Pontificia Universidad Católica de Chile
Escuela de Administración
Machine Learning Para Negocios (EAA3707-1)
Profesora: María Ignacia Vicuña

Tarea 2:
Machine Learning Para Negocios

Nombres:
Vicente Jaramillo
José Vilchez

Fecha de entrega:
25 de Noviembre, 2022

Pregunta 1:

(a) Modifique el nivel de referencia de la variable respuesta, ya que por defecto R considera el menor valor como clase de referencia. Lo anterior se puede hacer con el siguiente código:

```
datos = datos%>% mutate(Purchased =
factor(Purchased,levels=c(1,0)))
```

Se modifica el nivel de referencia de la variable respuesta.

(b) Ordene el dataframe de manera ascendente según la probabilidad predicha por el clasificador h().

Se ordenan el dataframe de manera ascendente según la probabilidad predicha por el clasificador h().

```
## knn:
knn_data = data[order(data$pred_knn),]
knn_data
## svm:
svm_data = data[order(data$pred_svm),]
svm_data
```

(c) Cree el vector thresholds que será igual a los valores predichos utilizando el clasificador h(). Elimine los valores repetidos y agregue el valor 0 y 1 si no está considerado.

Se crean los vectores thresholds para knn y svm:

```
thresholds_knn = sort(unique(c(0, knn_data$pred_knn, 1)))
thresholds_svm = sort(unique(c(0, knn_data$pred_svm, 1)))
```

(d) Para cada valor t del vector thresholds, calcule la sensitividad y la especificidad del clasificador h() a partir de la siguiente regla de decisión:

- **Las observaciones x donde $h(x) \geq t$ se consideran como positivas, $\hat{y} = 1$.**
- **Las observaciones tales que $h(x) < t$ se consideran en la clase negativa, $\hat{y} = 0$.**

Puede ser de utilidad usar la función specificity() y sensitivity() para el cálculo de la sensitividad y especificidad.

A continuación, para cada valor "t" del vector, se utiliza el clasificador h(x) para clasificar las observaciones como negativas o positivas (tanto para knn como para svm), como se muestra a continuación:

- Para KNN:

```
##knn
df_knn = data.frame(matrix(nrow = length(knn_data$pred_knn)))

pos_truth_knn = c()
false_pos_knn = c()
for (i in 1:length(thresholds_knn)) {
  y_hat = c()
  for (j in 1:length(knn_data$pred_knn)) {
    if(knn_data$pred_knn[j] >= thresholds_knn[i]){
      y_hat[j] = 1
    }else{
      y_hat[j] = 0
    }
  }
  df_knn = cbind(as.factor(y_hat), df_knn)
  pos_truth_knn[i] =
sensitivity_vec(knn_data$Purchased,factor(y_hat, levels=c(1,0)))
  false_pos_knn[i] = 1 - specificity_vec(knn_data$Purchased,
factor(y_hat, levels=c(1,0)))
}

df_auc_knn = data.frame(false_pos_knn,pos_truth_knn)
```

- Para SVM:

```
## svm
df_svm = data.frame(matrix(nrow = length(svm_data$pred_svm)))

pos_truth_svm = c()
false_pos_svm = c()
for (i in 1:length(thresholds_svm)) {
  y_hat = c()
```

```
  for (j in 1:length(svm_data$pred_svm)) {
    if(svm_data$pred_svm[j] >= thresholds_svm[i]){
      y_hat[j] = 1
    }else{
      y_hat[j] = 0
    }
  }
  df_svm = cbind(as.factor(y_hat), df_svm)
  pos_truth_svm[i] =
sensitivity_vec(svm_data$Purchased,factor(y_hat, levels=c(1,0)))
  false_pos_svm[i] = 1 - specificity_vec(svm_data$Purchased,
factor(y_hat, levels=c(1,0)))
}
```

(e) Grafique la tasa de falsos positivos (eje x) versus la tasa de verdaderos positivos (eje y) para cada uno de los clasificadores.

