Risk Prediction using Machine Learning

Meng Lan

Objective

The aim is to distinguish between the presence and absence of cardiac arrhythmia and to classify it in one of the 16 groups based on 12-lead ECG (or Electrocardiography) measurements on patients. Taking the cardiologist as a gold standard the goal is to minimize this difference by means of machine learning tools. Hence, it is a supervised learning problem for classification.

Data cleaning

First recode variable Class to binary variable, 01 indicates the absence of cardiac arrhythmia and 02-16 indicates the presence of cardiac arrhythmia. Then I turn "?" to NA and delete columns whose most values are NA (V14). Finally remove observations with missing values and unbalaced columns (whose 90% values are zero). The processed data set contains 420 observations and 156 variables.

Method

The data set is randomly split into two parts using 1:1 ratio: training and testing set. Fit models using different machine learning methods on training set, and use misclassification rate as measurement on testing set.

1. Consider the machine learning models and rationale that can be and cannot be applied to this problem.

1) methods that can be applied: KNN, logistic regression with elasticnet regularization, classfication tree, bagging, random forest, boosting (gradient boosted model & adaboost model)

Rationale for using these methods:

KNN: As a non-parametric approach, no assumptions are made about the shape of the decision boundary so KNN is included as one of the methods. k is selected from 1 to 10 and I choose the k with smallest test error.

logistic regression with elasticnet regularization: Logistic regression is a popular method especially when there are two classes. Because the number of training set and the number of features are similar, the least squares fit may have high variance and result in over fitting. Hence, variable selection is employed using elasticnet regularization because shrinking the coefficient estimates can significantly reduce the variance.

decision tree: It is a simple and easy-to-interpret method by partitioning the predictor space into a number of simple regions.

bagging: Basing on classfication tree, bagging generates a number of bootstrapped training data sets and aggregates them to reduces variances to improve prediction accuracy.

random forest: It is similar to bagging but it differs in that only a part of variables are randomly selected in each split, which lowers the corelation of the trees.

boosting: It similarly grows multiple trees, except that the trees are grown sequentially: each tree is grown using information from previously grown trees so trees are independent. Adaboost is similar but it progressively increases the weights of the wrongly predicted instances and decreases the ones of the correctly predicted instances to improve prediction accuracy.

In all tree-based methods, I choose the number of trees to be 300 which is large enough and dose not cause too much computational issue.

2) methods that I did not choose：

LDA & QDA: All features should follow normal distribution in each class but some are categorical variables (e.g. sex) and not all are normally distributed in this dataset.

Naive Bayes: Obviously there are some features that are not independent (e.g. QRS and QRST) which violates the assumption of naive bayes.

Polynomials, step functions, splines & general addictive model: The number of features is so large that it is not appropriate to use these unlinear methods.

Regarding to variable selection, not choosing best subset is because of computational issue and not stepwise is because it dose not guaranting the best model.

2. Compare the performance of these methods in predicting the presence or absence of cardiac arrhythmia.

Prediction Error of Different Methods

knn	ridge	tree	bagging	randomForest	gbm	adaboost
0.348	0.319	0.205	0.143	0.205	0.243	0.205

The table for comparing all these methods shows that in predicting the presence or absence of cardiac arrhythmia, bagging has the best performance with 14.3% misclassification error. The other tree-based methods perform similarly with approximately 20% misclassification error. KNN and ridge regression has the worst performance with over 30% misclassification error.

3. **Finally, choose the best performing method**

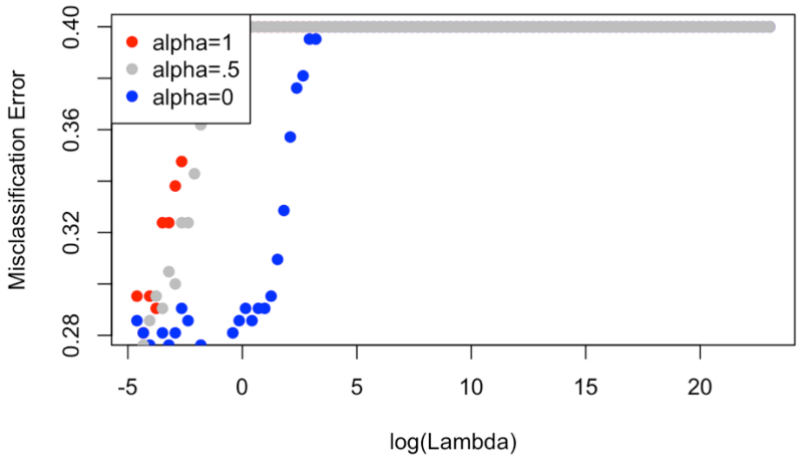
Among all these methods, I choose bagging because it reaches 85.7% accuracy. Comparing the results in original research article, which outperforms the original VF15 algorithm with only 62% accuracy.

