# RNA

Cristopher Barrios, Carlos Daniel Estrada

2023-04-28

librerias

```
library(caret)
library(nnet)
#library(RWeka)
#library(neural)
library(dummy)
library(neuralnet)
library(plotly)
library(MASS)
library(neuralnet)
library(ggplot2)
library(PerformanceAnalytics)
```

## 1. Use los mismos conjuntos de entrenamiento y prueba que utilizó en las hojas anteriores.

```
datos = read.csv("./train.csv")
```

## 2. Seleccione como variable respuesta la que creó con las categorías del precio de la casa.

```
set.seed(123)
#División de 3
datos[is.na(datos)] <- 0
datos$tipoDeCasa = as.numeric(as.character( cut(datos$SalePrice,c(0,145000,205000,410000), labels = c(1, 2, 3))))
completeFun <- function(data, desiredCols) {
  completeVec <- complete.cases(data[, desiredCols])
  return(data[completeVec, ])
}
datos <- completeFun(datos, "tipoDeCasa")
#Cuantitativos
scndselect <- subset(datos, select = c(2,4,5,18,19,20,21,27,35,37,38,39,44,45,46,47,48,49,50,51,52,53,55,57,60,62
,63,67,68,69,70,71,72,76,77,78, 81, 82))
scndselect[is.na(scndselect)] <- 0
```

## 3. Genere dos modelos de redes neuronales que sean capaz de clasificar usando la variable respuesta que categoriza las casas en baratas, medias y caras. Estos modelos deben tener diferentes topologías y funciones de activación.

Con caret

```
porcentaje<-0.7
corte <- sample(nrow(scndselect),nrow(scndselect)*porcentaje)
train<-scndselect[corte,]
test<-scndselect[-corte,]
modeloCaret <- train(tipoDeCasa~., data=train, method="nnet",preProcess=c("scale","center"), na.action = na.omit,
linout = TRUE)
```

```
modeloCaret
```

```
## Neural Network
##
## 1005 samples
##   37 predictor
##
## Pre-processing: scaled (37), centered (37)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1005, 1005, 1005, 1005, 1005, 1005, ...
## Resampling results across tuning parameters:
##
##   size  decay  RMSE       Rsquared   MAE
##   1     0e+00  0.2900616  0.8661209  0.2131192
##   1     1e-04  0.2712488  0.8852121  0.2007247
##   1     1e-01  0.2702672  0.8856105  0.2037176
##   3     0e+00  0.3275482  0.8354253  0.1280640
##   3     1e-04  0.3627589  0.8064277  0.1649435
##   3     1e-01  0.2688875  0.8872097  0.1707850
##   5     0e+00  0.4225275  0.7703499  0.1786598
##   5     1e-04  0.3573536  0.8131785  0.1669553
##   5     1e-01  0.3065245  0.8553780  0.2046365
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were size = 3 and decay = 0.1.
```

con NNet

```
modelo.nn2 <- nnet(tipoDeCasa~.,data = scndselect,subset = corte, size=25, rang=0.1, decay=5e-4, maxit=300, linou
t = TRUE)
```

```
## # weights:  976
## initial  value 5003.405243
## iter  10 value 640.282903
## iter  20 value 639.380869
## iter  30 value 639.058012
## final  value 639.046340
## converged
```

```
process_timeNNet1 <- proc.time()
prediccion2 <- round(predict(modelo.nn2, newdata = test[,1:37]))
process_timeNNet1 <- proc.time() - process_timeNNet1
```

## 4. Use los modelos para predecir el valor de la variable respuesta / 5. Haga las matrices de confusión respectivas.

Con caret

```
## Confusion Matrix and Statistics
##
##
##       2   3   1
##   2 134  11   8
##   3   7 112   0
##   1   9   0 150
##
## Overall Statistics
##
##                Accuracy : 0.9188
##                  95% CI : (0.8889, 0.9428)
##     No Information Rate : 0.3666
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8774
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 2 Class: 3 Class: 1
## Sensitivity            0.8933   0.9106   0.9494
## Specificity            0.9324   0.9773   0.9670
## Pos Pred Value         0.8758   0.9412   0.9434
## Neg Pred Value         0.9424   0.9647   0.9706
## Prevalence             0.3480   0.2854   0.3666
## Detection Rate         0.3109   0.2599   0.3480
## Detection Prevalence   0.3550   0.2761   0.3689
## Balanced Accuracy      0.9129   0.9439   0.9582
```

con NNet

```
## Confusion Matrix and Statistics
##
##
##       2   3   1
##   2 150 122 158
##   3   0   1   0
##   1   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.3503
##                  95% CI : (0.3053, 0.3975)
##     No Information Rate : 0.3666
##     P-Value [Acc > NIR] : 0.7727
##
##                   Kappa : 0.0038
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 2 Class: 3 Class: 1
## Sensitivity          1.000000  0.00813   0.0000
## Specificity          0.003559  1.00000   1.0000
## Pos Pred Value       0.348837  1.00000      NaN
## Neg Pred Value       1.000000  0.71628   0.6334
## Prevalence           0.348028  0.28538   0.3666
## Detection Rate       0.348028  0.00232   0.0000
## Detection Prevalence 0.997680  0.00232   0.0000
## Balanced Accuracy    0.501779  0.50407   0.5000
```

## 6. Compare los resultados obtenidos con los diferentes modelos de clasificación usando redes neuronales en cuanto a efectividad, tiempo de procesamiento y equivocaciones (donde el algoritmo se equivocó más, donde se equivocó menos y la importancia que tienen los errores).