A continuación se muestran los gráficos:

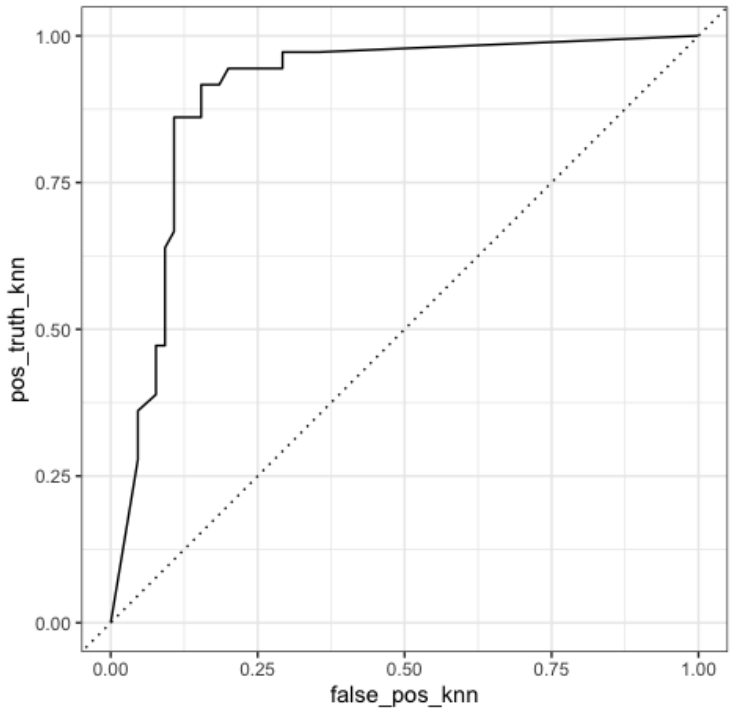

Fig. 1: Gráfico de la tasa de falsos positivos versus la tasa de verdaderos positivos para el clasificador SVM.

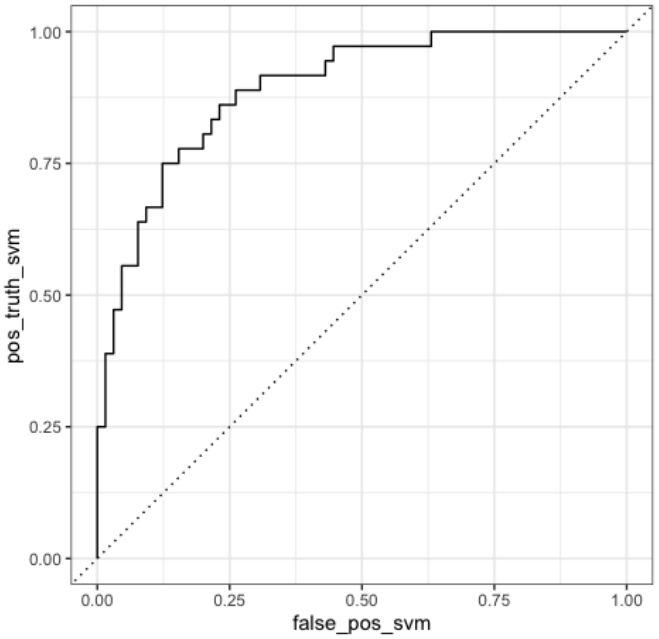

Fig. 2: Gráfico de la tasa de falsos positivos versus la tasa de verdaderos positivos para el clasificador KNN.

(f) Tidymodels tiene implementado el cálculo de la curva de ROC con la función roc curve( ) y el cálculo del área bajo la curva roc auc( ). Utilice ambas funciones y compare con lo obtenido anteriormente.

