ITESO

Bagging y Random Forest

Sara Eugenia Rodríguez Reyes

IF682324

Introducción

Bagging

Bagging es un método donde se promedia un conjunto de observaciones, logrando reducir la varianza.

Una forma de lograrlo tomar muchos conjuntos de entrenamiento, construir un modelo de predicción por separado usando cada conjunto y promediar las predicciones resultantes; obteniendo un modelo único con baja varianza.

Lo anterior no es práctico, ya que por lo general no se tiene acceso a muchos conjuntos de entrenamiento. Lo mejor, es tomar muestras repetidas de un conjunto de datos de entrenamiento y a partir de ahí generar diferentes conjuntos de entrenamiento. Después entrenamos el método para obtener predicciones y de ahí promediarlas y tener como resultado un modelo de baja varianza.

El problema de este método es que no es tan atractivo de interpretar como los árboles de decisión; le falta claridad para representar cuáles variables son las más importantes en el procedimiento. Para esto se puede obtener un resumen de la importancia de cada predictor usando la suma de residuales al cuadrado RSS para árboles de regresión o el índice de Gini para árboles de clasificación.

Random Forests

Éstos ofrecen una mejora en el método Bagging. Aquí se construye un número de árboles de decisión en conjuntos de entrenamiento, pero al construir estos árboles, cada vez que se considera una partición en el árbol, se escoge una muestra aleatoria de m predictores como candidatos del total conjunto de p predictores.

En otras palabras, al construir un random forest, en cada partición en el árbol no se considera a la mayoría de los predictores disponibles. La razón de esto es que si hay un predictor fuerte en el conjunto de datos; la mayoría de los árboles van a usar este predictor en la primera partición, van a parecer muy similares y tendrán alta correlación. Esto no lleva a una reducción de la varianza.

Los random forest solucionan este problema al forzar que cada partición considere solo una parte de los predictores, se trata de “descorrelacionar” los árboles; por lo tanto los árboles resultantes tienen menos variabilidad.

La función Random Forest, se compone de varias variables:

- data: contiene las variables en el modelo.

- subset: indica las filas que se deben utilizar.
- na.action: especifica la acción a tomar si se encuentran NA
- x, formula: datos, matriz de predictores o fórmula que describe el modelo a ajustarse.
- y: vector de respuesta
- xtest: datos que contienen los predictores para el conjunto de prueba
- ytest: respuesta para el conjunto de prueba
- ntree: número de árboles a cultivar. No tiene que ser un número muy pequeño, para asegurar que cada fila de entrada sea predicha mínimo varias veces
- mtry: número de predictores muestreados por división en cada nodo. Las variables utilizadas si es de clasificación es de sqrt(p), mientras que si es de regresión es p/3.
- replace: indicar si el muestreo debe de ser con remplazamiento.
- classwt: prioriza las clases
- strata: variable usada para el muestreo estratificado
- sampsize: tamaño de la muestra a dibujar.
- nodesize: tamaño mínimo de los nodos terminales. Si es un número grande, hace que los árboles a cultivar sean más pequeños.
- maxnodes: máximo número de nodos terminales de árboles que un random forest puede tener. Si no se pone un número, los árboles crecen a su máximo posible.
- importance: asigna la importancia de los predictores
- localImp: si se calcula los pasos por importancia
- nPerm: número de veces que los datos son permutados por árbol para asignar la importancia de las variables.
- proximity: si la media de proximidad entre las filas debe calcularse
- oob.prox: si la media de proximidad debe ser calculada solo en datos “out-of-bag”
- keep.inbag: se devuelve una matriz que da seguimiento de cuáles muestras están “in-bag” y en cuáles árboles.

Código

Para este caso se utilizará la base de datos "Boston", la cual contiene el valor de las viviendas en los suburbios de Boston.

```
library (randomForest)
library (MASS)
set.seed (1)
#conjunto de entrenamiento
train <- sample(1:nrow(Boston), nrow(Boston)/2)
#mtry=13 indica que los 13 predictores deben ser considerados para
cada división del árbol.
bag.boston <- randomForest(medv~., data=Boston, subset=train,
mtry=13, importance =TRUE)
bag.boston
#predicciones sin el conjunto de entrenamiento
yhatBag <- predict(bag.boston, newdata=Boston[-train, ])
#conjunto de prueba
bostonTest <- Boston[-train, "medv"]
plot(yhatBag, bostonTest)
abline(0,1)
mean(( yhatBag - bostonTest)^2)
13.33831
```

La prueba MSE con el árbol bagging es de 13.338, casi la mitad de la que se obtuvo con el árbol podado (25.04).