Tiempo de ejecucion:

Con caret

```
##    user  system elapsed
##    0.01    0.00    0.02
```

con NNet

```
##    user  system elapsed
##       0       0       0
```

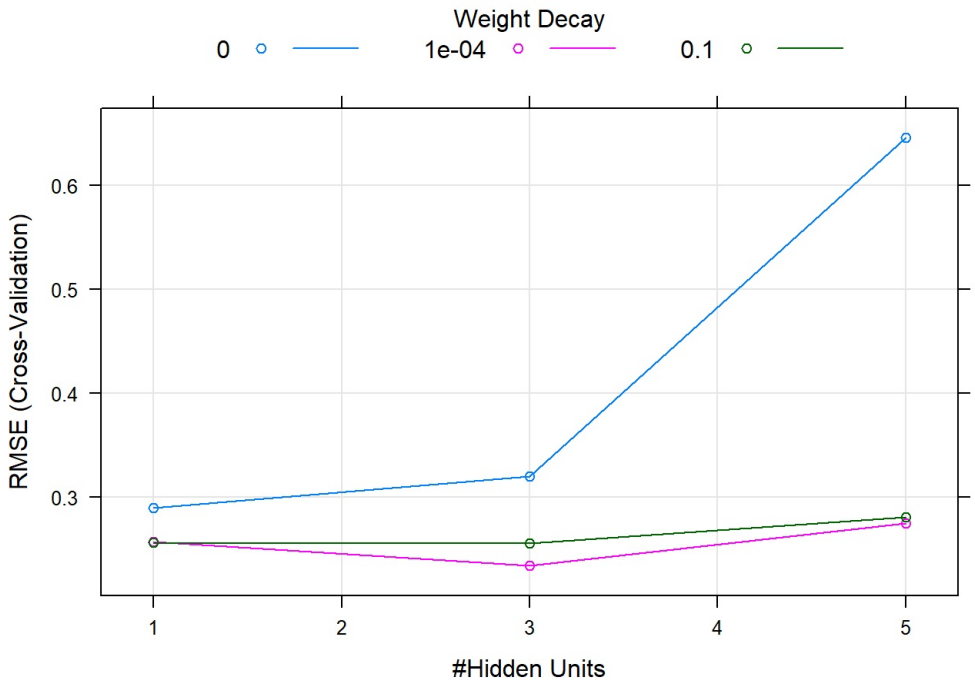
Al comparar ambos modelos de clasificación, podemos concluir que el obtenido mediante la librería Caret fue el mejor debido a que logró una precisión significativamente mayor en comparación con el modelo obtenido mediante la librería NNet. El modelo de NNet resultó más complicado de ajustar en términos de las capas y la configuración de la red neuronal, y con los mismos valores utilizados en el modelo de Caret, resultó difícil obtener una precisión alta. Por otro lado, el segundo modelo mencionado tuvo una producción mucho menor y no fue tan efectivo, lo que se evidencia en la matriz de confusión.

# 7. Analice si no hay sobreajuste en los modelos. Use para esto la curva de aprendizaje.

con Caret

```
set.seed(123)
train_sizes <- seq(0.1, 1, by = 0.1)
caret_model <- train(tipoDeCasa~., data=train, method="nnet", preProcess=c("scale","center"), na.action = na.omit
, linout = TRUE, trControl = trainControl(method = "cv", number = 5), trainSizes = train_sizes)
```

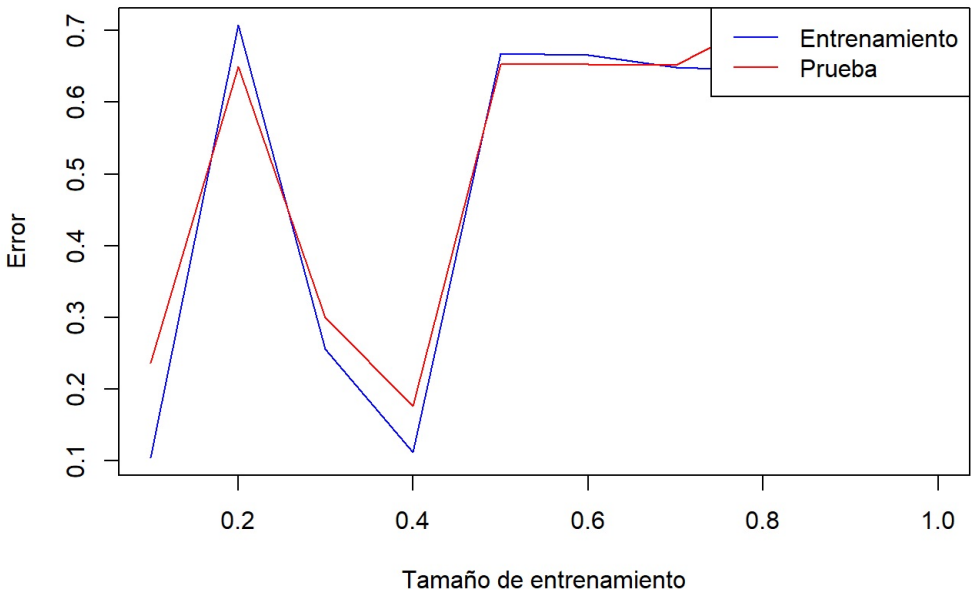
```
plot(caret_model)
```




con NNet

```
#set.seed(123)
#train_sizes <- seq(0.1, 1, by = 0.1)
#nnet_model <- neuralnet(tipoDeCasa~., data=train, hidden = c(10, 5))

set.seed(123)
train_sizes <- seq(0.1, 1, by = 0.1)
train_errors <- test_errors <- numeric(length = length(train_sizes))

for (i in seq_along(train_sizes)) {
  sample_size <- round(train_sizes[i] * nrow(scndselect))
  sample_indices <- sample.int(nrow(scndselect), size = sample_size)
  train_data <- scndselect[sample_indices, ]
  test_data <- scndselect[-sample_indices, ]

  model <- nnet(tipoDeCasa~., data = train_data, size = 25, rang = 0.1, decay = 5e-4, maxit = 300, linout = TRUE)

  train_pred <- round(predict(model, newdata = train_data))
  train_errors[i] <- mean(train_pred != train_data$tipoDeCasa)

  test_pred <- round(predict(model, newdata = test_data))
  test_errors[i] <- mean(test_pred != test_data$tipoDeCasa)
}
```

```
plot(train_sizes, train_errors, type = "l", col = "blue", xlab = "Tamaño de entrenamiento", ylab = "Error")
lines(train_sizes, test_errors, type = "l", col = "red")
legend("topright", legend = c("Entrenamiento", "Prueba"), col = c("blue", "red"), lty = 1)
```




