 1 1  
## 10 1 1 -1 1 1 -1 -1 1 -1 -1  
## 11 -1 1 1 -1 -1 -1 1 1 -1 1  
## 12 -1 -1 -1 -1 1 1 1 1 -1 1  
## 13 -1 1 -1 1 -1 1 -1 -1 -1 1  
## 14 -1 -1 1 1 1 -1 -1 -1 -1 1  
## 15 1 1 1 -1 1 1 1 -1 -1 -1  
## 16 -1 -1 -1 1 1 1 1 -1 1 -1  
## class=design, type= FrF2

In this fractional factorial design, each factor is included 8 times, which is half of the total number of conbinations.

featureTimes=data.frame(nrow=10,ncol=2)  
featureNames=c("A","B","C","D","E","F","G","H","J","K")  
for (i in seq(1,10)) {  
 name=featureNames[i]  
 featureTimes[i,1]=name  
 featureTimes[i,2]=nrow(conbination[conbination[,name]==1,name])  
}  
colnames(featureTimes)=c("Feature id","Included times")  
featureTimes

## Feature id Included times  
## 1 A 8  
## 2 B 8  
## 3 C 8  
## 4 D 8  
## 5 E 8  
## 6 F 8  
## 7 G 8  
## 8 H 8  
## 9 J 8  
## 10 K 8

And half of the combinations (8 our of 16) include 5 factors. One includes all of the ten factors and oneincludes two factors. Then four include four factors and two include six factors.

featureIn=data.frame(nrow=16,ncol=2)  
for (i in seq(1,16)) {  
 featureIn[i,1]=i  
 featureIn[i,2]=ncol(conbination[i,conbination[i,]==1])  
}  
colnames(featureIn)=c("House No.","Number of included features")  
featureIn

## House No. Number of included features  
## 1 1 5  
## 2 2 10  
## 3 3 4  
## 4 4 5  
## 5 5 6  
## 6 6 2  
## 7 7 5  
## 8 8 4  
## 9 9 5  
## 10 10 5  
## 11 11 5  
## 12 12 5  
## 13 13 4  
## 14 14 4  
## 15 15 6  
## 16 16 5

cat("frequency:\n");table(featureIn[,2])

## frequency:

##   
## 2 4 5 6 10   
## 1 4 8 2 1

# Question 14.1

**The breast cancer data set breast-cancer-wisconsin.data.txt from** [**http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/**](http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/) **(description at** [**http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Original%29**](http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Original%29) **) has missing values. 1. Use the mean/mode imputation method to impute values for the missing data. 2. Use regression to impute values for the missing data. 3. Use regression with perturbation to impute values for the missing data. 4. (Optional) Compare the results and quality of classification models (e.g., SVM, KNN) build using (1) the data sets from questions 1,2,3; (2) the data that remains after data points with missing values are removed; and (3) the data set when a binary variable is introduced to indicate missing values.**

To explore the missing data, we firstly run a frequency table for each column(excluding column 1 which is the ID of each data point). The results show that only one column has missing data and the missing data are represented by “?”.

rawData = read.csv("breast-cancer-wisconsin.data.txt",header=F)  
for (i in seq(2,ncol(rawData))) {  
 print(table(rawData[,i]))  
 print(sum(table(rawData[,i])))  
}

##   
## 1 2 3 4 5 6 7 8 9 10   
## 145 50 108 80 130 34 23 46 14 69   
## [1] 699  
##   
## 1 2 3 4 5 6 7 8 9 10   
## 384 45 52 40 30 27 19 29 6 67   
## [1] 699  
##   
## 1 2 3 4 5 6 7 8 9 10   
## 353 59 56 44 34 30 30 28 7 58   
## [1] 699  
##   
## 1 2 3 4 5 6 7 8 9 10   
## 407 58 58 33 23 22 13 25 5 55   
## [1] 699  
##   
## 1 2 3 4 5 6 7 8 9 10   
## 47 386 72 48 39 41 12 21 2 31   
## [1] 699  
##   
## ? 1 10 2 3 4 5 6 7 8 9   
## 16 402 132 30 28 19 30 4 8 21 9   
## [1] 699  
##   
## 1 2 3 4 5 6 7 8 9 10   
## 152 166 165 40 34 10 73 28 11 20   
## [1] 699  
##   
## 1 2 3 4 5 6 7 8 9 10   
## 443 36 44 18 19 22 16 24 16 61   
## [1] 699  
##   
## 1 2 3 4 5 6 7 8 10   
## 579 35 33 12 6 3 9 8 14   
## [1] 699  
##   
## 2 4   
## 458 241   
## [1] 699