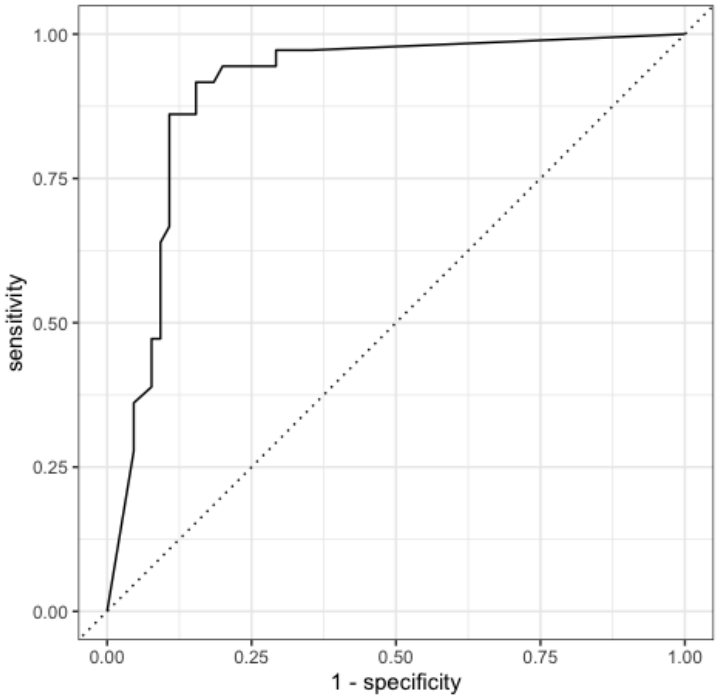

Fig. 3: Gráfico de la curva de ROC para el clasificador KNN.

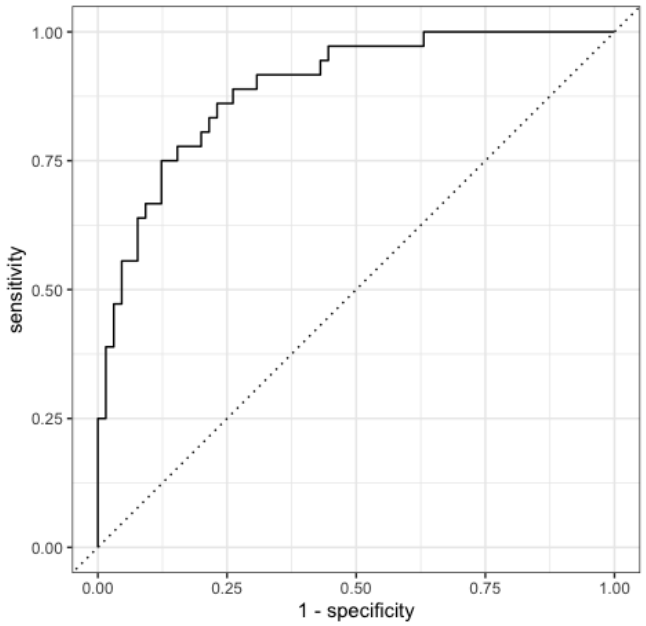

Fig. 4: Gráfico de la curva de ROC para el clasificador SVM.

Pregunta 2:

1. Descargue los datos de la librería ”keras”de R, y extraiga una muestra de tamaño n = 2000, que sería el conjunto de datos que utilizaremos. Debe instalar previamente la librería keras y posteriormente ejecutar el siguiente código:

```
library(keras)
    mnist =  dataset_mnist()
    train_images =  mnist$train$x
    train_labels  =  mnist$train$y
    set.seed(123)
    index = sample(1:60000,2000, replace = FALSE)
    sample_images  =  data_images[index,,]
    sample_y       =  data_labels[index]
```

2. Utilice el siguiente código para visualizar la primera imagen del data set:

```
plot(as.raster(sample_images[1,,],max=255))
```

¿Qué números escritos a mano contienen las primeras 5 imágenes del conjunto de datos sample images?