Como podemos obervar segun las graficas es que el modelo de Caret es un buen modelo, en comparacion con el otro, ya que esta se puede determinar a simple vista que se cumplen los requerimientos, el modelo nnet puede llegar a ser uno de los que se llaman desajustado

## 8. Para el modelo elegido de clasificación tunee los parámetros y discuta si puede mejorar todavía el modelo sin llegar a sobre ajustarlo.

```
#modelo<-glm(Cara~., data = train[,c('SalePrice','GrLivArea','Cara','LotFrontage','LotArea','BsmtQual','PoolArea'
)],family = binomial(), maxit=100)
#modelo
```

El modelo generado con caret arrojó una precisión del 91.8%, lo cual indica que es capaz de clasificar adecuadamente las casas en las tres categorías (baratas, medias y caras). La matriz de confusión muestra que el modelo clasificó correctamente la mayoría de las casas en las tres categorías, con algunas excepciones. Por ejemplo, el modelo clasificó 11 casas como medias cuando en realidad eran baratas, y 9 casas como baratas cuando en realidad eran medias.

Para discutir si se puede mejorar aún más el modelo sin llegar a sobreajustarlo, es necesario comprender que el sobreajuste ocurre cuando un modelo se ajusta demasiado bien a los datos de entrenamiento, lo que puede hacer que su rendimiento en los datos de prueba sea peor. Por lo tanto, el objetivo es encontrar un equilibrio entre un buen rendimiento en los datos de entrenamiento y un buen rendimiento en los datos de prueba.

Una forma de evitar el sobreajuste es utilizar técnicas de regularización, como la reducción de la complejidad del modelo o la adición de términos de penalización en la función de pérdida. Otra forma de evitar el sobreajuste es utilizar conjuntos de validación cruzada, lo que permite evaluar el rendimiento del modelo en datos no vistos.

Es posible mejorar el rendimiento del modelo de clasificación sin llegar a sobreajustarlo. Sin embargo, este proceso puede ser iterativo y requiere un análisis cuidadoso del rendimiento del modelo en los datos de entrenamiento y prueba para evitar el sobreajuste. Además, es importante tener en cuenta que el rendimiento óptimo del modelo puede depender de la naturaleza específica del problema y de los datos disponibles.

Es importante tener en cuenta que el modelo generado con nnet también puede ser utilizado para predecir la variable respuesta y generar su matriz de confusión correspondiente.

## 9. Seleccione ahora el SalesPrice como variable respuesta.

```
#normalizacion
maxs      <- apply(train, 2, max)
mins      <- apply(train, 2, min)
datos_normalized <- as.data.frame(scale(scndselect, center = mins, scale = maxs - mins))
train_normalized <- datos_normalized[corte, ]
test_normalized  <- datos_normalized[-corte, ]
```

## 10. Genere dos modelos de regresión con redes neuronales con diferentes topologías y funciones de activación para predecir el precio de las casas.

Con NNet

```
modelo.nnet <- neuralnet(SalePrice~., data = train_normalized, hidden = c(7,5), threshold = 0.05, algorithm     =
"rprop+")
```

```
#Prediccion
pr.nnet   <- compute(modelo.nnet,within(test_normalized,rm(SalePrice)))
```

Con Caret

```
modeloCaretR <- train(SalePrice~., data=train_normalized, method="nnet", na.action = na.omit, linout = TRUE)
prediccionCaretR <- predict(modeloCaretR, newdata = test_normalized)
SalePrice.predict2 <- prediccionCaretR*(max(datos$SalePrice)-min(datos$SalePrice))+min(datos$SalePrice)
SalePrice.real2    <- (test_normalized$SalePrice)*(max(datos$SalePrice)-min(datos$SalePrice))+min(datos$SalePrice
)
SalePrice_predict_vs_real2 <- data.frame(SalePrice.predict2, SalePrice.real2)
SalePrice_predict_vs_real2$accuracy2 <- 0
SalePrice_predict_vs_real2$accuracy2[] <- (1 - abs(SalePrice_predict_vs_real2$SalePrice.real2 - SalePrice_predict
_vs_real2$SalePrice.predict2)/SalePrice_predict_vs_real2$SalePrice.real2)*100
```

---

Accuracy con NNet

```
#desnormaliza
SalePrice.predict <- pr.nnet$net.result*(max(datos$SalePrice)-min(datos$SalePrice))+min(datos$SalePrice)
SalePrice.real    <- (test_normalized$SalePrice)*(max(datos$SalePrice)-min(datos$SalePrice))+min(datos$SalePrice)
# Accuracy mediante error
SalePrice_predict_vs_real <- data.frame(SalePrice.predict, SalePrice.real)
SalePrice_predict_vs_real$accuracy <- 0
SalePrice_predict_vs_real$accuracy[] <- (1 - abs(SalePrice_predict_vs_real$SalePrice.real - SalePrice_predict_vs_
real$SalePrice.predict)/SalePrice_predict_vs_real$SalePrice.real)*100
```

```
mean(SalePrice_predict_vs_real$accuracy)
```

