setwd("C:\\Users\\Big data\\Desktop\\MRI\_xgboost")

data2<- read.table("MR\_meanencoding.csv",header=T, sep=",", fileEncoding='utf-8')

data=data2[ , c("ITEM\_n","ORDERDR\_n","DEPT\_n","SEX\_n","AGE","total","POS\_n","PLACE\_n","IO\_n")]

#取20% testdata最後用

n=0.2\*nrow(data)

test.index=sample(1:nrow(data),n)

Train = data [-test.index,]

Test = data[test.index,]

#將Train 分成3份

n = 3

n.folds = rep(1:n, each=nrow(Train)/n) #分3組

train.folds = split(Train, n.folds)

#第一模型用多元回歸

meta.Train\_x = vector()

meta.Test\_x = list()

#把2份作為train,1份作為test,test出來的y\_hat最後要當x用

#多元回歸第一次

stacking.train = rbind(train.folds[[2]], train.folds[[3]])

stacking.valid = train.folds[[1]]

model\_1 = lm(total~., stacking.train) #Y1\_hat

tmp.meta.Train\_x = predict(model\_1, stacking.valid)

meta.Train\_x = c(meta.Train\_x, tmp.meta.Train\_x) #一開始meta.x是空的,加入第一次的y\_hat

#把TEST的資料一併處理

stacking.test = Test

tmp.meta.Test\_x = predict(model\_1, stacking.test) #預測 Test

meta.Test\_x[[1]] = tmp.meta.Test\_x

#多元回歸第二次

stacking.train = rbind(train.folds[[1]], train.folds[[3]])

stacking.valid = train.folds[[2]]

model\_1 = lm(total~., stacking.train) #Y2\_hat

tmp.meta.Train\_x = predict(model\_1, stacking.valid)

meta.Train\_x = c(meta.Train\_x, tmp.meta.Train\_x) #加入Y2\_hat

#把TEST的資料一併處理

stacking.test = Test

tmp.meta.Test\_x = predict(model\_1, stacking.test) #預測 Test

meta.Test\_x[[2]] = tmp.meta.Test\_x

#多元回歸第三次

stacking.train = rbind(train.folds[[1]], train.folds[[2]])

stacking.valid = train.folds[[3]]

model\_1 = lm(total~., stacking.train) #Y3\_hat

tmp.meta.Train\_x = predict(model\_1, stacking.valid)

meta.Train\_x = c(meta.Train\_x, tmp.meta.Train\_x)

#把TEST的資料一併處理

stacking.test = Test

tmp.meta.Test\_x = predict(model\_1, stacking.test) #預測 Test

meta.Test\_x[[3]] = tmp.meta.Test\_x

#把3個y\_hat的結果做成dataframe,放入y

meta.train.1 = data.frame(meta\_x = meta.Train\_x, y=Train$total)

#Test的y\_hat取平均值,做成dataframe,放入y

mean.meta.Test\_x = (meta.Test\_x[[1]] + meta.Test\_x[[2]] + meta.Test\_x[[3]]) / 3

meta.test.1 = data.frame(meta\_x = mean.meta.Test\_x, y= Test$total)

#第二模型用SVR

library(e1071)

meta.Train\_x = vector()

meta.Test\_x = list()

#SVR第一次

stacking.train = rbind(train.folds[[2]], train.folds[[3]])

stacking.valid = train.folds[[1]]

stacking.test = Test

model\_2 = svm(total~., stacking.train)

tmp.meta.Train\_x = predict(model\_2, stacking.valid)

tmp.meta.Test\_x = predict(model\_2, stacking.test)

meta.Train\_x = c(meta.Train\_x, tmp.meta.Train\_x)

meta.Test\_x[[1]] = tmp.meta.Test\_x

#SVR第二次

stacking.train = rbind(train.folds[[1]], train.folds[[3]])

stacking.valid = train.folds[[2]]

stacking.test = Test

model\_2 = svm(total~., stacking.train)

tmp.meta.Train\_x = predict(model\_2, stacking.valid)

tmp.meta.Test\_x = predict(model\_2, stacking.test)

meta.Train\_x = c(meta.Train\_x, tmp.meta.Train\_x)

meta.Test\_x[[2]] = tmp.meta.Test\_x

#SVR第三次

stacking.train = rbind(train.folds[[1]], train.folds[[2]])

