anexo_arboles

2023-02-03

Arboles de clasificacion

Seleccion de variables

Como complemento al trabajo realizado anteriormente con los algoritmos de ML y su posterior evaluación, se ha añadido un apartado de practica modelando arboles de clasificación utilizando R y el dataset *dengue*

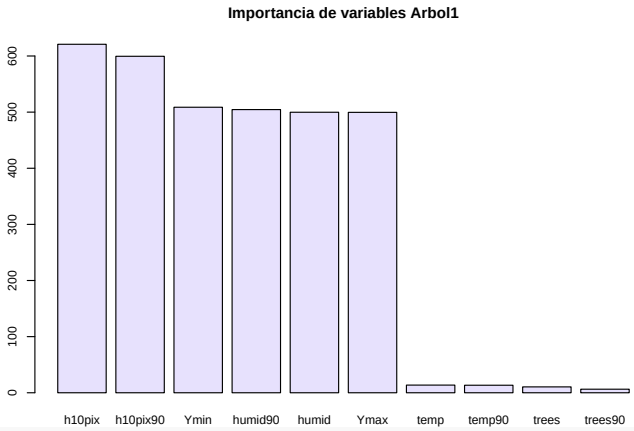
Primer modelado, modelo con todas las variables del dataset para realizar estudio de importancia de variables (que se hizo en el apartado de selección con Stepwise pero se prefiere agregarlo a la practica)

Preguntar si en arboles trabajo con las variables que da el modelo o me ajusto a lo que dio la selección de variables con stepwise

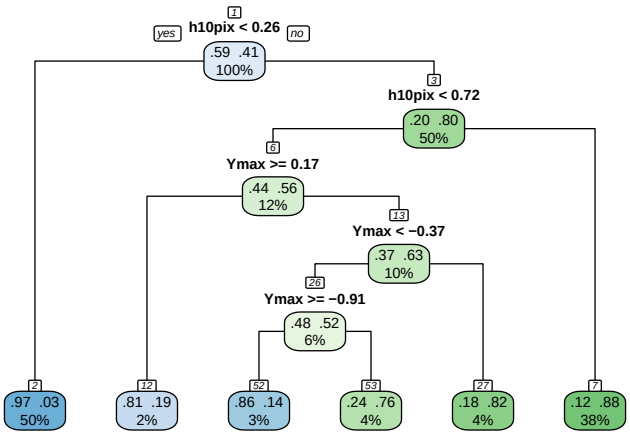
```
# Modelo completo con gini para variable dependiente categorica
arbol1 <- rpart(factor(varObjBin) ~ .,
    data = dengue,
    minbucket = 30,
    method = "class",
    parms = list(split = "gini"))

# Reglas de decision
rattle::asRules(arbol1)
```

Representación gráfica de las variables consideradas relevantes y árbol:

```
# Importancia de variables
par(cex = 0.7)
barplot(height = arbol1$variable.importance, col = "#e7e1fd",
    main = "Importancia de variables Arbol1")
```




```
# Ploteo del arbol
rpart.plot(arbol1, extra = 105, tweak = 1.2, type = 1, nn = TRUE)
```



Observaciones:

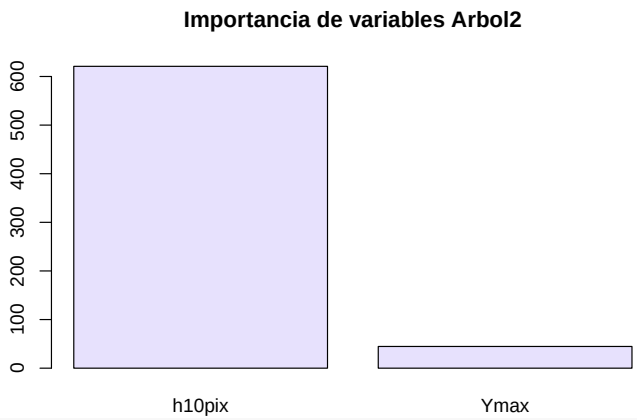
- Según el gráfico importancia de variables Árbol1, las varibles predictoras mas relevantes serian, h10pix, h10pix90, humid90, Ymin, humid e Ymax.
- agregar observaciones del árbol