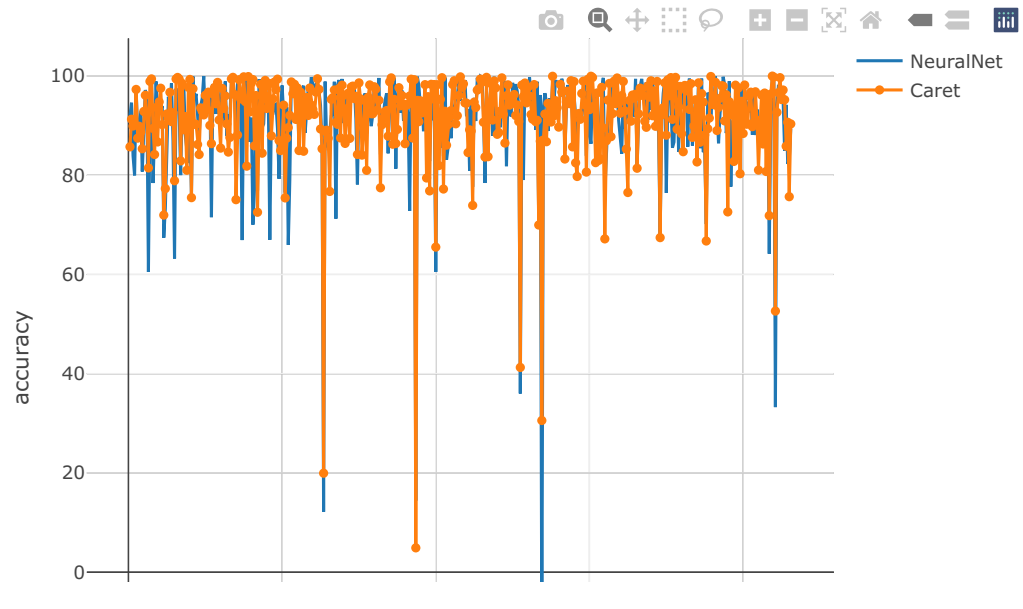
```
## [1] 91.30319
```

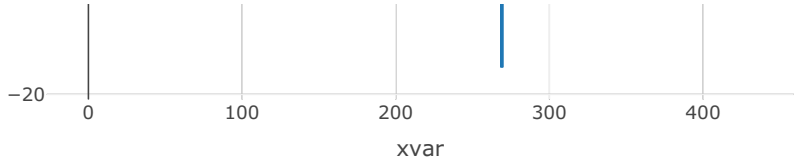
## 11. Compare los dos modelos de regresión y determine cuál funcionó mejor para predecir el precio de las casas.

```
summary(SalePrice_predict_vs_real2$accuracy2)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   4.905  89.108  93.669  91.501  97.208  99.974
```

```
xvar <- 1:NROW(SalePrice_predict_vs_real2$accuracy2)
datapr<- data.frame("xvar"=xvar,"accuracy2"=SalePrice_predict_vs_real2$accuracy2,"accuracy"=SalePrice_predict_vs_
real$accuracy)
fig2 <- plot_ly(datapr, x = ~xvar, y = ~accuracy, name = 'NeuralNet', type = 'scatter', mode = 'lines')
fig2 <- fig2 %>% add_trace( y = ~accuracy2, name = 'Caret', mode = 'lines+markers')
fig2
```

