Modelos y resultados

William Alexander Aguirre A

2024-04-13

1. Modelos de clasificación

En esta sección se muestran los resultados de los modelos más relevantes encontrados para clasificar como pobres o no pobres a los hogares. La variable objetivo a predecir es *Pobre* y como predictores se utilizaron las variables que se encuentran en Predictores.

Capturamos la data que se encuentra en el repositorio con las diferentes variables que hemos traido a partir de la información de personas y las transformaciones de interés. Para ver mayor detalle del procesamiento de la data se puede consultar el script Creación de variables de interés.R.

Predicciones con Logit.

Inicialmeante se entrena un modelo con todos los predictores que no poseen valores perdidos en los set de entrenamiento y prueba.Inicialmente se hizo un entrenamiento con la data sin Bogotá para utilizar la variable Dominio, sin embargo, al no ser significativa se retiró la variable y trabajamos con el set de entrenamiento completo. Tambien se retiraron las variables que tienen multicolinealidad, estos predictores se pueden consultar en Predictores Logit.

```
logit<-glm(as.formula(
  paste("Pobre~",
        paste(predictores_logit, collapse = " + "))),data = train_hogares)
```

Los detalles de los coeficientes, niveles de significancia y métricas de ajuste se pueden observar en summary logit. Se evalua el modelo dentro de muestra:

```
train_hogares$prob_logit<-predict(logit,newdata = train_hogares)
train_hogares$logit_predict<-ifelse(train_hogares$prob_logit>0.5,1,0)

matrix_logit<-table(train_hogares$logit_predict,
                    train_hogares$Pobre)

cf_logit<-confusionMatrix(matrix_logit,
                positive = "1",
                mode="prec_recall")
cf_logit$byClass["F1"]
```

```
##        F1
## 0.2889066
```

Los detalles de la salida de la matriz de confusion se encuentran en matriz de confusión modelo Logit.

Se realiza la predicción fuera de muestra y se genera el submission para Kaggle utilizando la sintaxis que se encuentra en Submission Logit

```
sub3<-test_hogares %>% select(id,logit_predict)
sub3<-sub3 %>% rename(pobre=logit_predict)
write_csv(x = sub3,"C:/Users/HP-Laptop/Documents/GitHub/Curso-Big-Data/Taller 2/2.Entregables/Submissio
```

También se genera un modelo con las interacciones entre las variables predictores para interacciones. Evaluamos el modelo dentro de muestra.

```
interacciones<-c("informalidad_jefe","edad_jefe_joven","sexo_jefe","desempleo_jefe","tipo_casa")



logit2<-glm(as.formula(
  paste("Pobre~",
        paste(interacciones, collapse = " * "))),data = train_hogares)
```

```
train_hogares$prob_logit2<-predict(logit2,newdata = train_hogares)
train_hogares$logit_predict2<-ifelse(train_hogares$prob_logit2>0.5,1,0)
summary(train_hogares$prob_logit2)
```

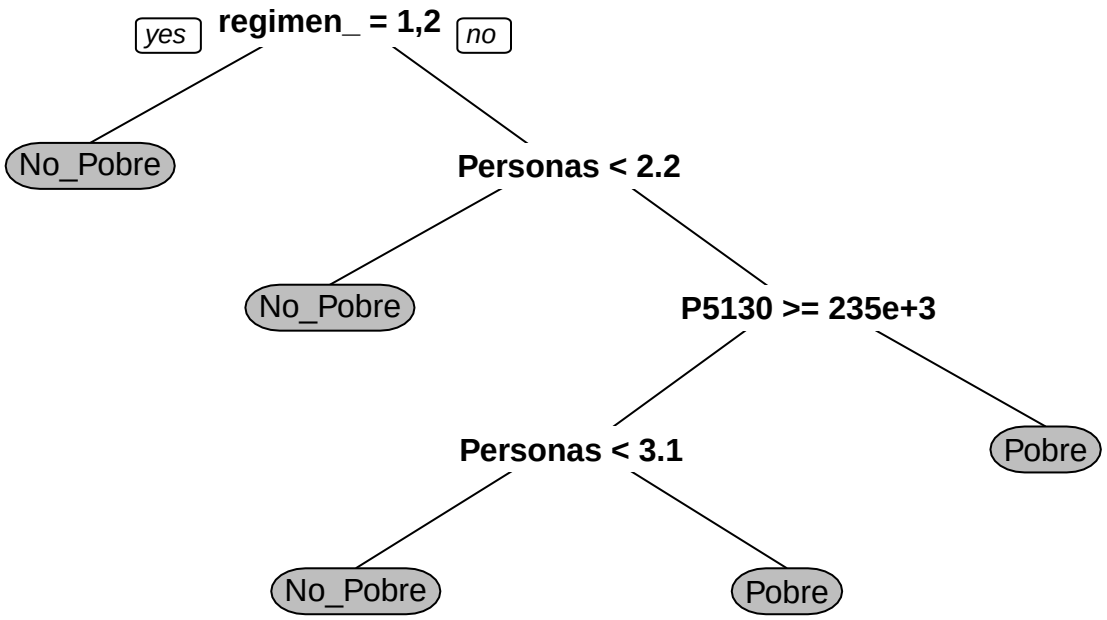
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.02374 0.13061 0.17777 0.20019 0.27360 0.43697
```

La probabilidad maxima que predice la interacción entre estas variables es inferior a 0.5 por lo que no se tomó en cuenta para hacer predicciones fuera de muestra.

Predicciones con Árbol

Para entrenar el árbol se utilizó la siguiente sintaxis.