Se prueba modelando un nuevo árbol esta vez usando maxsurrogate = 0, solo como complemento puesto que el archivo dengue no contiene datos faltantes:

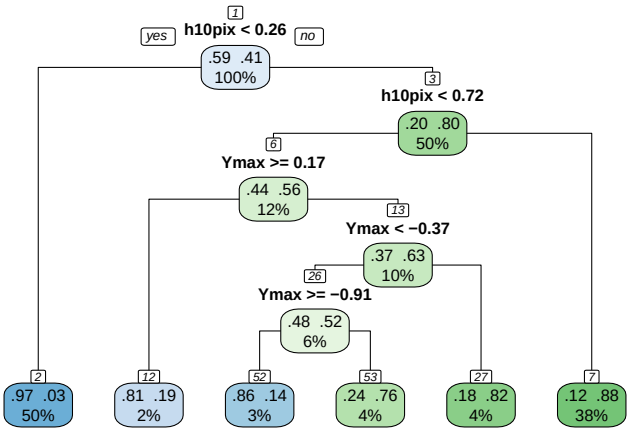
```
# Modelo con maxsurrogate = 0
arbol2 <- rpart(factor(varObjBin) ~ .,
    data = dengue,
    minbucket = 30,
    method = "class",
    maxsurrogate = 0,
    parms = list(split = "gini"))
```

Representación gráfica de las variables consideradas relevantes y árbol:

```
# Importancia de variables
barplot(height=arbol2$variable.importance, col = "#e7e1fd",
        main = "Importancia de variables Arbol2")
```




```
# Ploteo del arbol
rpart.plot(arbol2, extra = 105, tweak = 1.2, type = 1, nn = TRUE)
```



Observaciones:

- Según el gráfico importancia de variables Arbol2, las varibles predictoras mas relevantes serian, h10pix e Ymax.
- agregar observaciones del árbol

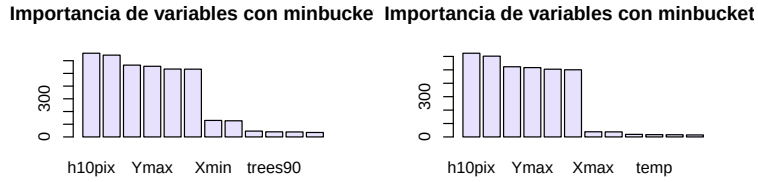
Tunning de algoritmos usando rpart()

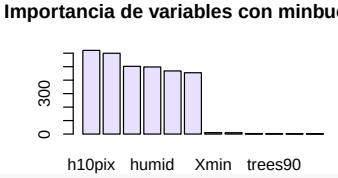
De forma inicial se realiza un primer tunning de arboles observando el comporta miento de la complejidad del árbol graficando su estructura en relación al parámetro minbucket (5, 30 y 60) utilizando la función rpart()

```
# Tuneado sobre complejidad del arbol con Rpart

arbol_min5 <- rpart(factor(varObjBin) ~ ., data = dengue, minbucket = 5, cp = 0)
arbol_min30 <- rpart(factor(varObjBin) ~ ., data = dengue, minbucket = 30, cp = 0)
arbol_min60 <- rpart(factor(varObjBin) ~ ., data = dengue, minbucket = 60, cp = 0)
```

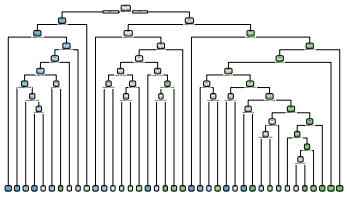
Representación gráfica del tunning de minbucket