So we import the data again and set “?” as missing data. To summary, there are 16 missing data points in column *Bare Nuclei*.

rawData = read.csv("breast-cancer-wisconsin.data.txt",header=F,na.string="?")  
colnames(rawData)=c("ID","ClumpThickness","UniformityofCellSize",  
 "UniformityofCellShape","MarginalAdhesion",  
 "SingleEpithelialCellSize","BareNuclei",  
 "BlandChromatin","NormalNucleoli",  
 "Mitoses","Class")  
rawData[rawData[,"Class"]==2,"Class"]="benign"  
rawData[rawData[,"Class"]==4,"Class"]="malignant"  
rawData$Class=as.factor(rawData$Class)  
summary(rawData)

## ID ClumpThickness UniformityofCellSize  
## Min. : 61634 Min. : 1.000 Min. : 1.000   
## 1st Qu.: 870688 1st Qu.: 2.000 1st Qu.: 1.000   
## Median : 1171710 Median : 4.000 Median : 1.000   
## Mean : 1071704 Mean : 4.418 Mean : 3.134   
## 3rd Qu.: 1238298 3rd Qu.: 6.000 3rd Qu.: 5.000   
## Max. :13454352 Max. :10.000 Max. :10.000   
##   
## UniformityofCellShape MarginalAdhesion SingleEpithelialCellSize  
## Min. : 1.000 Min. : 1.000 Min. : 1.000   
## 1st Qu.: 1.000 1st Qu.: 1.000 1st Qu.: 2.000   
## Median : 1.000 Median : 1.000 Median : 2.000   
## Mean : 3.207 Mean : 2.807 Mean : 3.216   
## 3rd Qu.: 5.000 3rd Qu.: 4.000 3rd Qu.: 4.000   
## Max. :10.000 Max. :10.000 Max. :10.000   
##   
## BareNuclei BlandChromatin NormalNucleoli Mitoses   
## Min. : 1.000 Min. : 1.000 Min. : 1.000 Min. : 1.000   
## 1st Qu.: 1.000 1st Qu.: 2.000 1st Qu.: 1.000 1st Qu.: 1.000   
## Median : 1.000 Median : 3.000 Median : 1.000 Median : 1.000   
## Mean : 3.545 Mean : 3.438 Mean : 2.867 Mean : 1.589   
## 3rd Qu.: 6.000 3rd Qu.: 5.000 3rd Qu.: 4.000 3rd Qu.: 1.000   
## Max. :10.000 Max. :10.000 Max. :10.000 Max. :10.000   
## NA's :16   
## Class   
## benign :458   
## malignant:241   
##   
##   
##   
##   
##

**1.Use the mean/mode imputation method to impute values for the missing data.**

We use both mean and mode to impute values. But according to the frequency table below, the distribution of *Bare Nuclei* is heavily bi-polared. Most of the data points equal to 1 or 10. We propose that using mode instead of mean seems to be a better option to impute values for the missing data. In question4, we will do further comparation towards mean and mode imputation method.

cat("frequency:");table(rawData[,"BareNuclei"])

## frequency:

##   
## 1 2 3 4 5 6 7 8 9 10   
## 402 30 28 19 30 4 8 21 9 132

# use mode  
dataMode=rawData  
dataMode[is.na(dataMode[,"BareNuclei"])==T,"BareNuclei"]=1  
summary(dataMode[,"BareNuclei"])

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 1.000 1.000 1.000 3.486 5.000 10.000

# use mean  
dataMean=rawData  
dataMean[is.na(dataMean[,"BareNuclei"])==T,"BareNuclei"]=  
 mean(dataMean[,"BareNuclei"], na.rm=TRUE)  
summary(dataMean[,"BareNuclei"])