stacking.valid = train.folds[[3]]

stacking.test = Test

model\_2 = svm(total~., stacking.train)

tmp.meta.Train\_x = predict(model\_2, stacking.valid)

tmp.meta.Test\_x = predict(model\_2, stacking.test)

meta.Train\_x = c(meta.Train\_x, tmp.meta.Train\_x) #一開始meta.x是空的

meta.Test\_x[[3]] = tmp.meta.Test\_x

#Test的y\_hat取平均值,做成dataframe,放入y

mean.meta.Test\_x = (meta.Test\_x[[1]] + meta.Test\_x[[2]] + meta.Test\_x[[3]]) / 3

# Mata-Train DataSet

meta.train.2 = data.frame(meta\_x = meta.Train\_x, y=Train$total)

meta.test.2 = data.frame(meta\_x = mean.meta.Test\_x, y= Test$total)

#第三模型用CART

library(rpart)

meta.Train\_x = vector()

meta.Test\_x = list()

#CART第一次

stacking.train = rbind(train.folds[[2]], train.folds[[3]])

stacking.valid = train.folds[[1]]

stacking.test = Test

model\_3 = rpart(total~., stacking.train)

tmp.meta.Train\_x = predict(model\_3, stacking.valid)

tmp.meta.Test\_x = predict(model\_3, stacking.test)

meta.Train\_x = c(meta.Train\_x, tmp.meta.Train\_x) #一開始meta.x是空的

meta.Test\_x[[1]] = tmp.meta.Test\_x

#CART第二次

stacking.train = rbind(train.folds[[1]], train.folds[[3]])

stacking.valid = train.folds[[2]]

stacking.test = Test

model\_3 = rpart(total~., stacking.train)

tmp.meta.Train\_x = predict(model\_3, stacking.valid)

tmp.meta.Test\_x = predict(model\_3, stacking.test)

meta.Train\_x = c(meta.Train\_x, tmp.meta.Train\_x)

meta.Test\_x[[2]] = tmp.meta.Test\_x

#CART第三次

stacking.train = rbind(train.folds[[1]], train.folds[[2]])

stacking.valid = train.folds[[3]]

stacking.test = Test

model\_3 = rpart(total~., stacking.train)

tmp.meta.Train\_x = predict(model\_3, stacking.valid)

tmp.meta.Test\_x = predict(model\_3, stacking.test)

meta.Train\_x = c(meta.Train\_x, tmp.meta.Train\_x)

meta.Test\_x[[3]] = tmp.meta.Test\_x

#Test的y\_hat取平均值,做成dataframe,放入y

mean.meta.Test\_x = (meta.Test\_x[[1]] + meta.Test\_x[[2]] + meta.Test\_x[[3]]) / 3

meta.test.3 = data.frame(meta\_x = mean.meta.Test\_x, y= Test$total)

# Mata-Train DataSet

meta.train.3 = data.frame(meta\_x = meta.Train\_x, y=Train$total)

#以下個人認為鐘老師的方法有問題,rbind之後只有1個X

#allitems.meta.train = rbind(meta.train.1, meta.train.2, meta.train.3)

#故用R老師的方式來做:把前面三個模型的Y\_hat當做X來訓練,有3個X

#MAPE(test)= 26.1516 %

library(xgboost)

#先把三個 Meta-Train合併一起

test\_data=data.frame(x1=meta.test.1[,"meta\_x"],x2=meta.test.2[,"meta\_x"],x3=meta.test.3[,"meta\_x"],"total"=meta.test.1[,"y"])

train\_data=data.frame(x1=meta.train.1[,"meta\_x"],x2=meta.train.2[,"meta\_x"],x3=meta.train.3[,"meta\_x"],"total"=meta.train.1[,"y"])

XDtest=test\_data

XDtrain=train\_data

XDtrain\_y=XDtrain[,'total']

XDtrain=XDtrain[,which(names(XDtrain)!="total")]

XDtest\_y = XDtest[,'total']

XDtest = XDtest[,which(names(XDtest)!="total")]

dtrain <- xgb.DMatrix(data.matrix(XDtrain),label=XDtrain\_y)

dtest <- xgb.DMatrix(data.matrix(XDtest),label=XDtest\_y)

watchlist <- list(train=dtrain , eval=dtest)

xgb.fit <- xgb.train(objective = "reg:linear",data = dtrain,

#label = XDtrain$total,

learning\_rate = 0.05,

nrounds = 50, #item0=33 , item3=41

watchlist = watchlist

)