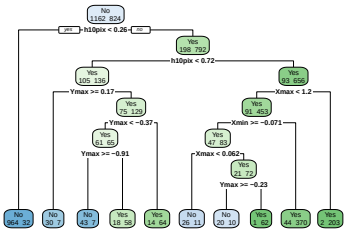
```
par(mfrow= c(2,2))
barplot(height = arbol_min5$variable.importance, col = "#e7e1fd",
        main = "Importancia de variables con minbucket 5")
barplot(height = arbol_min30$variable.importance, col = "#e7e1fd",
        main = "Importancia de variables con minbucket 30")
barplot(height = arbol_min60$variable.importance, col = "#e7e1fd",
        main = "Importancia de variables con minbucket 60")
par(mfrow= c(2,2))
```

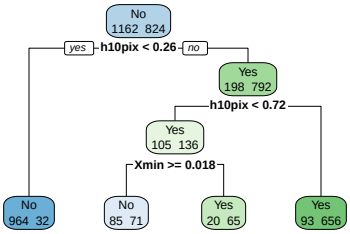





```
rpart.plot(arbol_min5, extra = 1)
rpart.plot(arbol_min30, extra = 1)
rpart.plot(arbol_min60, extra = 1)
```







Observaciones

Se puede apreciar que con minbucket 5 se genera un modelo complejo y con minbucket 60 un modelo básico, para encontrar un equilibrio y ver otras opciones de minbucket a continuación se profundiza en el tuneado utilizando caret de R

Tunning de algoritmos usando caret()

Como indican los apuntes, no se puede realizar tunning en el grid por lo que se construye un bucle for() que recorre por cada numero de minbucket indicado y va guardando los resultados para una posterior evaluación:

```
# Tunning con minbucket en bucle
set.seed(123456)
control_arbol <- trainControl(method = "cv",
    number = 4,
    classProbs = TRUE,
    savePredictions = "all")

arbolgrid <-  expand.grid(cp = c(0))
tabla_resultados <-c()

for (minbu in seq(from = 5, to = 60, by = 5)){
    arbolgrid <-  expand.grid(cp=c(0))
    arbolcaret <- train(factor(varObjBin) ~ h10pix + temp90 + Ymin + Ymax +
        temp + humid + humid90 + trees + trees90,
        data = dengue, method = "rpart", minbucket = minbu,
        trControl = control_arbol, tuneGrid = arbolgrid)

    accuracy <- arbolcaret$results$Accuracy
    sal <- arbolcaret$pred
    salconfu <- confusionMatrix(sal$pred, sal$obs)
    curvaroc <-roc(response = sal$obs, predictor = sal$Yes)
    auc <- curvaroc$auc
    # Guarda los resultados en formato tabla
    tabla_resultados <- rbind(tabla_resultados, c(minbu, accuracy, auc))
}

# Cambia los nombres de las columnas
colnames(tabla_resultados) <- c("minibucket","accuracy","AUC")
```

Con el código anterior se genera una tabla para una mejor visualización de los indicadores:

```
knitr::kable(tabla_resultados, "pipe")
```

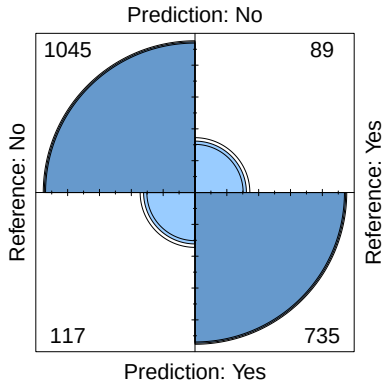
minibucket	accuracy	AUC
5	0.9018151	0.9607650
10	0.9058464	0.9486380
15	0.8997939	0.9534793
20	0.8962647	0.9551551
25	0.9018090	0.9501461
30	0.9164148	0.9591859
35	0.9048434	0.9482521
40	0.9128907	0.9600058
45	0.8887184	0.9429633
50	0.9053454	0.9583901
55	0.9113755	0.9558511
60	0.9043211	0.9507905