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 1.000 1.000 1.000 3.545 5.000 10.000

**2.Use regression to impute values for the missing data.**

We use *mice* package to impute values using regression. We use all the variables(excluding *BareNuclei* and *ID*) as predicting variables in the regressiong model to predict the value of *BareNuclei*. Results are showed below. And all the predicting values are in the feasible range of *BareNuclei’s* value(1 to 10).

library(mice)

## Loading required package: lattice

##   
## Attaching package: 'mice'

## The following object is masked from 'package:DoE.base':  
##   
## make.formulas

## The following objects are masked from 'package:base':  
##   
## cbind, rbind

dataReg=rawData  
model1=mice(dataReg[,2:11], method="norm.predict", m=1, maxit=1, seed=1234)

##   
## iter imp variable  
## 1 1 BareNuclei

model1$imp$BareNuclei

## 1  
## 24 7.201509  
## 41 3.412194  
## 140 1.200127  
## 146 1.588095  
## 159 1.271663  
## 165 1.444743  
## 236 1.960806  
## 250 1.407689  
## 276 1.625150  
## 293 6.343076  
## 295 1.219350  
## 298 1.000995  
## 316 2.005965  
## 322 1.407689  
## 412 1.200127  
## 618 1.048844

dataReg[,2:11]=complete(model1)

**3.Use regression with perturbation to impute values for the missing data.**

We also use *mice* package to impute values using regression with pertrbation. We use all the variables(excluding *BareNuclei* and *ID*) as predicting variables in the regressiong model and add random error terms to the predicted value. Results are showed below. We draw a scatter plot to compare the values from regression and values from regression with perturbation. Most of the points are located away from 45 degree line, which indicates the perturbation.

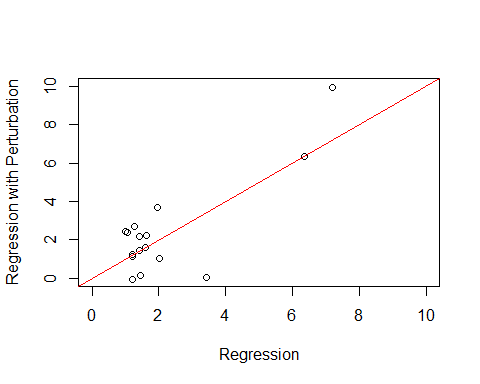
library(mice)  
dataRegPer=rawData  
model2=mice(dataRegPer[,2:11], method="norm.nob", m=1, maxit=1, seed=1234)

##   
## iter imp variable  
## 1 1 BareNuclei

model2$imp$BareNuclei

## 1  
## 24 9.94371380  
## 41 0.03484965  
## 140 -0.05574598  
## 146 1.62475788  
## 159 2.68327455  
## 165 0.14967372  
## 236 3.69855214  
## 250 2.15955873  
## 276 2.24616931  
## 293 6.35309746  
## 295 1.14402987  
## 298 2.45010184  
## 316 1.01169507  
## 322 1.43049705  
## 412 1.21986249  
## 618 2.40646790

plot(model1$imp$BareNuclei[,1], model2$imp$BareNuclei[,1],  
 xlim=c(0,10), ylim=c(0,10),  
 xlab="Regression", ylab="Regression with Perturbation")  
abline(a=0, b=1, col="red")



On the other hand, we notice that there are three values out of the feasible range of *BareNuclei*. We search some literatures and two common ways to deal with such issue are: 1) retaining the values, 2) post-imputation rounding(<https://bmcmedresmethodol.biomedcentral.com/articles/10.1186/1471-2288-14-57>). So we use these two method to deal with the imputed values and compare them in question4.

cat("out of range values:\n");sum(model2$imp$BareNuclei<1 | model2$imp$BareNuclei>10)

## out of range values:

## [1] 3

dataRegPer[,2:11]=complete(model2)  
dataRegPerRetain=dataRegPer  
dataRegPerRound=dataRegPer  
dataRegPerRound[dataRegPerRound$BareNuclei<1,"BareNuclei"]=1  
dataRegPerRound[dataRegPerRound$BareNuclei>10,"BareNuclei"]=10