Se obtienen las primeras cinco imágenes del conjunto de datos, las cuales fueron: **6, 8, 7, 7, 5.**

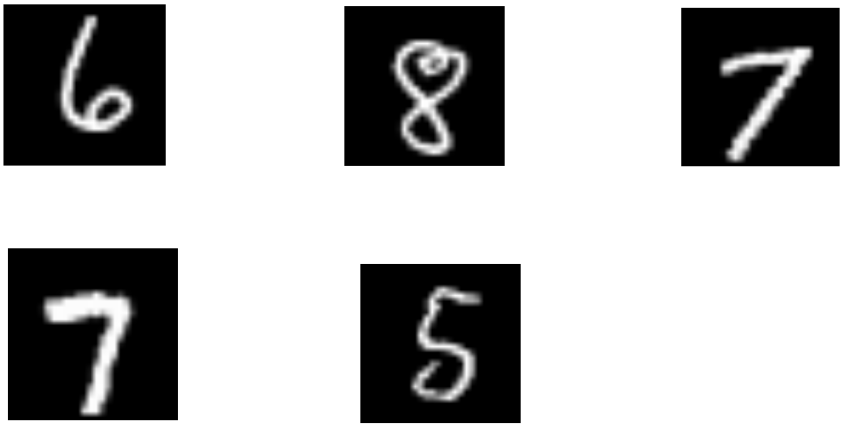

Fig. 5: 5 primeras imágenes del conjunto de datos.

3. Cree un data frame que contenga las imágenes y la variable respuesta. Luego utilice la semilla set.sed(3707) y realice una partición al 90 % para los datos de entrenamiento.

Creamos el dataframe con todas las imágenes en cada fila y sus píxeles en cada columna:

```
df = as.data.frame(sample_images)
# Añadimos una nueva columna con los valores respuesta y cambiamos
el nombre de la col por amor al arte
df = cbind(as.factor(sample_y), df)
colnames(df)[1] = 'var_response'

# Realizamos la partición del 90%
set.seed(3707)
data_split <- rsample::initial_split(df, prop = 0.9)

## Los set de datos:
# Entrenamiento
data_train  = rsample::training(data_split)
# Prueba
data_test  = rsample::testing(data_split)
```

4. Ajuste un SVM lineal con margen suave a los datos de entrenamiento. Realice validación cruzada para determinar el valor óptimo del parámetro C. Utilice k = 5 folds, grilla aleatoria de 10 valores y seleccione según la métrica roc auc.

Utilizando los valores señalados se obtuvo que el valor óptimo del parámetro C es: 0.00132 con un roc auc de 93,5%.

5. Ajuste un SVM no lineal utilizando el kernel RBF a los datos de entrenamiento. Realice validación cruzada para determinar el valor óptimo de los parámetros C y σ. Utilice k = 5 folds, grilla aleatoria de 10 valores y seleccione según la métrica roc auc.

Utilizando los valores señalados se obtuvo que el valor óptimo del parámetro C es de 14.3 y el del σ es de 1.09e-7 con un roc auc de 93.6%.

6. Para cada uno de los ajustes de SVM lineal y no lineal, entregue la matriz de confusión y calcule la tasa de clasificación errónea para el conjunto de prueba. Comente los resultados.

- Matriz de confusión SVM lineal:

	Verdadero									
Predicción	0	1	2	3	4	5	6	7	8	9
0	24	0	0	0	0	1	1	0	0	0
1	0	17	0	0	1	0	0	0	0	0
2	0	0	18	0	1	0	1	0	0	1
3	0	0	0	17	0	0	0	0	1	2
4	0	0	0	0	21	0	0	0	2	1
5	0	0	0	1	0	13	1	0	3	1
6	0	0	0	0	0	0	15	0	0	0
7	0	0	1	1	1	0	0	11	0	1
8	0	0	2	0	0	0	0	0	20	0
9	0	0	0	0	0	0	0	0	0	20

Con esto se tiene que la precisión para el conjunto de prueba es del 88%, por lo tanto, la tasa de clasificación errónea es de un 12%.

