How to use health indicators to distinguish the degree of physical fitness

Yuhong Wang

2022-12-29

Github Livepage

12/28/2022

Introduction

After the impact of COVID-19, people are paying more and more attention to health. The question is if we can use the machine learning method to classify people's health through health indicators. Then calculate the health level by monitoring health indicators and then issue warnings to people with low health levels.

Data

I. Data sources

The data set was found on Kaggle, an online source of datasets. The set we selected is titled "Worldwide Deaths by Country/Risk Factors," and the data is sourced from the Korea Sports Promotion Foundation(KSPO). This dataset contains 13,393 observations with 11 different feature variables. The number of cases for class A is 3348, for class B is 3347, for class C is 3349, and for class D is 3349. Thus, balancing data to make every group occupy a similar proportion is unnecessary.

II. Data preprocessing

Features per case:

```r
grade <- data.frame(
  names = c("X1",'gender', "X2", "X3","X4", "X5","X6", "X7","X8", "X9","X10",
 "X11"),
  meaning = c("age : 20 ~64 ",'F,M(deleted)', 'height_cm : (If you want to co
nvert to feet, divide by 30.48)', 'weight_kg','body fat_%','diastolic : diast
olic blood pressure (min)','systolic : systolic blood pressure (min)','gripFo
rce','sit and bend forward_cm','sit-ups counts','broad jump_cm','class : A,B,
C,D ( A: best)')
)
knitr::kable(grade)
```

```

|names  |meaning                                                          |
|:------|:----------------------------------------------------------------|
|X1     |age : 20 ~64                                                     |
|gender |F,M(deleted)                                                     |
|X2     |height_cm : (If you want to convert to feet, divide by 30.48)    |
|X3     |weight_kg                                                        |
|X4     |body fat_%                                                       |
|X5     |diastolic : diastolic blood pressure (min)                       |
|X6     |systolic : systolic blood pressure (min)                         |
|X7     |gripForce                                                        |
|X8     |sit and bend forward_cm                                          |
|X9     |sit-ups counts                                                   |
|X10    |broad jump_cm                                                    |
|X11    |class : A,B,C,D ( A: best)                                       |

```
bp = read.csv("G:/UH/S1/6350 Statistical Learning and Data Mining/Final/archi
ve/bodyPerformance.csv",head = TRUE)
bp = subset(bp,select = -c(2))

D1 = bp[bp$class =="A",]
D2 = bp[bp$class =="B",]
D3 = bp[bp$class =="C",]
D4 = bp[bp$class =="D",]
# Discard the missing data
#Removing rows with a null or " "
D1[D1==""]<-NA
D1[D1==" "]<-NA
D1<-D1[complete.cases(D1),]

D2[D2==""]<-NA
D2[D2==" "]<-NA
D2<-D2[complete.cases(D2),]

D3[D3==""]<-NA
D3[D3==" "]<-NA
D3<-D3[complete.cases(D3),]

D4[D4==""]<-NA
D4[D4==" "]<-NA
D4<-D4[complete.cases(D4),]