**4. Compare the results and quality of classification models (e.g., SVM, KNN) build using (1) the data sets from questions 1,2,3; (2) the data that remains after data points with missing values are removed; and (3) the data set when a binary variable is introduced to indicate missing values.**

We firstly creat a data set by removing the missing values. Then we introduce the binary variable to indicate missing values in another data set *dataBinary*. We also introduce the interaction terms between the binary variable with all other independent variable into the data set.

dataRemove=rawData  
dataRemove=na.omit(dataRemove)  
  
dataBinary=rawData  
dataBinary$Missing=rep(1,nrow(dataBinary)) # 0=missing, 1=not  
dataBinary[is.na(dataBinary$BareNuclei)==T, "Missing"]=0  
  
dependents=c("ClumpThickness","UniformityofCellSize",  
 "UniformityofCellShape","MarginalAdhesion",  
 "SingleEpithelialCellSize","BlandChromatin",  
 "NormalNucleoli","Mitoses")  
  
for (col in dependents) {  
 newCol=paste(col,"Interact",sep="")  
 dataBinary[,newCol]=dataBinary[,col]\*dataBinary$Missing  
}  
  
dataBinary[is.na(dataBinary$BareNuclei)==T, "BareNuclei"]=0

So to summarize, the approach of dealing with missing data and the corresponding data set we use are:  
1). Replace by mode: dataMode  
2). Replace by mean: dataMean  
3). Use regression to impute values for the missing data: dataReg  
4). Use regression with perturbation to impute values for the missing data: dataRegPerRetain  
5). Use regression with perturbation to impute values for the missing data and round the values that is out of range to the nearest bound: dataRegPerRound  
6). Remove data points with missing values: dataRemove  
7). Intorduce binary variable to indicate missing values: dataBinary

We use KNN model to compare the results and quality of these approaches. We split the data into training set and test set. To ensure the results is comparable, the rows of data point in training and in test among different data set is identical. The only exception is *dataRemove*, in which several rows are removed. Our method is to split the data set according to *ID*. For *dataRemove*, the way to split it is also to check whether the *ID* is in training list or test list.

# sampling ID for training set  
trainingSize=floor(0.75\*nrow(rawData))  
set.seed(1234)  
trainRows=sample(x=rawData[,"ID"],size=trainingSize,replace=F)

We use *train.kknn* function to train our KNN model, which apply leave-one-out approach to find the best combination of k and kernel. We build a function upon *train.kknn*, using data set as input and outputing best k and kernel and model quality evaluation via test set (predicting accuracy).

library(kknn)  
knnRun = function(data) {  
 trainset=data[data$ID %in% trainRows==T,2:ncol(data)]  
 testset=data[data$ID %in% trainRows==F,2:ncol(data)]  
 modelTrain=train.kknn(Class~., trainset, kmax=15,   
 kernel= c("rectangular", "triangular", "epanechnikov",   
 "gaussian", "rank", "optimal"))  
 testResults=predict(modelTrain, testset)  
 return (list("k"=modelTrain$best.parameters$k,   
 "kernel"=modelTrain$best.parameters$kernel,  
 "accuracy"=sum(testResults==testset$Class)/nrow(testset)))  
}  
  
# use different data set and make comparison  
r=data.frame(ncol=4,nrow=7)  
  
r[1,1]="dataMode"  
r[1,2]=knnRun(dataMode)$k  
r[1,3]=knnRun(dataMode)$kernel  
r[1,4]=knnRun(dataMode)$accuracy  
  
r[2,1]="dataMean"  
r[2,2]=knnRun(dataMean)$k  
r[2,3]=knnRun(dataMean)$kernel  
r[2,4]=knnRun(dataMean)$accuracy  
  
r[3,1]="dataReg"  
r[3,2]=knnRun(dataReg)$k  
r[3,3]=knnRun(dataReg)$kernel  
r[3,4]=knnRun(dataReg)$accuracy  
  
r[4,1]="dataRegPerRetain"  
r[4,2]=knnRun(dataRegPerRetain)$k  
r[4,3]=knnRun(dataRegPerRetain)$kernel  
r[4,4]=knnRun(dataRegPerRetain)$accuracy  
  