Como se puede apreciar en la tabla anterior, en combinación entre mejor accuracy y AUC para este modelo es adecuado utilizar un minbucket de 30. Con ese numero se realiza la validación cruzada repetida utilizando la función cruzadaarbolbin()

```
arbol4 <- train(factor(varObjBin) ~ h10pix + temp90 + Ymin + Ymax + temp +
    humid + humid90 + trees + trees90,
    data = dengue,
    method = "rpart",
    minbucket = 30,
    trControl = control_arbol,
    tuneGrid = arbolgrid)
```

Se obtiene la matriz de confusión para este modelo

```
salida_a4 <- arbol4$pred
salida4_confusion <- confusionMatrix(salida_a4$pred, salida_a4$obs)
fourfoldplot(salida4_confusion$table)
```



Se obtiene la curva ROC

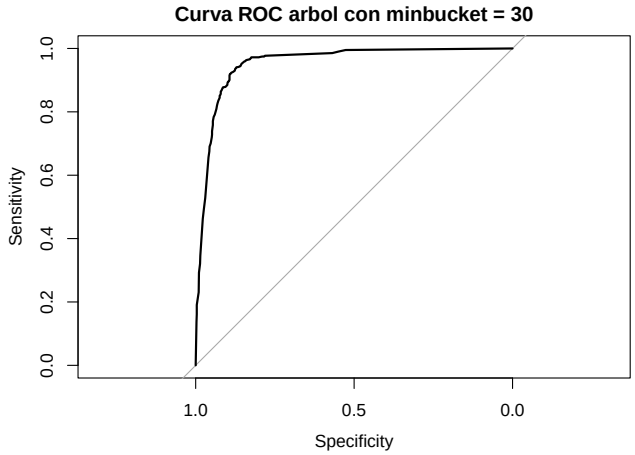
```
# Curva roc
curvaroc <-roc(response = salida_a4$obs, predictor = salida_a4$Yes)
```

```
## Setting levels: control = No, case = Yes
```

```
## Setting direction: controls < cases
```

```
auc <- curvaroc$auc
plot.roc(roc(response = salida_a4$obs, predictor = salida_a4$Yes), main = "Curva ROC arbol con minbucket = 30")
```

```
## Setting levels: control = No, case = Yes
## Setting direction: controls < cases
```



Comparacion con otros algoritmos de ML

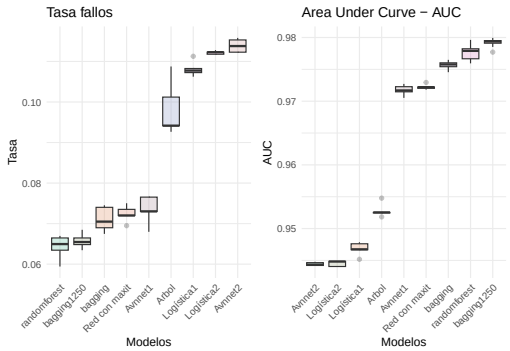
Con los resultados obtenidos de entrenar el modelo se realiza la validación cruzada repetida para posteriormente comparar las medias con los demás modelos y evaluar sesgo-varianza

```
# Con los resultados de la tabla con iteraciones se configura la validacion cruzada
medias_arbol <-cruzadaarbolbin(
    data = dengue,
```

```
    vardep = "varObjBin",
    listconti = c("h10pix", "temp90", "trees", "trees90", "Ymin", "Ymax", "temp",
    "humid", "humid90"),
    listclass = c(""), grupos = 4, sinicio = 1234, repe = 5,
    cp = c(0), minbucket = 30)

medias_arbol$modelo <- "Arbol"
```

Con la función genera_gráficos() se despliega la comparación de medias para los distintos algoritmos entrenados

```
genera_graficos(medias1, medias2, medias3, medias4, medias5,medias6, medias9,medias11, medias_arbol)
```



Observaciones:

- Para esta practica en particular, un árbol no es lo suficientemente competitivo frente a otros algoritmos modelados. Presenta mejores indicadores que la regresión logística o una red neuronal pero ante otros algoritmos como bagging o random forest se queda un tanto atrás.