D1=na.omit(D1) # Discard the "NA"
D2=na.omit(D2)
D3=na.omit(D3)
```
```

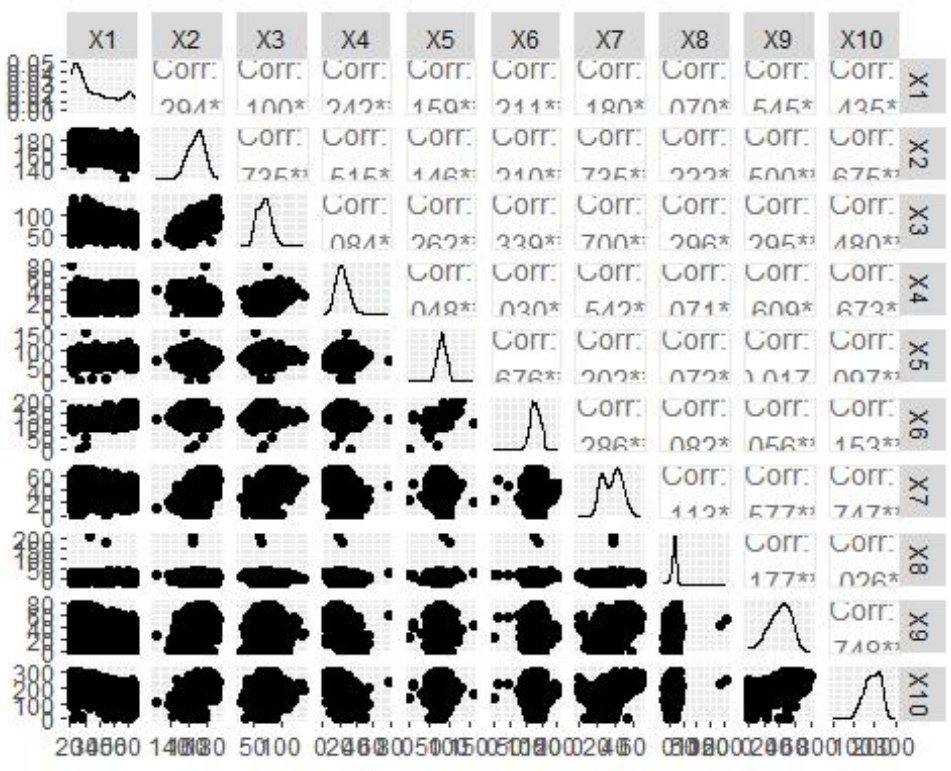
```
D4=na.omit(D4)

D1[,11] = 1
D2[,11] = 2
D3[,11] = 3
D4[,11] = 4

a = dim(D1)[1]
b = dim(D2)[1]
c = dim(D3)[1]
d = dim(D4)[1]

#I don't deed to balance my data
N = a +b + c + d
per_D1 = round(a/N*100,2)
per_D2 = round(b/N*100,2)
per_D3 = round(c/N*100,2)
per_D4 = round(d/N*100,2)

paste('The total number of cases N =',N)

## [1] "The total number of cases N = 13393"

paste('per_D1 =', per_D1, '%')

## [1] "per_D1 = 25 %"

paste('per_D2 =', per_D2, '%')

## [1] "per_D2 = 24.99 %"

paste('per_D3 =', per_D3, '%')

## [1] "per_D3 = 25.01 %"

paste('per_D4 =', per_D4, '%')

## [1] "per_D4 = 25.01 %"
```

Delete the “gender” feature because it is not a quantitative feature. Now cases have ten features. Dataset ‘bodyPerformance.csv’ was imported into R using the read.csv() function. The gender column was discarded, and the rest data was defined as matrix D (11 columns and 13393 rows, all rows are numeric). To center and scale the data, calculate the mean and standard deviation(data - mean)/(std). The scaled data frame of variables(X1-X10) is named RESF.

As the below figure shows, the correlation between variables could be a lot higher, for example, 0.9. Because cases have few features, using PCA to reduce dimension is not helpful.

```
D = rbind(D1,D2,D3,D4)
num=c(1:11)
for(i in num){
```

```
  names(D)[i]=paste("X",i,sep="")
}

library("broom")
library("ggplot2")
library("GGally")

## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2

variables = subset(D,select = -c(11))

pairs = ggpairs(variables)



RESF<-D[,1:10]
# Compute the means m_j and standard deviations std_j
m_j<-vector()
for(j in 1:10){
  m_j=round(c(m_j,mean(RESF[,j])),2)
}

std_j<-vector()
for(j in 1:10){
  std_j=round(c(std_j,sd(RESF[,j])),2)
}
RESF<-D[,1:10]
for(j in 1:10 ){
  RESF[,j]=(D[,j]-m_j[j])/std_j[j]
}

x = RESF
y = factor(D$X11)
```