- Matriz de confusión SVM no lineal:

	Verdadero									
Predicción	0	1	2	3	4	5	6	7	8	9
0	24	0	0	0	0	0	1	0	0	0
1	0	17	0	0	0	0	0	0	1	0
2	0	0	18	0	1	0	1	0	0	0
3	0	0	0	17	0	0	0	0	0	2
4	0	0	0	0	23	0	0	0	0	0
5	0	0	0	2	0	14	0	0	2	1
6	0	0	0	0	0	0	16	0	0	0
7	0	0	1	0	0	0	0	11	0	1
8	0	0	2	0	0	0	0	0	22	0
9	0	0	0	0	0	0	0	0	1	22

Con esto se tiene que la precisión para el conjunto de prueba es del 92%, por lo tanto, la tasa de clasificación errónea es de un 8%.

Con estos resultados podemos observar que SVM no lineal tiene una mejor precisión que el SVM lineal, lo cual se puede explicar debido a que el algoritmo de SVM no-lineal está clasificando mejor en áreas donde el algoritmo de SVM lineal, por la posibilidad de utilizar kernels en la clasificación de los datos y mejorar la precisión.

Pregunta 3:

(a) Inspeccione la estructura de los datos con el comando str( ) y verifique si las variables numéricas y categóricas están en el formato correcto. De no serlo, modifique las numéricas y factor.

A continuación se revisan las variables y se modifican para que estas estén en el formato correcto:

```
str(df)

df <- df %>%
  dplyr::mutate(gender = factor(gender, levels = c("Male",
"Female", "Other"))) %>%
  dplyr::mutate(hypertension = factor(hypertension, levels =
c("1", "0"),
                                      labels = c("Yes", "No"))) %>%
  dplyr::mutate(heart_disease = factor(heart_disease, levels =
c("1", "0"),
                                       labels = c("Yes", "No"))) %>%
  dplyr::mutate(ever_married = factor(ever_married, levels =
c("No", "Yes"))) %>%
  dplyr::mutate(work_type = factor(work_type, levels =
c("children",
"Govt_jov","Never_worked","Private","Self-employed"))) %>%
  dplyr::mutate(Residence_type = factor(Residence_type, levels =
c("Rural", "Urban"))) %>%
  dplyr::mutate(smoking_status = factor(smoking_status, levels =
c("formerly smoked", "never smoked","smokes","Unknown"))) %>%
  dplyr::mutate(stroke = factor(stroke, levels = c("1", "0")))

glimpse(df)
```

(b) ¿Qué porcentaje de pacientes no revela el status de fumador? ¿Qué sugiere hacer con ese grupo de pacientes para el modelamiento?

```
table(df$smoking_status)
round(table(df$smoking_status)/nrow(df) * 100,2)
```

Donde el porcentaje de los estatus de fumador es el siguiente:

formerly smoked	never smoked	smokes	Unknown
17.32%	37.03%	15.44%	30.22

El 30,22% de los pacientes no revela el estatus de fumador. Además si observamos el porcentaje que tiene cada estatus de fumador respecto a la variable Stroke, notamos que no se observa una gran diferencia. Es por esta razón, que en función del análisis se eliminará este grupo de pacientes del modelamiento.

