Universidad del Desarrollo
Facultad de Ingeniería

Certamen 2

Taller de Inteligencia de Negocios

Docente:

Mauricio Herrera

Integrantes:

Jan Frese

Jorge Ramírez

Marcelo Cáceres

Problema 1

Para el desarrollo de este problema ocuparemos el dataset “myocarde.csv”, se limpian todas las variables y eliminan gráficos existentes para comenzar a trabajar.

```
# PROBLEMA 1: PRONÓSTICO DE SOBREVIDA EN INFARTO
################################################################################
rm(list=ls()) # Para limpiar todas las variables
graphics.off() # Borra todos los gráficos
setwd("C:/Users/jorge/OneDrive/Escritorio/SEM 6/")
datos <- read.csv("TIN/Certamen2/myocarde.csv")
```

Pregunta 1:

(A) Se solicita hacer un entrenamiento con los datos 80/20, 80% para “train” y 20% para “test”, luego utilizamos “set.seed(123)” y entrenamos según los valores dados.

Para construir el modelo de clasificación SVM con kernel lineal y verificar el valor del costo C, primero, confirmamos con “tune.out” cuál es el mejor costo para el modelo SVM que explique “PRONO” (variable dependiente binaria) de acuerdo a los costos postulados, obteniendo de esta manera que el mejor para el modelo SVM con kernel lineal, al iterar 10 veces es el 0.1, que será el costo a utilizar.

```
# (A) Construya un modelo de clasificación SVM con kernel lineal. Verifique el valor del costo C
library(e1071)
tune.out <- tune(svm,PRONO~.,data=train,kernel="linear",ranges=list(cost=c(0.001, 0.01, 0.1,
1,5,10,100,1000)))
summary(tune.out)
bestmod <- tune.out$best.model

svmfit <- svm(PRONO~., data=datos, kernel="linear", cost=0.1,scale=FALSE)
svmfit$index #indica los vectores soportes
summary(svmfit)
```

(B) Para generar la tabla de confusión y evaluar el desempeño del clasificador con tal de comentar sobre la exactitud, precisión, especificidad y sensibilidad, se ocupó el comando “predict” con los datos “train” (exceptuando la variable de decisión 8) y se obtuvieron los siguientes valores de la tabla:

- Exactitud: 89%
- Precisión: 86%
- Especificidad: 86%
- Sensibilidad: 91%

Dado esto, se puede concluir que el clasificador generó una tabla de confusión bastante “buena”, ya que todos sus porcentajes evaluados están por sobre el 85%.

```
# (B) Construya la tabla de confusión para evaluar el desempeño del clasificador y

# comente sobre la exactitud, precisión, especificidad y sensibilidad obtenidos.

p <- predict(svmfit,train[,-8])


tabla <- table(predichos=p,reales=train$PRONO)

tabla


TP=tabla[1,1]

TN=tabla[2,2]

FP=tabla[1,2]

FN=tabla[2,1]


(exactitud=(TP+TN)/(TP+TN+FN+FP))

(precision=TP/(TP+FP))

(sensibilidad=TP/(TP+FN))

(Especificidad=TN/(TN+FP))
```

(C) Para construir las curvas ROC, se ocuparon las librerías “ROCR” y “pROC”, se define el gráfico ROC y se construye para los datos “train”, obteniendo que el área bajo la curva abarca el 94% de los datos del clasificador.