```
arbol<-rpart(formula = as.formula(paste("pobre_texto~",
                                        paste(predictores,
                                              collapse = " + "))),
             data=train_hogares,
             method = "class",
             parms = list(split="Gini"))
prp(arbol,box.col = "gray")
```



Se evaluan los resultados dentro de muestra, el detalle de la matriz de confusión se encuentra en matriz de confusión árbol de clasificación.

```
train_hogares$predic_arbol<-predict(arbol,newdata =train_hogares,
                                    type = "class")

cf_arbol<-confusionMatrix(data = train_hogares$predic_arbol,
                reference = train_hogares$pobre_texto,
                positive="Pobre",
                mode = "prec_recall")
cf_arbol$byClass["F1"]
```

```
##        F1
## 0.3313086
```

Se realiza la predicción fuera de muestra y se genera el submission para Kaggle. La sintaxis utilizada para generar el submission se encuentra en Submission Árbol.

```
test_hogares$predic_arbol<-predict(arbol,newdata =test_hogares,type = "class")
table(test_hogares$predic_arbol)
```

```
##
## No_Pobre    Pobre
##    61469     4699
```

```
sub9<-test_hogares %>% select(id,predic_arbol)
sub9<-sub9 %>% rename(pobre=predic_arbol)
sub9$pobre<-ifelse(sub9$pobre=="Pobre",1,0)
table(test_hogares$predic_arbol)
table(sub9$pobre)
write_csv(x = sub9,"C:/Users/HP-Laptop/Documents/GitHub/Curso-Big-Data/Taller 2/2.Entregables/Submissio
```

Predicciones con Random Forest

El algoritmo de Random Forest no nos permite utilizar variables con valores perdidos por lo que se excluyen de los predictores algunas variables que no aplican para toda la muestra. Los predictores utilizados para entrenar el modelo se pueden consultar en Predictores Random Forest. Se realiza un entrenamiento inicial de un modelo de Ramdom Forest utilizando la siguiente sintaxis.

```
RF<- ranger(formula = as.formula(paste("pobre_texto~",
                                       paste(predictores, collapse = " + "))),
            data = train_hogares,
            num.trees= 500, ## Numero de arboles a estimar
            mtry= 4,    # N. var aleatoriamente seleccionadas en cada partición.
            min.node.size  = 1, ## Numero minimo de observaciones en un nodo
            importance="impurity")
RF
```

```
## Ranger result
##
## Call:
##  ranger(formula = as.formula(paste("pobre_texto~", paste(predictores,      collapse = " + "))), data
##
## Type:                             Classification
## Number of trees:                  500
## Sample size:                      164960
## Number of independent variables:  21
## Mtry:                             4
## Target node size:                 1
## Variable importance mode:         impurity
## Splitrule:                        gini
## OOB prediction error:             16.91 %
```

Para definir la cantidad optima de variables seleccionada por cada partición se prueba aumentar la cantidad de arboles para identificar que hay una reducción importante del OOB predictor error.