Correlation between variables

Data analysis

I.Random Forest

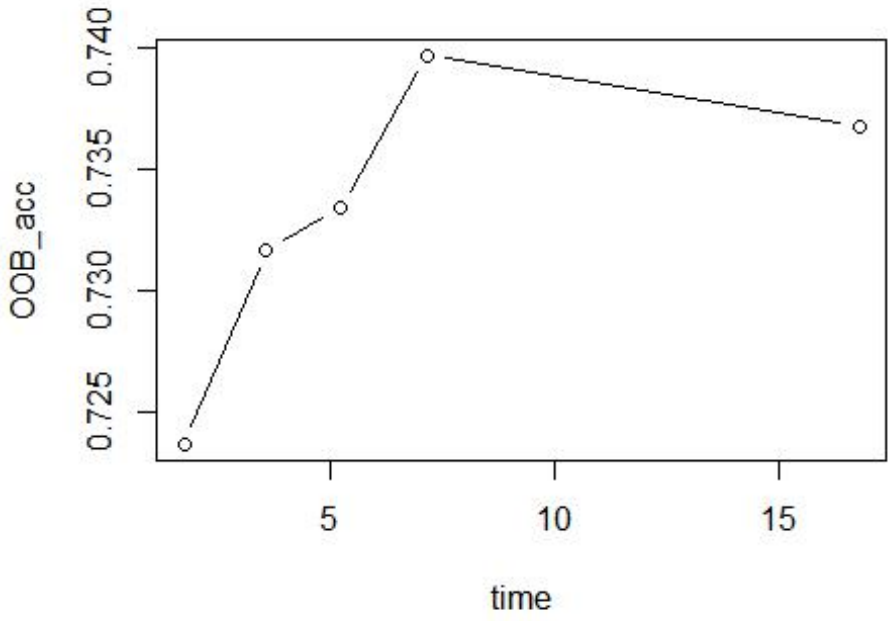
In the beginning, 100,200,300,400,500 are used as ntree to test which number of trees I should use. The function system.time() is used to get the time of each ntree used.

```
library(randomForest)

## randomForest 4.7-1.1

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

##
## 载入程辑包:'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin

ntree = c(100,200,300,400,500)
OOB_acc = vector(mode='numeric',length=length(ntree))
set.seed(123)
for (i in 1:length(ntree)) {
```

```
  forest = randomForest(y~.,x,ntree=ntree[i])
  OOB_acc[i] = round((1-forest$err.rate[ntree[i],1]),4)
}



time = c(1.74,3.58,5.22,7.14,16.79)
table1 = data.frame(row.names = ntree,time,OOB_acc)
table1
```

```
##      time OOB_acc
## 100  1.74  0.7237
## 200  3.58  0.7317
## 300  5.22  0.7334
## 400  7.14  0.7397
## 500 16.79  0.7368
```

```
plot(time,OOB_acc,'b')
```




```
# ntree=200 is the best choice,with acc is high and time is low.
library(randomForest)
set.seed(123)
forest200 = randomForest(y~.,x,ntree=200, importance=T)
OOB_acc200 = round(1-forest200$err.rate[200,1],4)
```

As we can see from the above diagrams, when ntree increased from 100 to 200, the accuracy of OOB increased a lot. When ntree increased from 200 to 500, time increased, but the accuracy of OOB did not increase that much. Thus 200 is to use to train the model.

The function sample(1:a,floor(a*0.85)) is used to take 0.85 of each rank(A-D) randomly. Let 0.85 of the whole data become training data. Moreover, 0.15 of the whole data becomes test data. When ntree=200, the OOB-accuracy of this classifier is 0.7281.

```
#Split into test/train
y1=D1[,11]
y2=D2[,11]
y3=D3[,11]
y4=D4[,11]

y1=as.factor(y1)
y2=as.factor(y2)
y3=as.factor(y3)
y4=as.factor(y4)

set.seed(42)
idx_D1 = sample(1:a,floor(a*0.85))
idx_D2 = sample(1:b,floor(b*0.85))
idx_D3 = sample(1:c,floor(c*0.85))
idx_D4 = sample(1:d,floor(d*0.85))

D1.n = RESF[1:nrow(D1),]
D2.n = RESF[(nrow(D1)+1):(nrow(D2)+nrow(D1)),]
D3.n = RESF[(nrow(D1)+nrow(D2)+1):(nrow(D3)+nrow(D2)+nrow(D1)),]
D4.n = RESF[(nrow(D3)+nrow(D2)+nrow(D1)+1):
            (nrow(D4)+nrow(D3)+nrow(D2)+nrow(D1)),]


x_train = rbind(D1.n[idx_D1,],D2.n[idx_D2,],D3.n[idx_D3,],D4.n[idx_D4,])
x_test = rbind(D1.n[-idx_D1,],D2.n[-idx_D2,],D3.n[-idx_D3,],D4.n[-idx_D4,])
y_train =c(y1[idx_D1],y2[idx_D2],y3[idx_D3],
           y4[idx_D4])
y_test = c(y1[-idx_D1],y2[-idx_D2],y3[-idx_D3],
           y4[-idx_D4])
```