r[5,1]="dataRegPerRound"  
r[5,2]=knnRun(dataRegPerRound)$k  
r[5,3]=knnRun(dataRegPerRound)$kernel  
r[5,4]=knnRun(dataRegPerRound)$accuracy  
  
r[6,1]="dataRemove"  
r[6,2]=knnRun(dataRemove)$k  
r[6,3]=knnRun(dataRemove)$kernel  
r[6,4]=knnRun(dataRemove)$accuracy  
  
r[7,1]="dataBinary"  
r[7,2]=knnRun(dataBinary)$k  
r[7,3]=knnRun(dataBinary)$kernel  
r[7,4]=knnRun(dataBinary)$accuracy  
  
colnames(r)=c("dataset", "best k", "best kernel", "accuracy")  
  
r

## dataset best k best kernel accuracy  
## 1 dataMode 15 rectangular 0.9679487  
## 2 dataMean 7 rectangular 0.9615385  
## 3 dataReg 15 rectangular 0.9679487  
## 4 dataRegPerRetain 15 rectangular 0.9679487  
## 5 dataRegPerRound 15 rectangular 0.9679487  
## 6 dataRemove 11 rectangular 0.9610390  
## 7 dataBinary 9 rectangular 0.9487179

According to the results above, generally the difference between every missing data imputing approach is small based on our raw data set. We believe it’s plausible because there are only 16 missing values in one predictor, which is really a small proportion of the whole data set (699 observations with 9 predictors). If we want to compare between different imputation approaches, a data set with a bit more missing values maybe useful. But we can also recognize two meaningful points in the results. The accuracy of imputing by mean is relatively lower comparing to other approaches. Because the distribution of *BareNuclei* is bi-polar with 402 ones and 132 tens. Mean value is not a good representive of the variable. Second, removing data points with missing values also generates a lower results. We check all independent variables’ mean values as well as the distribution of response variable *Class* between missing-value group and none-missing gourp. From the table below we can see that the mean values in missing-value group are relatively low. Besides, the distribution among *Class Benigh* and *Class Malignant* is different. So the data points with missing values may not be random. There seems existing some systematic difference between observations with missing values and other observations. Removing them directly may not be a good choice.

groupcompare=data.frame(nrow=11, ncol=3)  
var=c("ClumpThickness","UniformityofCellSize","UniformityofCellShape",  
 "MarginalAdhesion","SingleEpithelialCellSize", "BareNuclei",  
 "BlandChromatin","NormalNucleoli","Mitoses")  
for (i in seq(1,length(var))){  
 groupcompare[i,1]=var[i]  
 groupcompare[i,2]=mean(rawData[is.na(rawData$BareNuclei)==T,var[i]])  
 groupcompare[i,3]=mean(rawData[is.na(rawData$BareNuclei)==F,var[i]])  
}  
  
groupcompare[10,1]="Num.of benigh"  
groupcompare[10,2]=sum(rawData[is.na(rawData$BareNuclei)==T,"Class"]=="benign")  
groupcompare[10,3]=sum(rawData[is.na(rawData$BareNuclei)==F,"Class"]=="benign")  
  
groupcompare[11,1]="Num.of malignant"  
groupcompare[11,2]=sum(rawData[is.na(rawData$BareNuclei)==T,"Class"]=="malignant")  
groupcompare[11,3]=sum(rawData[is.na(rawData$BareNuclei)==F,"Class"]=="malignant")  
  
colnames(groupcompare)=c("variables", "missing-value group", "none-missing group")  
  
groupcompare

## variables missing-value group none-missing group  
## 1 ClumpThickness 3.3750 4.442167  
## 2 UniformityofCellSize 2.4375 3.150805  
## 3 UniformityofCellShape 2.8750 3.215227  
## 4 MarginalAdhesion 1.8125 2.830161  
## 5 SingleEpithelialCellSize 2.4375 3.234261  
## 6 BareNuclei NA 3.544656  
## 7 BlandChromatin 3.1250 3.445095  
## 8 NormalNucleoli 2.7500 2.869693  
## 9 Mitoses 1.0000 1.603221  
## 10 Num.of benigh 14.0000 444.000000  
## 11 Num.of malignant 2.0000 239.000000