Danie_Ramos_Cancer

Daniel Ramos

6/2/2021

Clasificación del cancer

En el siguiente documento se utilizan los datos extraidos de https://www.kaggle.com/uciml/breast-cancer-wisconsin-data para hacer competir modelos de Boosting y Random Forest.

Carga de datos y limpieza

Se cargan los datos y se eliminan 2 columnas que no nos interesan para clasificar datos y aparte un de ellos contenia NA's

```
set.seed(2021)
#Cargamos los datos
cancer <- read.csv("archive/data.csv")
View(cancer)

summary(cancer)
#Podemos observar que los datos estan "balanceados" ya que diagnosis_number
#tiene un a media de 0.62 y sus valores van entre 0 y 1
attach(cancer)
#No nos interesa la variable "id" ni "X"
cancer <- subset(cancer, select = -c(id))
cancer <- subset(cancer, select = -c(X))
```

Prueba con regresión logística

Aquí realizamos una regresión logística para poder observar de manera rápida la precisión con la que se clasifican los datos y el margen de mejora que tenemos.

```
#Diagnosis esta dividido entre B (Benigno) y M (Malgino)
#B = 1 y M = 0
cancer$diagnosis_number = ifelse(diagnosis=="B",1,0)


#Simplemente hacemos la regresión logística con variables escogidas al azar
#para que que margen de mejora tenemos
glm.fit = glm(diagnosis_number~radius_mean + perimeter_mean, data = cancer, family = binomial)
summary(glm.fit)
glm.probs=predict(glm.fit,type="response")
glm.probs[1:5]
glm.pred=ifelse(glm.probs>0.5,"B","M")
```

```r
table(glm.pred,diagnosis)
mean(glm.pred==diagnosis)
#Se puede observar que la predicción con la regresión logística es muy alta.

#Eliminamos diagnosis number de los datos para que no interfiera en los analisis posteriores
cancer <- subset(cancer, select = -c(diagnosis_number))
```

Se observa que la precisión de los datos es muy buena ya que con logit ya obtenemos una precisión de 90%.

Comparación de Random Forest con Boosting

Hacemos competir boosting con random forest, para ellos utilizamos caret para combinar diversos parametros de cada modelo y Nested CV:

```r
#Creamos una lista del 75% de la filas del dataset original, que utilizaremos
#para entrenar el modelo.
# create a list of 75% of the rows in the original dataset we can use for training
validation_index <- createDataPartition(cancer$diagnosis, p=0.75, list=FALSE)
#Seleccionamos un 25% de la muestra para validar
validation <- cancer[-validation_index,]
#Utilizamos el 75% restante para entrenar y testear los modelos
dataset <- cancer[validation_index,]

#Dividimos los datos en 5 grupos CV = 5
ntrain=length(dataset$diagnosis)
train.ext=createFolds(dataset$diagnosis,k=5,returnTrain=TRUE)
test.ext=lapply(train.ext,function(x) (1:ntrain)[-x])

#Elegimos los parametros de la función train
fitControl <- trainControl(method = 'cv', number = 5, summaryFunction=defaultSummary)

#Escogemos los valores del grid para el modelo gbm
gbmGrid <-  expand.grid(interaction.depth = c(1,4,7,10),
                        n.trees = c(500,1000,2000),
                        shrinkage = c(.005, .02,.05),
                        n.minobsinnode = 10)

gbmGrid

getModelInfo()$rf$parameters
#mtry max:
#Este deberia ser el valor hasta el cual deberia llegar mtry, pero es este caso
#elegiré un valor más bajo para no hacer el cálculo tan largo.
ncol(cancer)-1

#Escogemos los valores del grid para el modelo rf
rfGrid <-  expand.grid(mtry = c(2,3,4,6,9,12))
rfGrid

#Inicializamos la precisión de los modelos a 0
gbmacc <- 0
rfacc <- 0
```

```r
#Guardamos el modelo optimo en su tune correspondiente.
#nrow es 5, porque usamos 5 inner folds, ncol depende de cuantos parametros de "tuneo"
gbmtune = matrix(nrow = 5, ncol = 4)
dimnames(gbmtune) = list(c("k1", "k2", "k3", "k4", "k5"),          # row names
                         c("n.trees", "interaction.depth", "shrinkage", "n.minobsinnode")) # column nam
rftune = matrix(nrow = 5, ncol = 1)
dimnames(rftune) = list(c("k1", "k2", "k3", "k4", "k5"),          # row names
                        c("mtry")) # column names
#Bucle para el CV
for (i in 1:5){

  usedata = cancer[train.ext[[i]],]
  valdata = cancer[test.ext[[i]],]

  fit.gbm <- train(diagnosis~., data=usedata, method = 'gbm', trControl=fitControl, tuneGrid=gbmGrid, m
  fit.gbm

  boost.caret.pred <- predict(fit.gbm,valdata)

  gbmacc[i]=mean(boost.caret.pred==valdata$diagnosis)
  gbmtune[i,1:4] = as.matrix(fit.gbm$bestTune)

  fit.rf <- train(diagnosis~., data=usedata, method = 'rf', trControl=fitControl, tuneGrid=rfGrid, metri
  fit.rf

  rf.caret.pred <- predict(fit.rf,valdata)

  rfacc[i]=mean(rf.caret.pred==valdata$diagnosis)
  rftune[i,1] = as.matrix(fit.rf$bestTune)

}

#Obtenemos la precisión de clasificación de los 5 CV
gbmacc #0.9883721 0.9534884 0.9651163 0.9764706 0.9761905
rfacc #0.9651163 0.9302326 0.9069767 0.9764706 0.9761905

#La media de las precisiones
mean(gbmacc) #0.9719276
mean(rfacc) #0.9509973

#La mejores opciones para cada "tuneo"
gbmtune
rftune
```

El modelo ganador es boosting ya que tiene un precisión mayor, en concreto, 97%. Mientras que random forest 95%.

Modelo ganador

Finalmente el modelo ganador es boosting, por tanto, volvemos a hacer el "tuning" de boosting para evitar una combinación de las diferentes k del CV anterior.

```r
#Elegimos boosting porque es más preciso
fit.gbm <- train(diagnosis~., data=dataset, method = 'gbm', trControl=fitControl, tuneGrid=gbmGrid, meti
fit.gbm
fit.gbm$bestTune

boost.caret.pred <- predict(fit.gbm,validation)
table(boost.caret.pred,validation$diagnosis)
#    B  M
#B 85  1
#M  4 52
mean(boost.caret.pred==validation$diagnosis)
#0.9647887
#Observamos que la predicción ha bajado un poco
```

Finalmente observamos que el modelo ganador de boosting es un modelo con 2000 árboles, interaction.depth de 4, shrinkage de 0.05 y n.minobsinnode de 10 predice con una precisión de 96%