When ntree=200, the OOB-accuracy training data of this classifier= 0.7281

The standard accuracy of training data = 1, which means the model performs very well in training data.

The standard accuracy of test data = 0.8946322

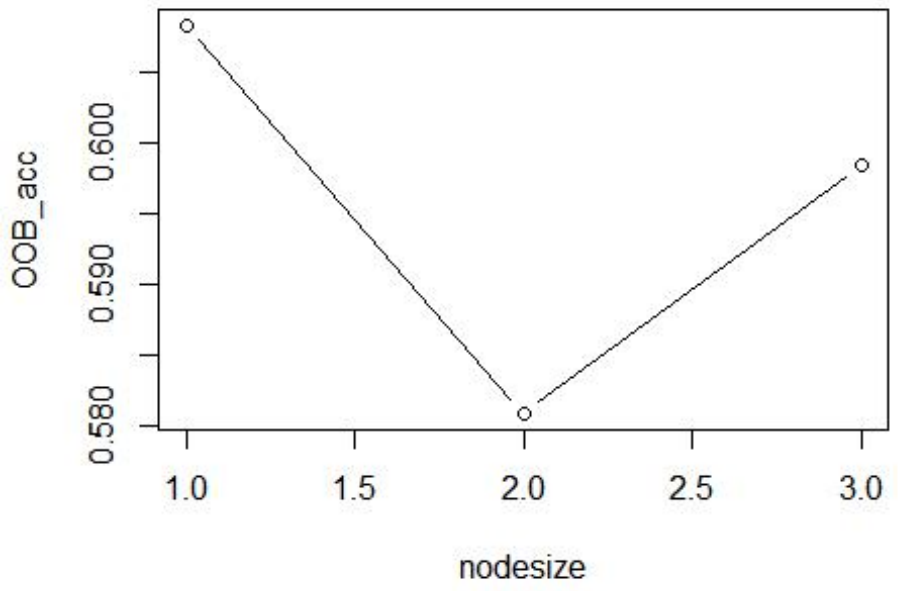
The standard accuracy of test data was obtained by sum(forest_sp_pred1 == y_test)/length(y_test).

```
# When ntree=200 create new Random Forest
library(randomForest)
set.seed(123)
forest_sp = randomForest(y_train~.,x_train,ntree=200,
                         importance=T,
                         min_impurity_decrease = 0.03,
                         min_samples_split = 4,
                         class_weight = balanced_subsample)

# Compute the standard accuracy of the training set
forest_sp_pred1 = predict(forest_sp,x_train)
ACCtrain_sp = sum(forest_sp_pred1 == y_train)/length(y_train)
ACCtrain_sp

## [1] 1

# Compute the standard accuracy of the test set
forest_sp_pred2 = predict(forest_sp,x_test)
ACCtest_sp = sum(forest_sp_pred2 == y_test)/length(y_test)
ACCtest_sp

## [1] 0.7365805

# Compute the error margins on ACCtrain_sp
p1=ACCtrain_sp
em_train = round(sqrt(p1*(1- p1 )/nrow(x_train)),6)
em_train

## [1] 0

# Compute the error margins on ACCtest_sp
p2=ACCtest_sp
em_test = round(sqrt(p2*(1- p2 )/nrow(x_test)),6)
em_test

## [1] 0.00982

# Compute the OOB-accuracy of forest_sp on train set and test set
OOB_acc_sp = round(1-forest_sp$err.rate[200,1],4)
paste('When ntree=200 the OOB-accuracy of this classifier= ',OOB_acc_sp)

## [1] "When ntree=200 the OOB-accuracy of this classifier=  0.7281"
```