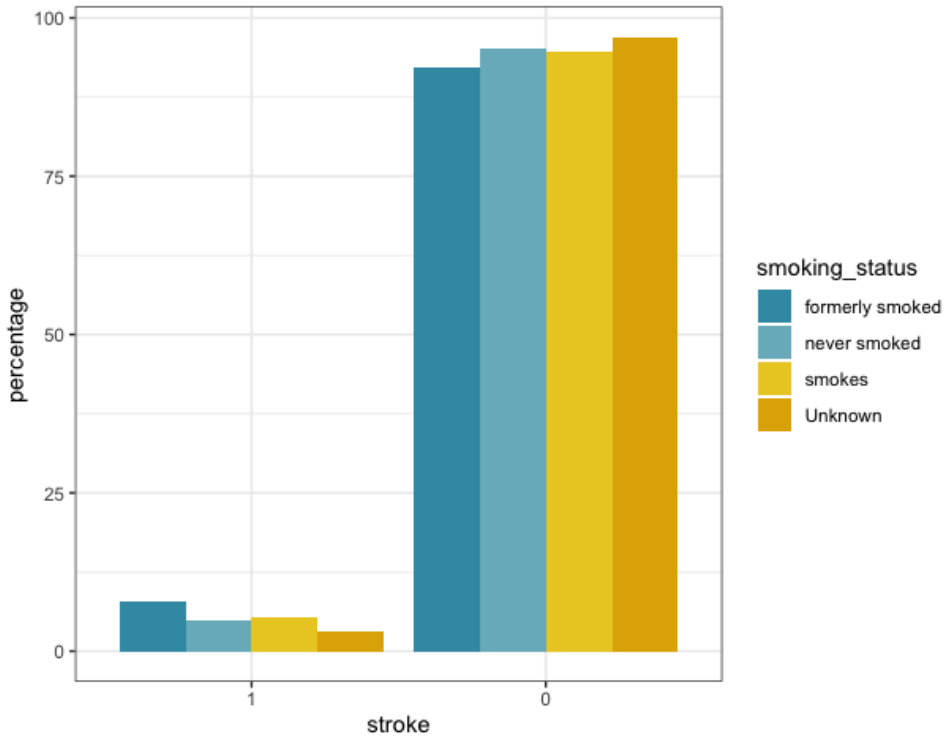

Fig. 6: Porcentaje que tiene cada estatus de fumador respecto a la variable "stroke".

(c) Sin excluir los valores missing, separe la data en conjunto de entrenamiento (80 %) y muestra de validación (20 %). Utilice la variable stroke para estratificar el muestreo.

Se procede a separar la data, sin excluir los valores missing de Unknown:

```
set.seed(314)

data_split     =   initial_split(df, prop = 0.80)
data_train     = data_split %>% training()
```

```
data_test      = data_split %>% testing()
data_folds =  vfold_cv(data_train , v = 10)
```

(d) Los árboles de decisión pueden manejar los valores faltantes mediante el uso de predictores sustitutos, considerando la categoría missing como un nivel del factor. Ajuste un árbol de clasificación a la muestra de entrenamiento, considerando los siguientes valores de los hiper parámetros: cost complexity = 0, tree depth = 20 , min n = 15, para obtener un árbol "grande".

```
# Especificación del modelo
tree_model_large  =  decision_tree(cost_complexity =  0,
                                   tree_depth = 20,
                                   min_n = 15) %>%
  set_engine('rpart') %>%
  set_mode('classification')

# Especificación de la receta
df_recipe = recipe(stroke ~ ., data = data_train )

df_recipe %>%
  prep() %>%
  bake(new_data = data_train)%>%View()

# Modelo
tree_workflow_large  =  workflow() %>%
  add_model(tree_model_large) %>%
  add_recipe(df_recipe)

# Fit a Model

tree_wf_fit_large  =  tree_workflow_large %>%
  fit(data = data_train)


prediccion_large = predict(tree_wf_fit_large, new_data =
data_test)
prediccion_large_train = predict(tree_wf_fit_large, new_data =
data_train)

data_test  =  data_test%>%
  mutate(pred_large = prediccion_large$.pred_class)
```

```
data_train1  =  data_train%>% mutate(pred_large_train =
prediccion_large_train$.pred_class)
```

(e) Grafique el árbol de clasificación obtenido e interprete los resultados. ¿Cuál es la tasa de error del conjunto test?

```
# Decision Tree Plot

tree_fit_large  =  tree_wf_fit_large %>%
  extract_fit_engine()
View(tree_fit_large$splits)

par(xpd = TRUE)
plot(tree_fit_large, compress = TRUE)
#text(tree_fit_large, use.n = TRUE)

# Tasa de error conjunto (pendiente)
rpart.plot::prp(tree_fit_large, type = 0, fallen.leaves = TRUE,
                tweak = 1.3, roundint = FALSE)


# tasa de error del conjunto

conf_mat(
  data = data_train1,
  truth = stroke,
  estimate = pred_large_train
  )

acc_test <- accuracy(
  data = data_train1,
  truth = stroke,
  estimate = data_train1$pred_large_train
)
acc_test
```

Observamos un accuracy de 97,4%.