```
RF1000<- ranger(formula = as.formula(paste("pobre_texto~",
                                       paste(predictores, collapse = " + "))),
            data = train_hogares,
            num.trees= 1000, ## Aumentamos de 500 a 1000
            mtry= 4,
            min.node.size  = 1,
            importance="impurity")
RF1000
```

```
## Ranger result
```

```
##
## Call:
##  ranger(formula = as.formula(paste("pobre_texto~", paste(predictores,      collapse = " + "))), data
##
## Type:                             Classification
## Number of trees:                  1000
## Sample size:                      164960
## Number of independent variables:  21
## Mtry:                             4
## Target node size:                 1
## Variable importance mode:         impurity
## Splitrule:                        gini
## OOB prediction error:             16.95 %
```

Se observa que no hay reducción importante del OBB predictor error. Para continuar de afinar hiperparametros del modelo aumentamos la cantidad de minima de observaciones por nodo.

```
RF_NODE100<- ranger(formula = as.formula(paste("pobre_texto~",
                                    paste(predictores, collapse = " + "))),
            data = train_hogares,
            num.trees= 500,
            mtry= 4,
            min.node.size  = 100, #Aumentamos de 1 a 100
            importance="impurity")
RF_NODE100
```

```
## Ranger result
##
## Call:
##  ranger(formula = as.formula(paste("pobre_texto~", paste(predictores,      collapse = " + "))), data
##
## Type:                             Classification
## Number of trees:                  500
## Sample size:                      164960
## Number of independent variables:  21
## Mtry:                             4
## Target node size:                 100
## Variable importance mode:         impurity
## Splitrule:                        gini
## OOB prediction error:             16.92 %
```

Vemos que tampoco existe un cambio importante en el OBB predictor error. Finalmente probamos aumentar la cantidad de variables por árbol.

```
RF_5VAR<- ranger(formula = as.formula(paste("pobre_texto~",
                                    paste(predictores, collapse = " + "))),
            data = train_hogares,
            num.trees= 500,
            mtry= 5,
            min.node.size  = 1,
            importance="impurity")
```

```
## Growing trees.. Progress: 16%. Estimated remaining time: 2 minutes, 40 seconds.
```

```
## Growing trees.. Progress: 31%. Estimated remaining time: 2 minutes, 18 seconds.
## Growing trees.. Progress: 48%. Estimated remaining time: 1 minute, 42 seconds.
## Growing trees.. Progress: 64%. Estimated remaining time: 1 minute, 10 seconds.
## Growing trees.. Progress: 74%. Estimated remaining time: 55 seconds.
## Growing trees.. Progress: 85%. Estimated remaining time: 33 seconds.
## Growing trees.. Progress: 95%. Estimated remaining time: 10 seconds.
```

```
RF_6VAR<- ranger(formula = as.formula(paste("pobre_texto~",
                                    paste(predictores, collapse = " + "))),
            data = train_hogares,
            num.trees= 500,
            mtry= 6,
            min.node.size  = 1,
            importance="impurity")
```

```
## Growing trees.. Progress: 13%. Estimated remaining time: 3 minutes, 35 seconds.
## Growing trees.. Progress: 27%. Estimated remaining time: 2 minutes, 45 seconds.
## Growing trees.. Progress: 42%. Estimated remaining time: 2 minutes, 8 seconds.
## Growing trees.. Progress: 57%. Estimated remaining time: 1 minute, 32 seconds.
## Growing trees.. Progress: 73%. Estimated remaining time: 57 seconds.
## Growing trees.. Progress: 88%. Estimated remaining time: 25 seconds.
```

```
RF_7VAR<- ranger(formula = as.formula(paste("pobre_texto~",
                                    paste(predictores, collapse = " + "))),
            data = train_hogares,
            num.trees= 500,
            mtry= 7,
            min.node.size  = 1,
            importance="impurity")
```

```
## Growing trees.. Progress: 11%. Estimated remaining time: 4 minutes, 5 seconds.
## Growing trees.. Progress: 23%. Estimated remaining time: 3 minutes, 32 seconds.
## Growing trees.. Progress: 35%. Estimated remaining time: 2 minutes, 51 seconds.
## Growing trees.. Progress: 48%. Estimated remaining time: 2 minutes, 17 seconds.
## Growing trees.. Progress: 60%. Estimated remaining time: 1 minute, 42 seconds.
## Growing trees.. Progress: 73%. Estimated remaining time: 1 minute, 10 seconds.
## Growing trees.. Progress: 85%. Estimated remaining time: 38 seconds.
## Growing trees.. Progress: 97%. Estimated remaining time: 7 seconds.
```