#---

#檢查overfitting

evalue\_log <-xgb.fit$evaluation\_log

plot(evalue\_log$iter, evalue\_log$train\_rmse, col='blue')

lines(evalue\_log$iter, evalue\_log$eval\_rmse, col='red')

#---

xgb.pred <- predict(xgb.fit, dtest)

y = XDtest\_y

y\_hat <- xgb.pred

test.MAPE=mean(abs(y-y\_hat)/y)

cat("MAPE(test)=",test.MAPE\*100,"%\n")

R2=1-(sum((y-y\_hat)\*\*2)/sum((y-mean(y))\*\*2))

R2

MAEPmean=mean(abs(y-mean(y))/y)

MAEPmean

#重要係數X

# importanceRaw = xgb.importance(colnames(dtrain), model = xgb.fit)

# importanceRaw

# xgb.plot.importance(importance\_matrix = importanceRaw)

#計算射中率

test=abs(y-y\_hat)/y

shot=sapply(test,function(x){

if(x<0.1){

return(1)

}else{

return(0)

}

})

shot\_t=as.data.frame(table(shot))

shot\_t[2,"Freq"]/sum(shot\_t[,"Freq"])

#Stacking

allitems.meta.train = rbind(meta.train.1, meta.train.2, meta.train.3)

allitems.meta.test = rbind(meta.test.1, meta.test.2, meta.test.3)

train\_matrix = xgb.DMatrix(data = as.matrix(allitems.meta.train[,1]), label = allitems.meta.train[, 2])

test\_matrix = xgb.DMatrix(data = as.matrix(allitems.meta.test[,1]) , label = allitems.meta.test[, 2])

xgb.params = list(

#col的抽樣比例，越高表示每棵樹使用的col越多，會增加每棵小樹的複雜度

colsample\_bytree = 0.7,

# row的抽樣比例，越高表示每棵樹使用的col越多，會增加每棵小樹的複雜度

subsample = 0.5,

booster = "gbtree",

# 樹的最大深度，越高表示模型可以長得越深，模型複雜度越高

max\_depth = 6,

# boosting會增加被分錯的資料權重，而此參數是讓權重不會增加的那麼快，因此越大會讓模型愈保守

eta = 0.03,

# 預測問題，用'mae'也可以

eval\_metric = "rmse",

objective = "reg:linear", #預測問題，所以使用regression

# 越大，模型會越保守，相對的模型複雜度比較低

gamma = 0

)

cv.model = xgb.cv(

params = xgb.params,

data = train\_matrix,

nfold = 5, # 5-fold cv

nrounds=50, # 測試1-500，各個樹總數下的模型

# 如果當nrounds < 50 時，就已經有overfitting情況發生，那表示不用繼續tune下去了，可以提早停止

early\_stopping\_rounds = 50,

print\_every\_n = 20 # 每20個單位才顯示一次結果

)

log = cv.model$evaluation\_log

plot(x=1:nrow(log), y= log$train\_rmse\_mean, col='red', xlab="nround", ylab="rmse", main="Check Overfitting")

points(x=1:nrow(log), y= log$test\_rmse\_mean, col='blue')

legend("topright", pch=1, col = c("red", "blue"),

legend = c("Train", "Validation") )

best.nrounds = cv.model$best\_iteration

xgb.model = xgb.train(paras = xgb.params,

data = train\_matrix,

nrounds = best.nrounds)

dtest.1 = xgb.DMatrix(data = as.matrix(meta.test.1[,1]), label = meta.test.1[, 2])

final\_1 = predict(xgb.model, dtest.1)

dtest.2 = xgb.DMatrix(data = as.matrix(meta.test.2[,1]), label = meta.test.2[, 2])

final\_2 = predict(xgb.model, dtest.2)

dtest.3 = xgb.DMatrix(data = as.matrix(meta.test.3[,1]), label = meta.test.3[, 2])

final\_3 = predict(xgb.model, dtest.3)

final\_y = (final\_1 + final\_2 + final\_3)/3

test.MAPE=mean(abs(Test$total-final\_y)/Test$total)

cat("MAPE(test)=",test.MAPE\*100,"%\n")