1,2,3 are used as nodesize to test which number of nodesize I should use. Nodesize means the number of groups should be divided.

```
nodesize = c(1,2,3)
OOB_acc = vector(mode='numeric',length=length(nodesize))
set.seed(223)
for (i in 1:length(nodesize)) {
  forest = randomForest(y~.,x,ntree=200,nodesize = nodesize[i])
  print(system.time(randomForest(y~.,x,nodesize=nodesize[i])))
```

```
  OOB_acc[i] = round((1-forest$err.rate[nodesize[i],1]),4)
}

## 用户 系统 流逝
## 8.57 0.17 8.91
## 用户 系统 流逝
## 8.39 0.25 8.76
## 用户 系统 流逝
## 8.44 0.28 8.80

table2 = data.frame(OOB_acc,nodesize,row.names = nodesize)

plot(nodesize,OOB_acc,'b')
```




```
#when nodesize = 1 I can get the bet OOB_ACC
forest_sp = randomForest(y_train~.,x_train,ntree=200,
                         importance=T,
                         min_impurity_decrease = 0.03,
                         min_samples_split = 4,
                         class_weight = balanced_subsample)
OOB_acc_sp = round(1-forest_sp$err.rate[200,1],4)

# Compute the standard accuracy of the training set
forest_sp_pred1 = predict(forest_sp,x_train)
ACCtrain_sp = sum(forest_sp_pred1 == y_train)/length(y_train)
ACCtrain_sp
```

```
## [1] 1
```

```
# Compute the standard accuracy of the test set
forest_sp_pred2 = predict(forest_sp,x_test)
ACCtest_sp = sum(forest_sp_pred2 == y_test)/length(y_test)
ACCtest_sp
```

```
## [1] 0.7355865
```

When node = 1, the highest accuracy of OOB appeared. Then nodesize = 1 is taken to train the model.

Loaded Library "caret" to use the confusionMatrix() function. The test confusion Matrix below shows that the 2 of the lowest confusion classes are D2 and D3. Rank B and rank C are chosen as data to train the SVM model to get better grading accuracy.

```
# Compute the confusion matrix of forest_sp on trainning set
library(caret)
```

```
## 载入需要的程辑包:lattice
```

```
expected = factor(y_train)
predicted = factor(forest_sp_pred1)
CM_train = confusionMatrix(predicted,expected)


C1=round(CM_train$table[1,]/sum(CM_train$table[1,]),4)*100
C2=round(CM_train$table[2,]/sum(CM_train$table[2,]),4)*100
C3=round(CM_train$table[3,]/sum(CM_train$table[3,]),4)*100
C4=round(CM_train$table[4,]/sum(CM_train$table[4,]),4)*100

conf_matrix_train = rbind(C1,C2,C3,C4)


# Compute the confusion matrix of forest_sp on test set
library(caret)
expected = factor(y_test)
predicted = factor(forest_sp_pred2)
CM_test = confusionMatrix(predicted,expected)


R1=round(CM_test$table[1,]/sum(CM_test$table[1,]),4)*100
R2=round(CM_test$table[2,]/sum(CM_test$table[2,]),4)*100
R3=round(CM_test$table[3,]/sum(CM_test$table[3,]),4)*100
R4=round(CM_test$table[4,]/sum(CM_test$table[4,]),4)*100

conf_matrix_test = rbind(R1,R2,R3,R4)
conf_matrix_test # The 2 of lowest confusion classes are D2,D3
```

```
##        1     2     3     4
## R1 73.76 18.44  6.38  1.42
## R2 15.52 60.34 18.58  5.56
## R3  0.86 14.56 72.59 11.99
## R4  0.44  3.49  6.75 89.32
```