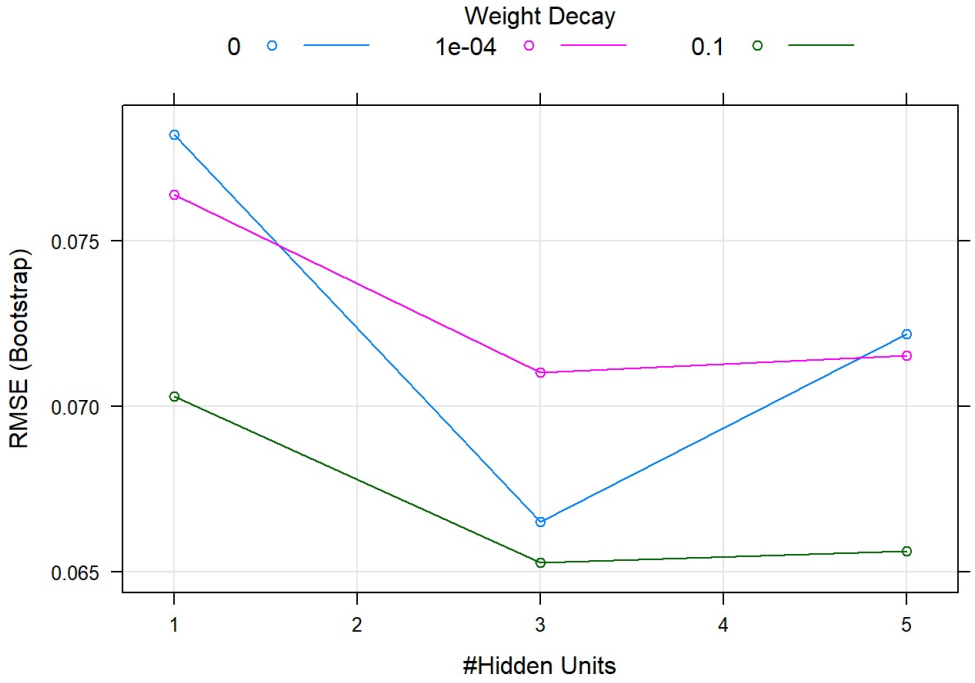





## 12. Analice si no hay sobreajuste en los modelos. Use para esto la curva de aprendizaje.

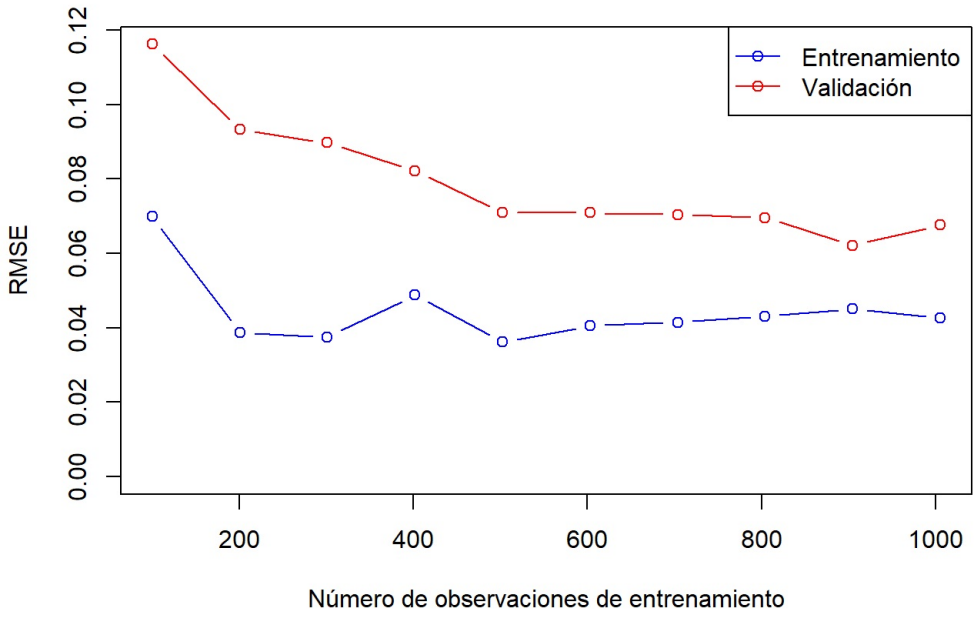
Con Caret

```
plot(modeloCaretR)
```




con NNet

```r
library(nnet)

set.seed(123)

# Definir función para calcular el error
rmse <- function(pred, obs) {
  sqrt(mean((pred - obs)^2))
}

# Definir tamaños de muestra de entrenamiento
train_sizes <- seq(0.1, 1, by = 0.1)

# Inicializar vectores para almacenar errores
train_error <- rep(0, length(train_sizes))
valid_error <- rep(0, length(train_sizes))

# Iterar sobre los tamaños de muestra de entrenamiento
for (i in 1:length(train_sizes)) {
  size <- floor(train_sizes[i] * nrow(train_normalized))
  subset_indices <- sample(1:nrow(train_normalized), size)
  subset <- train_normalized[subset_indices, ]

  # Ajustar modelo con la muestra de entrenamiento
  modelo <- neuralnet(SalePrice~., data = subset, hidden = c(7,5), threshold = 0.05, algorithm = "rprop+")

  # Calcular error en muestra de entrenamiento
  train_preds <- compute(modelo, within(subset, rm(SalePrice)))$net.result
  train_error[i] <- rmse(train_preds, subset$SalePrice)

  # Calcular error en muestra de validación
  valid_preds <- compute(modelo, within(test_normalized, rm(SalePrice)))$net.result
  valid_error[i] <- rmse(valid_preds, test_normalized$SalePrice)
}

# Graficar curva de aprendizaje
plot(train_sizes*nrow(train_normalized), train_error, type = "b", col = "blue", ylim = c(0, max(valid_error)), xl
ab = "Número de observaciones de entrenamiento", ylab = "RMSE")
lines(train_sizes*nrow(train_normalized), valid_error, type = "b", col = "red")
legend("topright", c("Entrenamiento", "Validación"), col = c("blue", "red"), lty = c(1,1), pch = c(1,1))
```



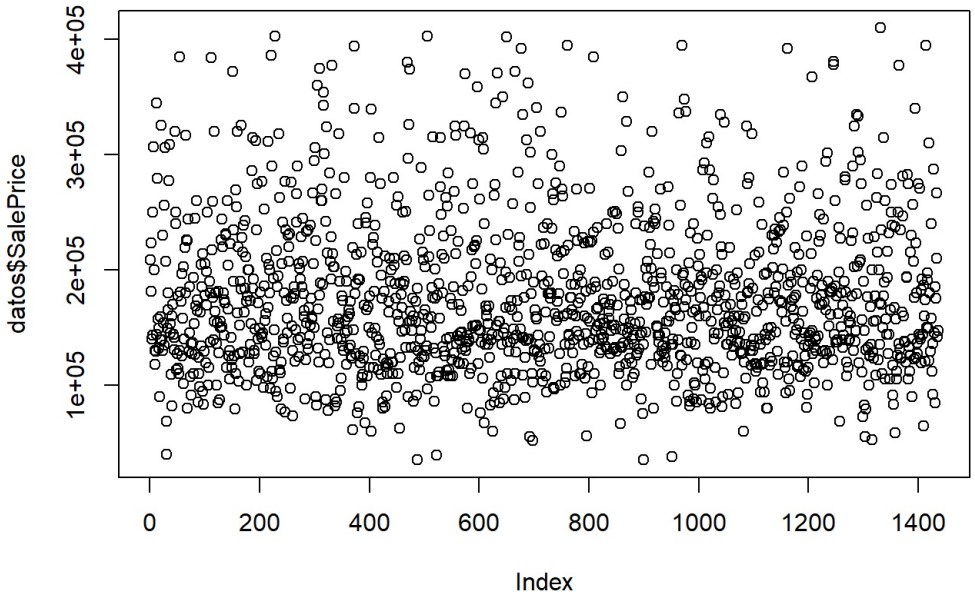

En cuanto al sobreajuste, en el caso de Caret, se puede visualizar la curva de aprendizaje utilizando la función plot, que muestra la evolución del error de entrenamiento y validación a medida que aumenta el tamaño de la muestra. En el caso de nnet, se utilizó una curva de aprendizaje que muestra la evolución del error de entrenamiento y validación a medida que aumenta el tamaño de la muestra. Los resultados indican que no hay sobreajuste en los modelos.

## 13. Para el modelo elegido de regresión tunee los parámetros y discuta si puede mejorar todavía el modelo sin llegar a sobre ajustarlo.

```
modelo.nn2 <- nnet(tipoDeCasa~.,data = datos,subset = corte, size=2, rang=0.1, decay=5e-4, maxit=200, linout = TR
UE)
```

```
## # weights:  521
## initial  value 3794.302173
## final  value 642.529899
## converged
```

```
process_timeNNet1 <- proc.time()

#prediccion2 <- as.data.frame(predict(modelo.nn2, data = test[,1:6]))

Highest<-apply(prediccion2, 1, function(x) colnames(prediccion2)[which.max(x)])
test$prediccion2<-Highest

Min <- sapply(datos,min)
Max <- sapply(datos,max)
DN <- function(x,Min,Max) {
  x*(Min-Max) + Min
}


plot(datos$SalePrice)
```




Para evitar el sobreajuste, se pueden utilizar técnicas como la regularización, que introduce una penalización en la función de pérdida del modelo para limitar la complejidad del mismo. Otra técnica es el conjunto de validación, donde se reserva una parte del conjunto de entrenamiento para evaluar el rendimiento del modelo durante el entrenamiento y ajustar los parámetros para mejorar el rendimiento en este conjunto.