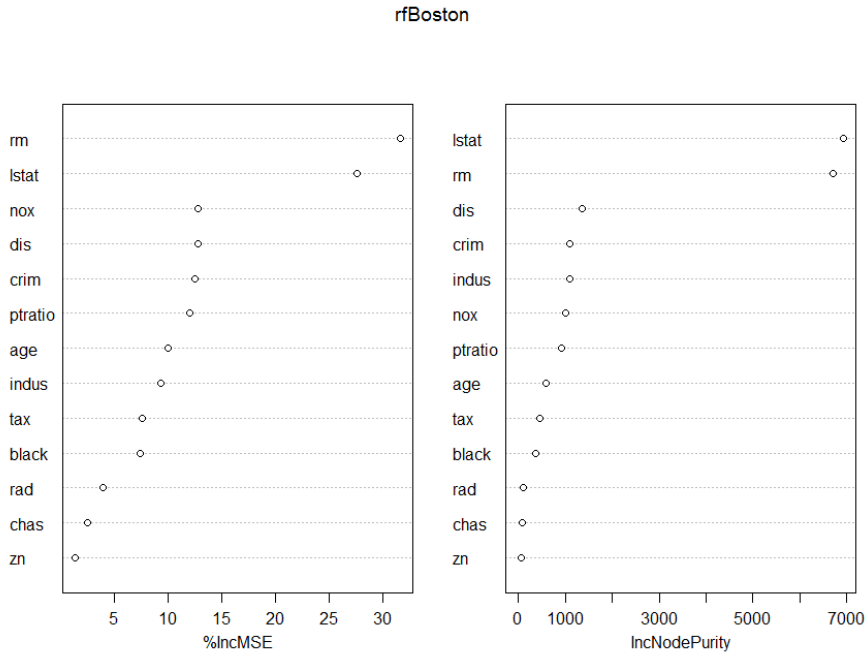
```
bag.boston   <-   randomForest(medv~.,   data=Boston   ,subset=train,
mtry=13, ntree=25) #se agrega el número de árboles a usar
yhatBag <- predict (bag.boston, newdata=Boston[-train, ])
mean((yhatBag - bostonTest)^2)
13.05748
set.seed (1)
rfBoston <- randomForest(medv~., data=Boston, subset=train, mtry=6,
importance=TRUE) #para hacer un random forest se hace de la misma
manera, solo se pone un valor más pequeño de predictores (mtry=6)
yhatRf <- predict(rfBoston, newdata=Boston[-train, ])
mean(( yhatRf - bostonTest)^2)
11.48022
```

Aquí la prueba es de 11.48, menor a la de bagging, lo que indica que los random forest muestran una mejora.

```
importance (rfBoston)
```

```
         %IncMSE IncNodePurity
crim    12.547772    1094.65382
zn       1.375489      64.40060
indus    9.304258    1086.09103
chas     2.518766      76.36804
nox     12.835614    1008.73703
rm      31.646147    6705.02638
age      9.970243     575.13702
dis     12.774430    1351.01978
rad      3.911852      93.78200
tax      7.624043     453.19472
ptratio 12.008194     919.06760
black    7.376024     358.96935
lstat   27.666896    6927.98475
```

```
varImpPlot (rfBoston)
```



Las variables más importantes son las de:

- Lstat: porcentaje del estatus más bajo de la población
- Rm: número promedio de habitaciones por vivienda