II.SVM

```
The library 'e1070'  is imported to use SVM calculations.

In the beginning, 1,50,100,150,200,250,300,350,400,450 are used as the
C value. And the kernel is radial, which is Gaussian Kernel(K(x,y) =
exp(-gamma)\*abs(x-y)\^2). The best gamma should be chosen to
get the best SVM model. I used G = c(1/10/dim(x_train)[1],
1/dim(x_train)[1], 10/dim(x_train)[1], 100/dim(x_train)[1]) to
calculate my gamma.
```

```
x_train = rbind(D2.n[idx_D2,],D3.n[idx_D3,])
x_test = rbind(D2.n[-idx_D2,],D3.n[-idx_D3,])
y_train =c(y2[idx_D2],y3[idx_D3])
y_test = c(y2[-idx_D2],y3[-idx_D3])

library(e1071)
c = c(1,50,100,150,200,250,300,350,400,450)
G = c(1/10/dim(x_train)[1], 1/dim(x_train)[1], 10/dim(x_train)[1], 100/dim(x_
train)[1])
support = vector(length = 40)
train_acc = vector(length = 40)
test_acc = vector(length = 40)
for (i in 1:length(c)) {
  for (j in 1:length(G)) {
    svm_result = svm(y_train~.,x_train, scale = TRUE, gamma=G[j],kernel='radi
al',cost = c[i])

    support[i*j] = dim(svm_result$SV)[1]/length(y_train)

    train_pred = predict(svm_result,x_train)
    train_acc[i*j] = sum(train_pred == y_train)/length(y_train)

    test_pred = predict(svm_result,x_test)
    test_acc[i*j] = sum(test_pred == y_test)/length(y_test)
  }
}

result = data.frame('C' = rep(c,each=4),'Gamma'=rep(G,times=10),'testAcc'=tes
t_acc,'trainAcc'=train_acc,'ratio'=test_acc/train_acc,'%support'=support)


svm_result = svm(y_train~.,x_train, verbose = TRUE, kernel='radial',cost=200,
gamma=0.001)
```

Moreover, when the C value increased, the accuracy of the test increased. Thus, a high C value should be set to see if a higher C value could help to improve the SVM model. Then the C value is reset to increase the punishment of the error point.

Now the set of C values is c(1,10,50,100,300,500,1000,2000,3000,5000).

```
c = c(1,10,50,100,300,500,1000,2000,3000,5000)
G = c(1/10/dim(x_train)[1], 1/dim(x_train)[1], 10/dim(x_train)[1], 100/dim(x_
train)[1])
support = vector(length = 40)
train_acc = vector(length = 40)
test_acc = vector(length = 40)
for (i in 1:length(c)) {
  for (j in 1:length(G)) {
    svm_result = svm(y_train~.,x_train, scale = TRUE, gamma=G[j],kernel='radi
al',cost = c[i])

    support[i*j] = dim(svm_result$SV)[1]/length(y_train)

    train_pred = predict(svm_result,x_train)
    train_acc[i*j] = sum(train_pred == y_train)/length(y_train)

    test_pred = predict(svm_result,x_test)
    test_acc[i*j] = sum(test_pred == y_test)/length(y_test)
  }
}

result = data.frame('C' = rep(c,each=4),'Gamma'=rep(G,times=10),'testAcc'=tes
t_acc,'trainAcc'=train_acc,'ratio'=test_acc/train_acc,'%support'=support)
```

The C value cannot be extremely high. In that case, the model will focus too much on the out-group points. The C value should not be small. In that case, the model will have a low discriminate ability.

Radial kernel with cost=1000, gamma=0.018 has good performance. With this model, the accuracy of the test is 0.774, and the accuracy of the training data is 0.801. The accuracy of training data is similar to the test data. The over-fitting problem is mild.

```
svm_result = svm(y_train~.,x_train, verbose = TRUE, kernel='radial',cost=1000,
gamma=0.018)
```

Conclusion

Using Random Forest and SVM to classify people's healthy levels is reasonable. The accuracy of Random Forest to classify group A and group D is 73.76% and 89.32%. However, the accuracy of classifying group B and group C could be better. Especially the accuracy of group B is 60.34%. After using SVM, the accuracy of distinguishing between group B and group C is greatly improved. The accuracy of the test is 0.774. Therefore, we

should use Random Forest to classify group A and group D. Then, use SVM to differentiate group B and group C.