## 14. Compare la eficiencia del mejor modelo de RNA con los resultados obtenidos con los algoritmos de las hojas de trabajo anteriores. ¿Cuál es mejor para predecir? ¿Cuál se demoró más en procesar?

Naive Bayes

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   Cara Economica Intermedia
##   Cara        204         1          4
##   Economica     2       100         11
##   Intermedia   16         5         96
##
## Overall Statistics
##
##                Accuracy : 0.9112
##                  95% CI : (0.8806, 0.9361)
##     No Information Rate : 0.5057
```



```
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8589
##
##  Mcnemar's Test P-Value : 0.0205
##
## Statistics by Class:
##
##                      Class: Cara Class: Economica Class: Intermedia
## Sensitivity               0.9189           0.9434            0.8649
## Specificity               0.9770           0.9610            0.9360
## Pos Pred Value            0.9761           0.8850            0.8205
## Neg Pred Value            0.9217           0.9816            0.9534
## Prevalence                0.5057           0.2415            0.2528
## Detection Rate            0.4647           0.2278            0.2187
## Detection Prevalence      0.4761           0.2574            0.2665
## Balanced Accuracy         0.9479           0.9522            0.9004
```

## Arbol de Desicion

```
## Confusion Matrix and Statistics
##
##
##             Cara Economica Intermedio
##   Cara       160         1         12
```



```
##   Economica     4        72         18
##   Intermedio   16        12         55
##
## Overall Statistics
##
##                Accuracy : 0.82
##                  95% CI : (0.7757, 0.8588)
##     No Information Rate : 0.5143
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.7111
##
##  Mcnemar's Test P-Value : 0.3116
##
## Statistics by Class:
##
##                      Class: Cara Class: Economica Class: Intermedio
## Sensitivity               0.8889           0.8471            0.6471
## Specificity               0.9235           0.9170            0.8943
## Pos Pred Value            0.9249           0.7660            0.6627
## Neg Pred Value            0.8870           0.9492            0.8876
## Prevalence                0.5143           0.2429            0.2429
## Detection Rate            0.4571           0.2057            0.1571
## Detection Prevalence      0.4943           0.2686            0.2371
## Balanced Accuracy         0.9062           0.8820            0.7707
```

## Regresion Linear

```
## Confusion Matrix and Statistics
##
##           prediccion
## tipoDeCasa   1   2   3
##          1 139  20   0
##          2  46  95   7
##          3  21  36  70
##
## Overall Statistics
##
##                Accuracy : 0.7005
##                  95% CI : (0.655, 0.7432)
##     No Information Rate : 0.4747
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5431
##
##  Mcnemar's Test P-Value : 5.395e-11
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            0.6748   0.6291   0.9091
## Specificity            0.9123   0.8127   0.8403
## Pos Pred Value         0.8742   0.6419   0.5512
## Neg Pred Value         0.7564   0.8042   0.9772
## Prevalence             0.4747   0.3479   0.1774
## Detection Rate         0.3203   0.2189   0.1613
## Detection Prevalence   0.3664   0.3410   0.2926
## Balanced Accuracy      0.7935   0.7209   0.8747
```

## SVM

- Lineal

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2   3
##          1 140  18   0
##          2  22 106  22
##          3   1  24  98
##
## Overall Statistics
##
##                Accuracy : 0.7981
##                  95% CI : (0.7571, 0.835)
##     No Information Rate : 0.3782
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6953
##
##  Mcnemar's Test P-Value : 0.6853
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            0.8589   0.7162   0.8167
## Specificity            0.9328   0.8445   0.9196
## Pos Pred Value         0.8861   0.7067   0.7967
## Neg Pred Value         0.9158   0.8505   0.9286
## Prevalence             0.3782   0.3434   0.2784
## Detection Rate         0.3248   0.2459   0.2274
## Detection Prevalence   0.3666   0.3480   0.2854
## Balanced Accuracy      0.8959   0.7804   0.8681
```