Appendix

1. Tables and graphics

Prediction Error of KNN

	error rate
knn1	0.3952381
knn2	0.3809524
knn3	0.3476190
knn4	0.3666667
knn5	0.3714286
knn6	0.3761905
knn7	0.3809524
knn8	0.3809524
knn9	0.3857143
knn10	0.3857143



prediction error for different alpha in logistic regression with elastic net regularization

2. Codes

```
library(ade4)
library(glmnet)
library(tidyverse)
library(class)
library(knitr)
```

```
library(tree)
library(randomForest)
library(gbm)
library(caret)
library(e1071)
# load data
arrhy <- read.csv("/Users/lanmeng/Desktop/arrhythmia_data.csv",header = F)
# clean data
colnames(arrhy)[280] <- "Class"
arrhy$Class <- ifelse(arrhy$Class==1,0,1)
arrhy.na <- arrhy
for (i in 1:280){
  arrhy.na[,i] <- ifelse(arrhy.na[,i] == "?", NA, arrhy.na[,i])
}
for (i in 1:280) {
  print(sum(is.na(arrhy.na[,i])))
}
arrhy.na <- arrhy.na[,-14]
arrhy.nona <- arrhy.na[complete.cases(arrhy.na),]
noc <- which(colSums(arrhy.nona==0)>0.9*nrow(arrhy.nona))
arrhy.data <- arrhy.nona[,-noc]
dim(arrhy.data)
arrhy.data$Class <- as.factor(arrhy.data$Class)
#split data
smp_size <- floor(0.5 * nrow(arrhy.data))
set.seed(123)
ind <- sample(seq_len(nrow(arrhy.data)), size = smp_size)
train <- arrhy.data[ind,]
```

```
test <- arrhy.data[-ind,]

#1.knn
KNN.error <- function(Train, Test, k=1){
Train.KNN <- scale(Train[, which(names(Train) !="Class")])
Test.KNN <- scale(Test[,which(names(Test) !="Class")])
Test.class <- Test$Class
Train.class <- Train$Class
set.seed(1)
pred.class <- knn(Train.KNN, Test.KNN, Train.class, k=k)
mean(Test.class != pred.class)}

error_df <- matrix(0, nrow=10, ncol=1) %>%
        as_tibble()
KNN.error(train, test,1)
for (i in 1:10){
 errs <- KNN.error(train, test,i)
 error_df[i,1] <- errs
}
error_df <- as.data.frame(error_df)
rownames(error_df) <- paste0("knn",1:10)
colnames(error_df) <- "error rate"
error_df_table <- kable(error_df,caption = "Prediction Error of KNN",align = "c")
error_df_table
min(error_df)

### 2: logistic with elasticnet regularization
#y & x in train
```

```
arrhy_y <- arrhy.data$Class

arrhy_x <- model.matrix(Class~.,arrhy.data)[,-1]

y <- arrhy_y[ind]

x <- arrhy_x[ind,]

newx <- arrhy_x[-ind,]

k <- 10

set.seed(123)

folds <- sample(1:k,nrow(train),replace =TRUE)

#determine alpha

grid =10^seq(10,-2,length =100)

cv1=cv.glmnet(x,y,type.measure="class",family="binomial",lambda=grid,foldid=folds,alpha
=1)

cv.5=cv.glmnet(x,y,type.measure="class",family="binomial",lambda=grid,foldid=folds,alpha
=.5)

cv0=cv.glmnet(x,y,type.measure="class",family="binomial",lambda=grid,foldid=folds,alpha
=0)

plot(log(cv1$lambda),cv1$cvm,pch=19,col="red",xlab="log(Lambda)",ylab=cv1$name)

points(log(cv0$lambda),cv0$cvm,pch=19,col="blue")

points(log(cv.5$lambda),cv.5$cvm,pch=19,col="grey")

legend("topleft",legend=c("alpha=1","alpha=.5","alpha=0"),pch=19,col=c("red","grey","blue
"))

a <- 0

#cv determine lambda

enet.cv <- cv.glmnet(x,y,type.measure="class",family="binomial",foldid = folds,alpha=a)

lambda.chosen <- enet.cv$lambda.1se

enet.mod <- glmnet(x,y,lambda = lambda.chosen,alpha=a,family = "binomial")

# prediction errors

pred.enet.class <- predict(enet.mod,newx,type="response")

pred.enet.class <- ifelse(pred.enet.class > 0.5, "1", "0")