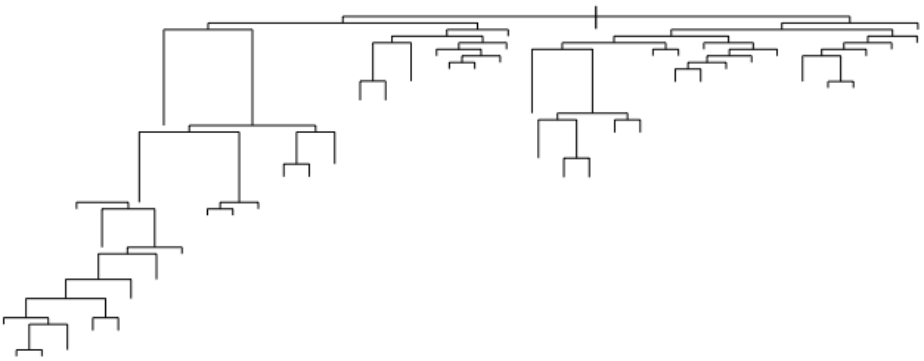

Fig. 7: Árbol de clasificación.

(f) Realice 10-fold validación para determinar los valores óptimos de los hiperparametros: cost complexity() y tree depth(). Para ello utilice una grilla regular con el rango de cada hiperparámetro que trae por defecto y utilice levels = 5. Seleccione el conjunto de parámetros que tenga mayor área bajo la curva (AUC).

El conjunto de parámetros que tiene un mayor área bajo la curva AUC, son: ***cost complexity*: 0.0101** y también ***tree depth*: 4.**

```
df_recipe = recipe(stroke ~ ., data = data_train )

df_recipe %>%
  prep() %>%
  bake(new_data = data_train)


model_tree_tune<- decision_tree(cost_complexity = tune(),
                                tree_depth = tune())         %>%
  set_engine("rpart")                 %>%
  set_mode("classification")


workflow_tree_tune = workflow() %>%
  add_model(model_tree_tune) %>%
  add_recipe(df_recipe)
```

```
min_n()
cost_complexity()

parallel::detectCores()

cl <- parallel::makeCluster(5)
doParallel::registerDoParallel(cl)

cv_tree = workflow_tree_tune %>%
  tune_grid(data_folds,
            metrics = metric_set(roc_auc),
            grid=10)

parallel::stopCluster(cl)

cv_tree %>% show_best(metric = 'roc_auc')

## Select best model based on accuracy
best_tree  =  cv_tree %>%
  select_best(metrics = roc_auc)

best_tree
```

(g) Con los valores encontrados, pode el árbol y calcule la tasa de error del conjunto test. ¿Mejora la poda la tasa de error en el conjunto de pruebas? Grafique el árbol podado e interprete los resultados.

Notamos que en el conjunto de pruebas el accuracy empeora levemente en comparación al conjunto de entrenamiento.

Accuracy train set	Accuracy test set
97,4%	95,5%

El árbol podado queda de la siguiente manera:

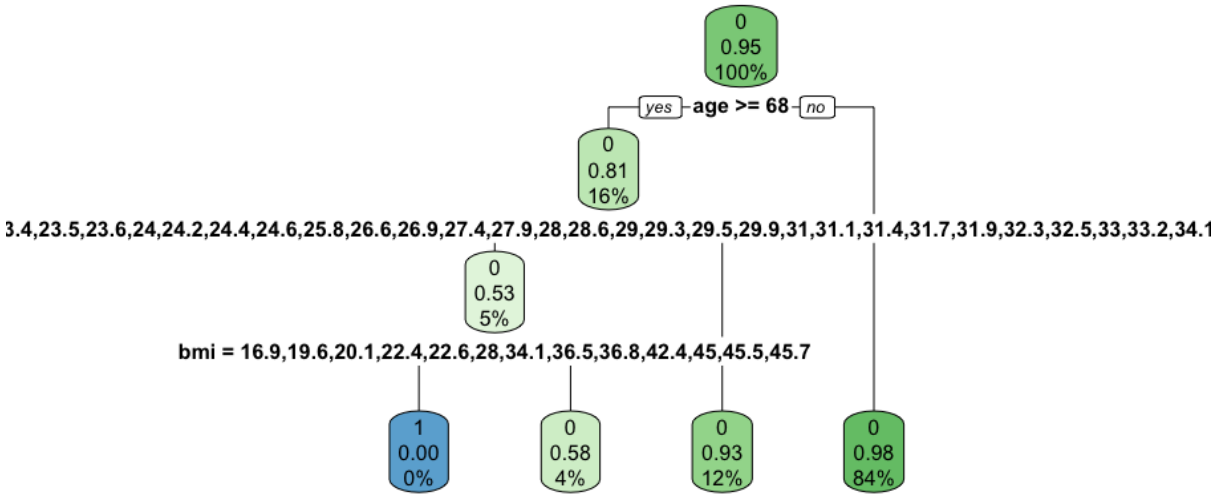

Fig. 8: Árbol podado.

(h) Calcule la importancia de la variable para cada predictor para el árbol podado.

Como se puede observar, las 3 variables más importantes serían: ***bmi, age y avg_glucose_level.***

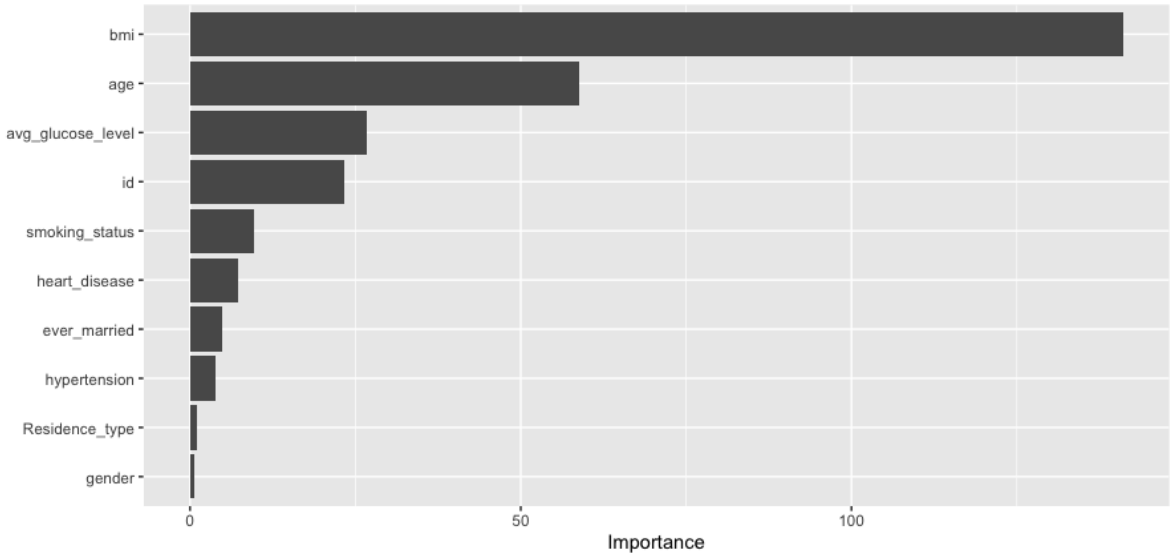

Fig. 9: Árbol podado.

(i) Utilice el enfoque de bagging tree para construir un modelo de clasificación. ¿Qué tasa de error de prueba se obtiene? ¿Qué variables son las más importantes?. Comente los resultados.

Se obtiene un accuracy de 95,4%.

(j) Utilice Random Forests para construir un modelo de clasificación. Obtenga el error de la prueba, la importancia y discuta el efecto de mtry en la tasa de error. Comente los resultados.