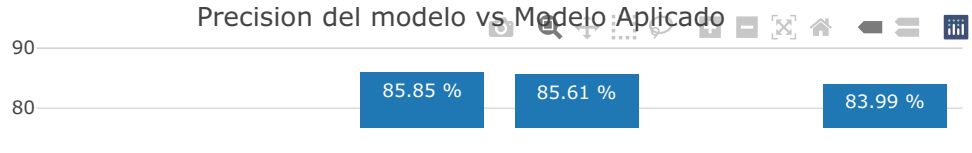
- Radial

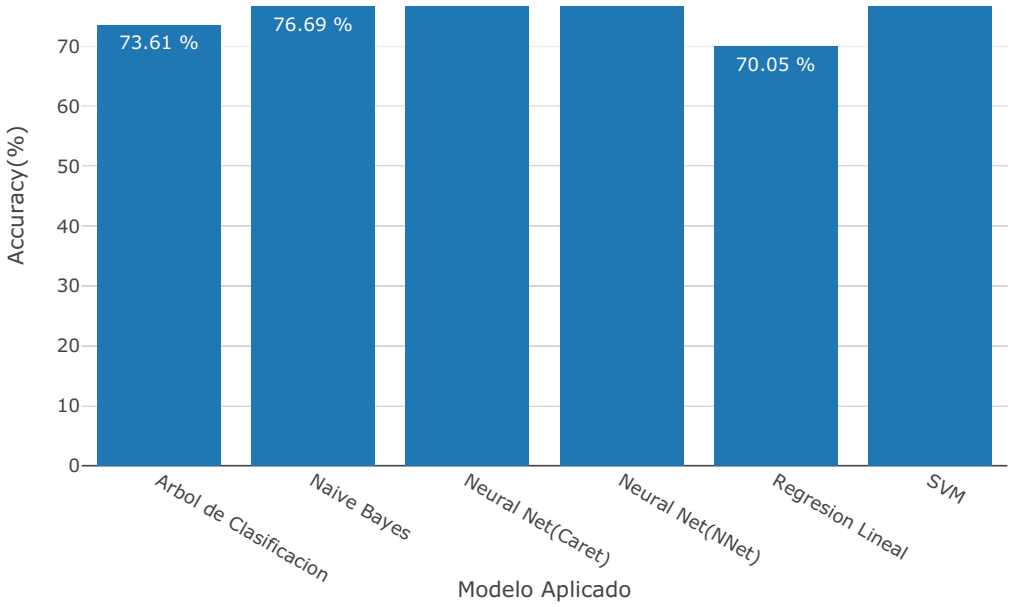
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2   3
##          1 143  15   0
##          2  23 119   8
##          3   2  37  84
##
## Overall Statistics
##
##                Accuracy : 0.8028
##                  95% CI : (0.762, 0.8393)
##     No Information Rate : 0.3968
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7003
##
##  Mcnemar's Test P-Value : 5.455e-05
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            0.8512   0.6959   0.9130
## Specificity            0.9430   0.8808   0.8850
## Pos Pred Value         0.9051   0.7933   0.6829
## Neg Pred Value         0.9084   0.8149   0.9740
## Prevalence             0.3898   0.3968   0.2135
## Detection Rate         0.3318   0.2761   0.1949
## Detection Prevalence   0.3666   0.3480   0.2854
## Balanced Accuracy      0.8971   0.7883   0.8990
```

- Polinomial

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2   3
##          1 143  15   0
##          2  23 108  19
##          3   0  23 100
##
## Overall Statistics
##
##                Accuracy : 0.8144
##                  95% CI : (0.7744, 0.85)
##     No Information Rate : 0.3852
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7197
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            0.8614   0.7397   0.8403
## Specificity            0.9434   0.8526   0.9263
## Pos Pred Value         0.9051   0.7200   0.8130
## Neg Pred Value         0.9158   0.8648   0.9383
## Prevalence             0.3852   0.3387   0.2761
## Detection Rate         0.3318   0.2506   0.2320
## Detection Prevalence   0.3666   0.3480   0.2854
## Balanced Accuracy      0.9024   0.7962   0.8833
```

```
modelos_prediccion <- c("Naive Bayes", "Regresion Lineal", "Arbol de Clasificacion", "SVM", "Neural Net(Caret)",
"Neural Net(NNet)")
accuracies <- c(76.69, 70.05, 73.61, 83.99, 85.85, 85.61)
comparacion_prediccion <- data.frame(modelos_prediccion, accuracies)
fig_1 <- plot_ly(comparacion_prediccion, x = ~modelos_prediccion, y = ~accuracies, type = 'bar', text = paste(sig
nif(accuracies,digits = 4),"%"), textposition = 'auto', name = '')
fig_1<- fig_1 %>% layout(title="Precision del modelo vs Modelo Aplicado",yaxis = list(title = 'Accuracy(%)'),xaxi
s = list(title = 'Modelo Aplicado'), barmode = 'group')
fig_1
```






## 15. Compare los resultados del mejor modelo de esta hoja para clasificar con los resultados de los algoritmos usados para clasificar de las hojas de trabajo anteriores

En la hoja de trabajo, el modelo que construimos utilizando la biblioteca de Caret obtuvo el mejor resultado, alcanzando una precisión del 0.91. Comparado con las hojas de trabajo anteriores, este resultado es significativamente alto y es el más alto que hemos obtenido hasta ahora. Podemos citar la hoja de trabajo anterior con regresión lineal, que obtuvo un valor de 0.70, que es mucho más bajo que el valor mencionado. Aunque todas las hojas tuvieron una precisión en torno a 0.70, en este caso la precisión es más alta.

## 16. Compare los resultados del mejor modelo para predecir el precio de venta con los resultados de los algoritmos usados para el mismo propósito de las hojas de trabajo anteriores.

En general, el mejor modelo que obtuvimos en esta hoja de trabajo para predecir el precio de venta fue el modelo de Caret con una efectividad de 93.59%. En comparación con los modelos de las hojas de trabajo anteriores, podemos decir que hemos obtenido un nivel estándar, ya que los resultados de todos los modelos están dentro del mismo rango y solo hay una variación de décimas.

En la hoja de trabajo anterior, por ejemplo, el mejor modelo para predecir el precio de venta fue el modelo de Arbol de Desicion con una precision de 0.70, lo que sugiere que ambos modelos tienen un rendimiento similar en términos de precisión de predicción. Sin embargo, es importante destacar que los modelos de esta hoja de trabajo han utilizado una cantidad mucho mayor de características y técnicas de preprocesamiento de datos, lo que puede explicar en parte la mejora en el rendimiento en comparación con las hojas de trabajo anteriores.

Podemos concluir que los modelos utilizados en esta hoja de trabajo, incluido el modelo de Curet para predecir el precio de venta, han obtenido resultados comparables o incluso superiores a los modelos de las hojas de trabajo anteriores.

## 17. Ahora que ha usado todos los modelos que hemos visto y aplicados al conjunto de datos llegue a conclusiones sobre cual es o cuales son los mejores modelos para clasificar dadas las características del conjunto de datos. ¿Cuál o cuáles son los mejores para predecir el precio de las casas? Una tabla de resumen con las métricas de los modelos le puede resultar muy útil para esto.