```
RF_5VAR$prediction.error
```

```
## [1] 0.1704292
```

```
RF_6VAR$prediction.error
```

```
## [1] 0.1723448
```

```
RF_7VAR$prediction.error
```

```
## [1] 0.174703
```

```
## [1] "OBB predictor error"
```

```
## [1] 0.1703322
```

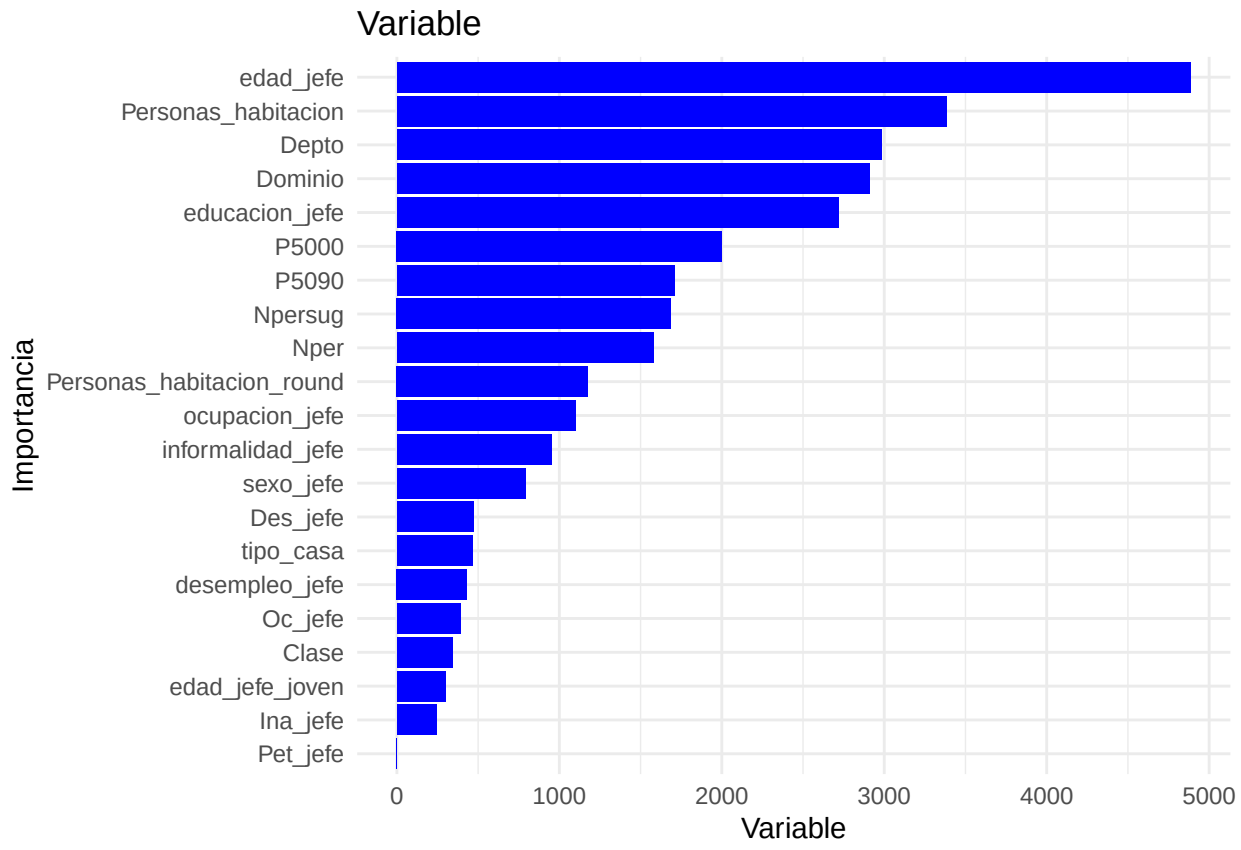
```
## [1] 0.1721266
```

```
## [1] 0.1749333
```

Observamos que después de cinco variables por árbol el OBB predictor error deja de reducirse. Por lo que se toma la decisión de trabajar con el primer árbol.

Podemos observar también la importancia que tienen las variables en el arbol para identificar que algunas cobran mayor relevancia en comparación de un arbol sencillo.

```
imp<-importance(RF)
imp<-data.frame(variables=names(imp),importancia=imp)
ggplot(imp, aes(x = reorder(variables, importancia) , y =importancia )) +
  geom_bar(stat = "identity", fill = "blue") +
  labs(title = "Variable ", x = "Importancia", y="Variable") +
  theme_minimal() +
  coord_flip()
```



Analizamos las predicciones realizadas dentro de muestra. El detalle de la matriz de confusión se encuentra en matriz de confusión de modelo Ramdom Forest.