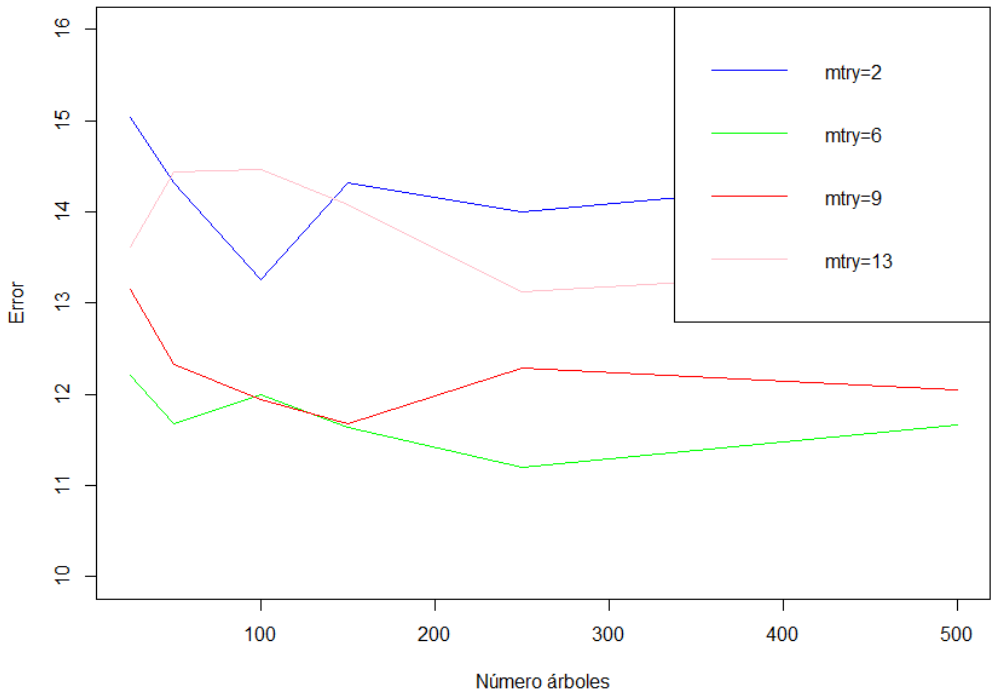
Por lo que el valor de las casas en los suburbios de Boston, dependen en su mayoría del estatus de la población y del número de habitaciones de la casa.

Ejercicio

Crear un gráfico donde se vea el error de las pruebas resultante de los random forest de ese conjunto de datos.

```
#Parámetros a probar
mtry <- c(2,6,9,13)
ntree <- c(25,50, 100,150, 250, 500)
error <-
matrix(rep(NA,length(mtry)*length(ntree)),length(ntree),length(mtry))
set.seed(1)
#muestra de entrenamiento y prueba
train <- sample(1:nrow(Boston), nrow(Boston)/2)
bostonTest <- Boston[-train,'medv']

for(i in 1:length(ntree)){
  for(j in 1:length(mtry)){
    rfBoston <-randomForest(medv~.,data=Boston, subset=train,
mtry=mtry[j], ntree=ntree[i], importance=TRUE)
    yhatRf <- predict(rfBoston, newdata=Boston[-train,])
    mse <- mean((yhatRf-bostonTest)^2)
    error[i,j] <- mse
  }
}
#Gráfico
cols <- c("blue", "green", "red", "pink")
plot(ntree, error[,1], xlab="Número árboles", ylim=c(10,16),
ylab="Error",col=cols[1],type='l')
for(j in 2:length(mtry)){
  lines(ntree,error[,j],col=cols[j])
}
legend("topright", c("mtry=2", "mtry=6", "mtry=9", "mtry=13"),
col=cols, cex = 1, lty = 1)
```



Se observa que el modelo óptimo en donde se da el menor error es utilizando250 árboles y 6 predictores.

Caso aplicado

A continuación se implementará el método de Random Forest a la base de datos Smarket, la cual contiene el rendimiento diario porcentual del índice accionario S&P 500 desde el año 2001 al 2005.

Contiene nueve columnas:

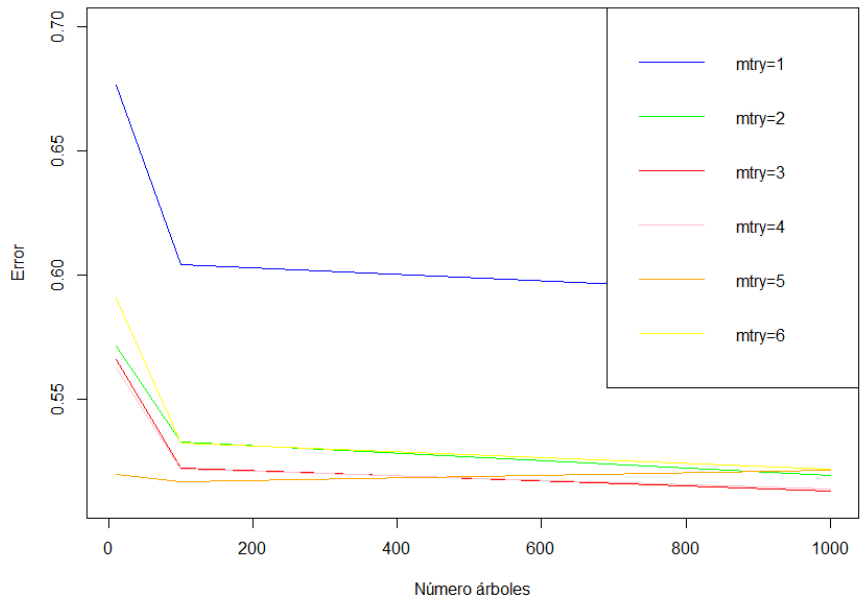
- Year: año en que el dato fue recuperado
- Lag1: Porcentaje de rendimiento para el día anterior
- Lag2: Porcentaje de rendimiento para los dos días anteriores
- Lag3: Porcentaje de rendimiento para los tres días anteriores
- Lag4: Porcentaje de rendimiento para los cuatro días anteriores
- Lag5: Porcentaje de rendimiento para los cinco días anteriores
- Volume: volumen de acciones negociadas
- Today: Rendimiento del día de hoy
- Direction: factor que indica si el mercado tuvo un rendimiento positivo o negativo en el día.

Lo que se buscará es predecir el rendimiento porcentual del día de hoy. De la misma manera se calculará los parámetros óptimos con los que se obtiene el error mínimo.

```
library (ISLR)
library(tree)
library (randomForest)
library (MASS)
set.seed(1)
train <- sample(1:nrow(Smarket), nrow(Smarket)/2) #entrenamiento
smarketTest=Smarket[-train,'Today']    #prueba
mtry <- c(1:6)
ntree <- c(10,100,1000)
error <- matrix(rep(NA,length(mtry)*length(ntree)),length(ntree),length(mtry))
for(i in 1:length(ntree)){
  for(j in 1:length(mtry)){
    rfSmarket <- randomForest(Today~. , data=Smarket, subset=train,
mtry=mtry[j],ntree=ntree[i], importance=TRUE)
    yhatRf <- predict(rfSmarket, newdata=Smarket[-train,])
    mse <- mean((yhatRf-smarketTest)^2)
    error[i,j] <- mse
  }
}
dimnames(error) <- list(c("10", "100", "1000"), c("1","2","3","4","5","6"))
cols <- c("blue", "green", "red", "pink", "orange", "yellow")
plot(ntree, error[,1], xlab="Número árboles", ylim=c(0.48,0.7),
ylab="Error",col=cols[1],type='l')
for(j in 2:length(mtry)){
  lines(ntree,error[,j],col=cols[j])
}
legend("topright", c("mtry=1", "mtry=2", "mtry=3", "mtry=4", "mtry=5", "mtry=6"),
col=cols, cex = 1, lty = 1)
min(error)
```

```
             1         2         3         4         5         6
10   0.6764663 0.5716383 0.5659967 0.5626237 0.5200791 0.5909218
100  0.6043693 0.5330807 0.5224702 0.5219627 0.5169969 0.5325449
1000 0.5925305 0.5195105 0.5129837 0.5140109 0.5214228 0.5218986
```

El mínimo error es de 0.5129, obtenido con los parámetros óptimos de 1000 árboles y 3 predictores.



Conclusión

Aunque los árboles de decisión son interpretables, eficientes y capaces de trabajar con bases de datos grandes, son muy inestables cuando pequeñas cantidades de ruido se introducen en los datos de aprendizaje.

Para evitar esta inestabilidad, se proponen dos modelos.

En bagging, la diversidad se obtiene construyendo cada clasificador con un conjunto de ejemplos diferente que se obtienen seleccionando observaciones con reemplazamiento del conjunto de ejemplos original.

Boosting crea un conjunto de clasificadores añadiendo un clasificador en cada paso. En el conjunto de ejemplos original, cada ejemplo tiene asignado un peso. El clasificador que se añade en cada paso, es aprendiendo del conjunto de ejemplos original donde el peso de cada ejemplo se va siendo modificado. En cada iteración, boosting aprende un modelo que minimiza la suma de los pesos de los ejemplos clasificados erróneamente. Los errores de cada iteración se utilizan para actualizar los pasos de los ejemplos del conjunto de entrenamiento de manera que se incremente el peso de los ejemplos erróneos y se reduzca el peso de los ejemplos clasificados correctamente.