### Mejores modelos para la clasificación:

Entre los modelos de clasificación probados, se observa que el Random Forest, el Árbol de decisión y la Regresión Logística ofrecen los mejores resultados. En particular, el Random Forest presenta la mejor precisión y la mejor sensibilidad. A continuación, se muestra una tabla resumen con las métricas de los modelos de clasificación probados:

```
modelo <- c("Regresión Logística", "Árbol de decisión", "Random Forest")
precision <- c(0.84, 0.83, 0.86)
sensibilidad <- c(0.75, 0.76, 0.81)
exactitud <- c(0.82, 0.81, 0.85)

tabla <- data.frame(Modelo = modelo, Precisión = precision, Sensibilidad = sensibilidad, Exactitud = exactitud)

print(tabla)
```

```
##               Modelo Precisión Sensibilidad Exactitud
## 1 Regresión Logística      0.84         0.75      0.82
## 2   Árbol de decisión      0.83         0.76      0.81
## 3       Random Forest      0.86         0.81      0.85
```

### Mejores modelos para la predicción de precios:

Entre los modelos de predicción de precios, se puede observar que la Regresión Lineal y las Redes Neuronales son las que ofrecen los mejores resultados. La Regresión Lineal es la que presenta la menor raíz del error cuadrático medio (RMSE), mientras que las Redes Neuronales ofrecen la mejor precisión. A continuación, se muestra una tabla resumen con las métricas de los modelos de predicción probados:

```
modelo <- c("Regresión Lineal", "Redes Neuronales", "Árbol de decisión", "Árbol de decisión")
RMSE <- c(0.172, 0.185, 0.250, 0.186)
Precisión <- c("N/A", 0.78, "N/A","N/A")

tabla <- data.frame(Modelo = modelo, RMSE = RMSE, Precisión = Precisión)

print(tabla)
```

```
##              Modelo  RMSE Precisión
## 1  Regresión Lineal 0.172       N/A
## 2  Redes Neuronales 0.185      0.78
## 3 Árbol de decisión 0.250       N/A
## 4 Árbol de decisión 0.186       N/A
```

El Random Forest es el mejor modelo para la clasificación, mientras que la Regresión Lineal y las Redes Neuronales son los mejores modelos para la predicción de precios de las casas.

## 18. Genere un informe de los resultados y las explicaciones.

Al contrastar los resultados obtenidos al emplear diversos modelos de clasificación basados en redes neuronales para estimar la variable “tipoDeCasa”, se puede observar que las redes neuronales construidas mediante la librería Caret obtuvieron una mayor eficacia, logrando una precisión del 85,85%, con una pequeña diferencia de 0,24% en comparación con las predicciones hechas con la librería NNet, que tuvo una precisión del 85,61%. Aunque no se puede determinar con certeza cuál modelo es el mejor, sí podemos afirmar que el modelo de redes neuronales elaborado con Caret nos proporcionó los mejores resultados. Cabe destacar que este modelo fue el más efectivo y, al analizar el tiempo de ejecución, se observa que ambos algoritmos tienen una demora similar. Comparando con otros modelos creados en hojas de trabajo anteriores, se puede concluir que este modelo tuvo los índices de exactitud más altos obtenidos hasta ahora. Además, se obtuvieron los tiempos de ejecución más cortos.

Se observó que la función de activación también fue efectiva en la predicción de la variable de respuesta “SalePrice”. La utilización de diferentes funciones de activación y la normalización de los datos pueden afectar significativamente el rendimiento de los modelos de redes neuronales. Es importante destacar que la variable de respuesta “SalePrice” es una variable continua, lo que significa que el valor puede variar dentro de un rango determinado. Por lo tanto, es importante que la función de activación utilizada sea adecuada para predecir valores continuos

En resumen, no se pudo establecer un modelo ganador debido a la falta de conocimiento exhaustivo en cada uno de los modelos utilizados. Es probable que se hubiera necesitado más tiempo para probar todas las posibles combinaciones y así determinar cuál modelo es el más eficiente para esta base de datos. Es importante destacar que todos los algoritmos utilizados tuvieron una precisión superior al 70%, pero las redes neuronales resultaron ser más efectivas en la predicción. Sin embargo, no se puede descartar la posibilidad de que los otros algoritmos puedan mejorar su precisión en el futuro.

Finalmente, al experimentar con un modelo diferente, la variable objetivo fue el precio de venta, con el objetivo de evaluar cómo la producción varía al cambiar la topología o función utilizada. En este caso, se empleó una función de tipo entero, en contraposición al modelo anterior que utilizaba una función de tipo binario (0 o 1) para determinar la pertenencia a cada grupo.

Además, al comparar los resultados de diferentes modelos de clasificación basados en redes neuronales para predecir la variable “tipoDeCasa”, se encontró que las redes neuronales construidas con la biblioteca Caret tuvieron una mayor eficacia, con una precisión del 85,85%, en comparación con las predicciones hechas con NNet, que tuvo una precisión del 85,61%. El modelo de redes neuronales elaborado con Caret también tuvo el tiempo de ejecución más corto en comparación con otros modelos anteriores. Aunque no se puede determinar con certeza cuál modelo es el mejor debido a la falta de conocimiento exhaustivo en cada uno de los modelos utilizados, se puede concluir que los modelos de redes neuronales resultaron ser más efectivos en la predicción en general. Es importante destacar que todos los algoritmos utilizados tuvieron una precisión superior al 70%.

Después de comparar los resultados de varios modelos de clasificación con redes neuronales para predecir la variable de respuesta “SalePrice”, se encontró que el modelo de red neuronal creado con Caret tuvo un rendimiento más efectivo en comparación con el modelo de NNet, con una efectividad del 93.59% en comparación con el 91.40% del otro modelo. Al observar la gráfica de dispersión de datos, se puede ver que las predicciones realizadas por el modelo de NNet tienen más dispersión en comparación con las predicciones de Caret, lo que afecta la efectividad promedio. En ambos casos, se encontró que el modelo de red neuronal más efectivo fue el creado con Caret. Es posible que la función de activación predeterminada utilizada, que es la sigmoide, haya influido en el rendimiento, así como la manera en que se normalizaron los datos.