```
cf_rf<-confusionMatrix(data = RF$predictions,
                reference = train_hogares$pobre_texto,
                positive = "Pobre",
                mode = "prec_recall")
cf_rf$byClass["F1"]
```

```
##        F1
## 0.4414414
```

Predecimos fuera de muestra y generamos el submission para Kaggle. Podemos consultar la sintaxis con la que se dió formato a las predicción en formato de submission con Random Forest.

2. Modelos de regresión del ingreso

En esta sección se muestran los resultados de los modelos más relevantes encontrados para predecir el ingreso con el objetivo de apartir de estas predicciones poder clasificar como pobres o no pobres a los hogares, tomando el umbral de pobreza como límite de clasificación entre los dos grupos de hogares.

Predicciones con regresión lineal

Para realizar la predicción del ingreso de los hogares utilizamos como variable dependiente el ingreso total de la unidad de gasto con impotación de arriendo a propietarios y usufructuario (*Ingtotugarr*). Las variables predictoras se pueden consultar en predictores para regresión líneal.

Para correr el modelo se utilizó la siguiente sintaxis:

```
modelo_lm<-lm(as.formula( #Entrenamos el modelo
  paste("Ingtotug~",paste(predictores, collapse = " + "))),
  data =train_hogares_sin_bogota)
```

Los detalles de la salida de regresión se pueden consultar en summary modelo de regresión lineal. Realizamos las predicciones en el conjunto de entrenamiento para poder clasificar, la tabla derivada de la matriz de confusión se puede consultar completa en matriz de confusión de modelo de regresión lineal.

```
train_hogares_sin_bogota$modelo_lm_ingreso<-predict(object = modelo_lm, newdata = train_hogares_sin_bog
```

```
## Warning in predict.lm(object = modelo_lm, newdata = train_hogares_sin_bogota):
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases
```

```
#Clasificamos los hogares en pobres o no según el ingreso predicho
train_hogares_sin_bogota$modelo_lm_predict<-ifelse(
  train_hogares_sin_bogota$modelo_lm_ingreso<train_hogares_sin_bogota$Lp,
                  1,0)
matrix_lm<-table(train_hogares_sin_bogota$modelo_lm_predict,train_hogares_sin_bogota$Pobre)
cf_lm<-confusionMatrix(matrix_lm,
                positive="1",
                mode = "prec_recall")
cf_lm$byClass["F1"]
```

```
##        F1
## 0.1371305
```

Luego realizamos la predicción en el conjunto de prueba y exportamos con el formato adecuado para realizar submitir en Keaggle. La sintaxis utilizada para esto se encuentra en formato para submission con regresión lineal.

```
sub1<-test_hogares %>% select(id,modelo_lm_predict)
sub1<-sub1 %>% rename(pobre=modelo_lm_predict)
write_csv(x = sub1,"C:/Users/HP-Laptop/Documents/GitHub/Curso-Big-Data/Taller 2/2.Entregables/Submissio
```

Posteriormente se realizan diferentes combinaciones de variables predictivas para encontrar aquellas que tengan una mejor capacidad de predicción.

Predicciones con Regularización.

Para realizar estos modelos utilizamos solo las variables que no poseen NA en train ni en test, estos se pueden consultar en predictores regularización. Para aplicar regularización tenemos que trabajar con nuestros datos en formato matriz, la sintaxis utilizada también se puede observar en transformación en matriz para regularización.

Ridge, Lasso y Elastic Net

Para entrenar los modelos fue necesario afinar los parámetros lambda utilizados para la predicción. En este caso, el parámetro lambda mínimo fue encontrado mediante cross validation utilizando la siguiente sintaxis:

```
cv_ridge <- cv.glmnet(x = X,
                      y = Y,alpha =i)
# Para Ridge i=1, Lasso i=0 Lasso, Elastic Net i=0.75.

coef(cv_ridge, s = "lambda.min")
test_hogares$predict_ingreso_rige<-predict(cv_ridge,
                                           newx = X_test,
                                           s = "lambda.min")
```

Elastic Net con Cartet

Para identificar los hiperparametros optimos de Elastic Net con Caret se utilizó la siguiente sintaxis.