enet.errors <- mean(test$Class != pred.enet.class)
```

```
enet.errors


# 3. classification tree
set.seed(123)
mytree <- tree(Class ~ .,data=train, method="gini")
mytree.cv <- cv.tree(mytree,FUN=prune.misclass,K=10)
final.tree <-
prune.tree(mytree,best=mytree.cv$size[mytree.cv$dev==min(mytree.cv$dev)])
mypredict.class <- predict(final.tree,newdata=test, type="class")
tmp.class.tree <- table(mypredict.class,test$Class)
tmp.class.tree
misclass.class.tree <- 1-sum(diag(tmp.class.tree)/sum(tmp.class.tree))
misclass.class.tree


#4. bagging
p <- ncol(train)-1
y <- ncol(train)
n.tree <- 300
set.seed(123)
bagged.tree <- randomForest(Class~.,data=train,mtry=p,ntree=n.tree)


mypredict.bagg <- predict(bagged.tree, test, type="class")
tmp.bagg.tree <- table(mypredict.bagg,test$Class)
tmp.bagg.tree


misclass.bagg.tree <- 1-sum(diag(tmp.bagg.tree)/sum(tmp.bagg.tree))
misclass.bagg.tree
```

```
#5. Random Forest
set.seed(123)
rf.tree <- randomForest(Class~.,data=train,mtry=ceiling(sqrt(p)),ntree=n.tree)
mypredict.rf <- predict(rf.tree, test, type="class")
tmp.rf.tree <- table(mypredict.rf,test$Class)
tmp.rf.tree

misclass.rf.tree <- 1-sum(diag(tmp.rf.tree)/sum(tmp.rf.tree))
misclass.rf.tree

### Boosting
##gbm
Grid <- expand.grid(interaction.depth=c(1, 2), n.trees = n.tree,
                    shrinkage=c(0.01, 0.001),
                    n.minobsinnode=c(5,10))
metric <- "Accuracy"
trainControl <- trainControl(method="cv", number=5)
set.seed(99)
grd <- train(Class ~ ., data=train, distribution="bernoulli", method="gbm",
             trControl=trainControl, verbose=FALSE,
             tuneGrid=Grid, metric=metric)

#The final parameters used for the model were n.trees = 300,interaction.depth = 1,
shrinkage = 0.01 and n.minobsinnode = 5
##ada
set.seed(99)
ada <- train(Class ~ ., data=train, distribution="adaboost", method="gbm",
             trControl=trainControl, verbose=FALSE,
```

```
        tuneGrid=Grid, metric=metric)

#The final parameters used for the model were n.trees = 300,interaction.depth = 2,
shrinkage = 0.01 and n.minobsinnode = 5


##final model

train$Class.int = as.integer(train$Class)-1

train <- train[,-y]

grd.boost.final <- gbm(Class.int~.-Class.int,data=train,distribution
="bernoulli",n.trees=n.tree,interaction.depth = 1,shrinkage=0.01,n.minobsinnode = 5)


mypred.grd<-predict(grd.boost.final,newdata = test,n.trees=n.tree,type="response")

pred.grd.class <- ifelse(mypred.grd > 0.5, "1","0")

boost.test.error<-mean(pred.grd.class != test$Class)

boost.test.error


ada.boost.final <- gbm(Class.int~.-Class.int,data=train,distribution
="adaboost",n.trees=n.tree,interaction.depth = 2,shrinkage=0.01,n.minobsinnode = 5)

mypred.ada<-predict(ada.boost.final,newdata = test,n.trees=n.tree,type="response")

pred.class.ada <- ifelse(mypred.ada > 0.5, "1", "0")

ada.test.error<-mean(pred.class.ada != test$Class)

ada.test.error


## final comparison

error.compare <-
data.frame(knn=min(error_df),ridge=enet.errors,tree=misclass.class.tree,bagging=misclass
.bagg.tree,randomForest=misclass.rf.tree,gbm=boost.test.error,adaboost=ada.test.error)

kable(error.compare,digits=3, caption = "Prediction Error of Different Methods",align = "c")
```