```
tc_10 <- trainControl(method = "cv", number = 10)

en_caret <- train(x=X,y=Y,method = "glmnet",trControl = tc_10,
  tuneLength=100)
```

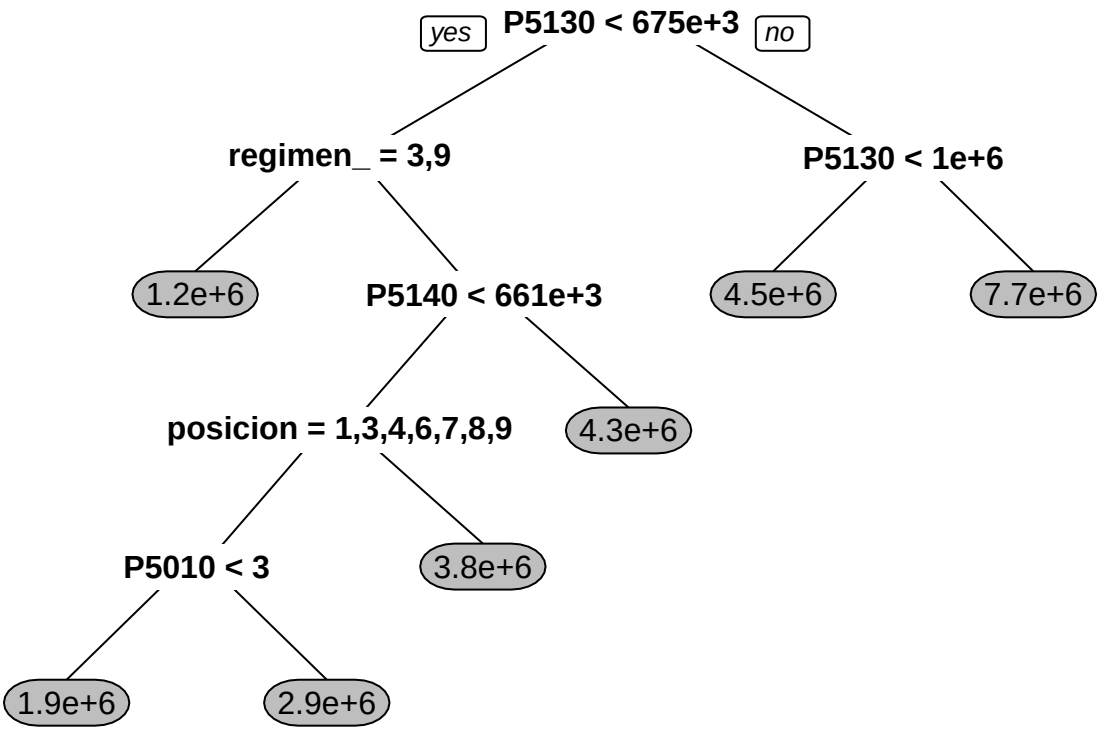
```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,
## : There were missing values in resampled performance measures.
```

Predicciones con Árbol

Para predecir el ingreso contamos con mas flexibilidad al momento de utilizar variables con NA, estas variables se pueden consultar en predictores para árbol de regresión . Para realizar la predicción con árboles de decisión se realizó la siguiente sintaxis

```
arbol<-rpart(formula = as.formula(paste("Ingtotugarr~",
                                        paste(predictores,
                                              collapse = " + "))),
                        data=train_hogares_sin_bogota,
                        parms = list(split="Gini"))

prp(arbol,box.palette = "gray")
```



Luego de entrenar el modelo realizamos la predicción en train y evaluamos su resultado dentro de muestra. La totalidad de las métricas derivadas de la matriz de confusión se encuentran en matriz de confusión árbol de regresión.

```
train_hogares_sin_bogota$predic_arbol_ingreso<-predict(arbol,
                                            newdata =train_hogares_sin_bogota)

train_hogares_sin_bogota$predict_arbol<-ifelse(
  train_hogares_sin_bogota$predic_arbol_ingreso<
    train_hogares_sin_bogota$Lp*train_hogares_sin_bogota$Npersug,
  1,0)
```

```
matriz_arbol<-table(train_hogares_sin_bogota$predict_arbol,train_hogares_sin_bogota$Pobre)
cf_arbol_reg<-confusionMatrix(matriz_arbol,
                positive="1",
                mode = "prec_recall")

cf_arbol_reg$byClass["F1"]
```

```
##        F1
## 0.3388104
```

Luego realizamos la predicción fuera de muestra:

```
test_hogares$predic_arbol_ingreso<-predict(arbol,
                                           newdata =test_hogares)

test_hogares$predict_arbol3<-ifelse(
  test_hogares$predic_arbol_ingreso<
    test_hogares$Lp*test_hogares$Npersug,
  1,0)
table(test_hogares$predict_arbol)
```

```
##
##     0     1
## 59622  6546
```