Mismo caso para los datos test, se obtiene que el área bajo la curva abarca un 87% de los datos.

```
# (C) Construya la curva ROC para el clasificador y determine la AUC. Comente el resultado.

library(ROCR)
library(pROC)
#Define la funcion para comparar la prediccion con los valores reales y graficar la curva ROC
rocplot=function(predichos, reales, ...){
  predob = prediction(predichos, reales)
  perf = performance(predob,"tpr","fpr")
  plot(perf,...)}

library(dplyr)
svmfit.opt=svm(PRONO~., data=train, kernel="linear", cost=0.1,decision.values=T)
fitted=attributes(predict(svmfit.opt,train,decision.values=TRUE))$decision.values
par(mfrow=c(1,2))
rocplot(fitted,train$PRONO,main="Training Data")
pred=predict(svmfit.opt,train,decision.values=T,probability = T)
val <- attributes(pred)$decision.values
q_SVM <- roc(predictor = as.numeric(val), response = train$PRONO,
             levels = levels(train$PRONO))
plot(q_SVM, col = "#9E0142")
auc(q_SVM)

svmfit.opt1=svm(PRONO~., data=test, kernel="linear", cost=0.1,decision.values=T)
fitted1=attributes(predict(svmfit.opt,test,decision.values=TRUE))$decision.values
rocplot(fitted1,test$PRONO,main="Test Data")
pred1=predict(svmfit.opt1,train,decision.values=T,probability = T)
val1 <- attributes(pred1)$decision.values
q_SVM1 <- roc(predictor = as.numeric(val1), response = train$PRONO,
              levels = levels(train$PRONO))
plot(q_SVM1, col = "#9E0142")
auc(q_SVM1)
```

Pregunta 2:

(A) Se solicita realizar el mismo tipo de modelo que la pregunta 1 pero con kernel radial, para esto buscamos el mejor costo y gamma para el modelo, tras iterar 10 veces se encontró que los mejores valores eran 10 y 1 respectivamente. Por lo que armamos nuestro modelo SVM de kernel radial con costo = 10 y gamma = 1.

Tras realizar el "summary", se puede observar que el modelo estableció 56 vectores soporte.

```
# Pregunta 2:
# (A) construya un modelo SVM con kernel radial. Utilice distintos valores
# de costo C y gamma para mejorar, si es posible, el desempeño del clasificador
tune.out1=tune(svm, PRONO~., data=train, kernel="radial", ranges=list(cost=c(0.1,1,10,100,1000),
                                                    gamma=c(0.5,1,2,3,4)))
summary(tune.out1)
svmfit1=svm(PRONO~., data=train, kernel="radial",gamma=1,cost=10)
svmfit1$index #indica los vectores soportes
summary(svmfit1)
```

(B) Para efectos de encontrar el mejor modelo y realizar las predicciones, se obtuvo el área bajo la curva (AUC) del kernel lineal para el SVM con los datos "train", obteniendo así un valor de 94%.

Por otro lado, el área bajo la curva (AUC) del modelo SVM de kernel radial con los datos "train", fue un porcentaje de 100%, por lo que abarca todos los datos y no genera margen de error, siendo kernel radial entonces el mejor modelo.

Pregunta 3:

Para visualizar los datos empleando un análisis de componentes principales y ver si los datos son separables linealmente, se ocuparon los dos primeros componentes principales. Primero se creó una variable 'x' que contenga todos los datos exceptuando la variable de decisión, luego se procede a convertir en matriz y se calcula la covarianza de 'x' para luego calcular los valores y vectores propios.

Luego se transforman las variables originales a componentes principales con la variable ‘y’ para generar el gráfico del primer componente principal vs el segundo componente principal (**fig 1**)

```
# Pregunta 3:

# (A) Visualice los datos empleando análisis de componentes pricipales y comente sobre si

# los datos son separables linealmente o no.

x <- datos[,-8]

head(x)

x<-data.matrix(x)

class(x)

dim(x)

(n = nrow(x))

cov(x)

#cálculo de los valores y vectores propios de la matriz de covarianza:

e = eigen(cov(x))

(e1 = e$values)

e$vectors

#datos multiplicados por la matriz formada por los vectores propios

y = as.matrix(x) %*% e$vectors #Hace la transformación de las variables originales a los componentes principales

head(y)

par(mfrow=c(1,1))

#plot primero vs. segundo PC

plot(y[, 1], y[, 2], pch = c(rep(1, 7), rep(3, 7)),

    col = c(rep("blue", 7), rep("red", 7)),

    xlab = "PC1", ylab = "PC2", main = "Primero vs. Segundo PC",

    cex.lab = 1, cex.axis = 1, cex.main = 1.4)

# no son linealmente separables
```

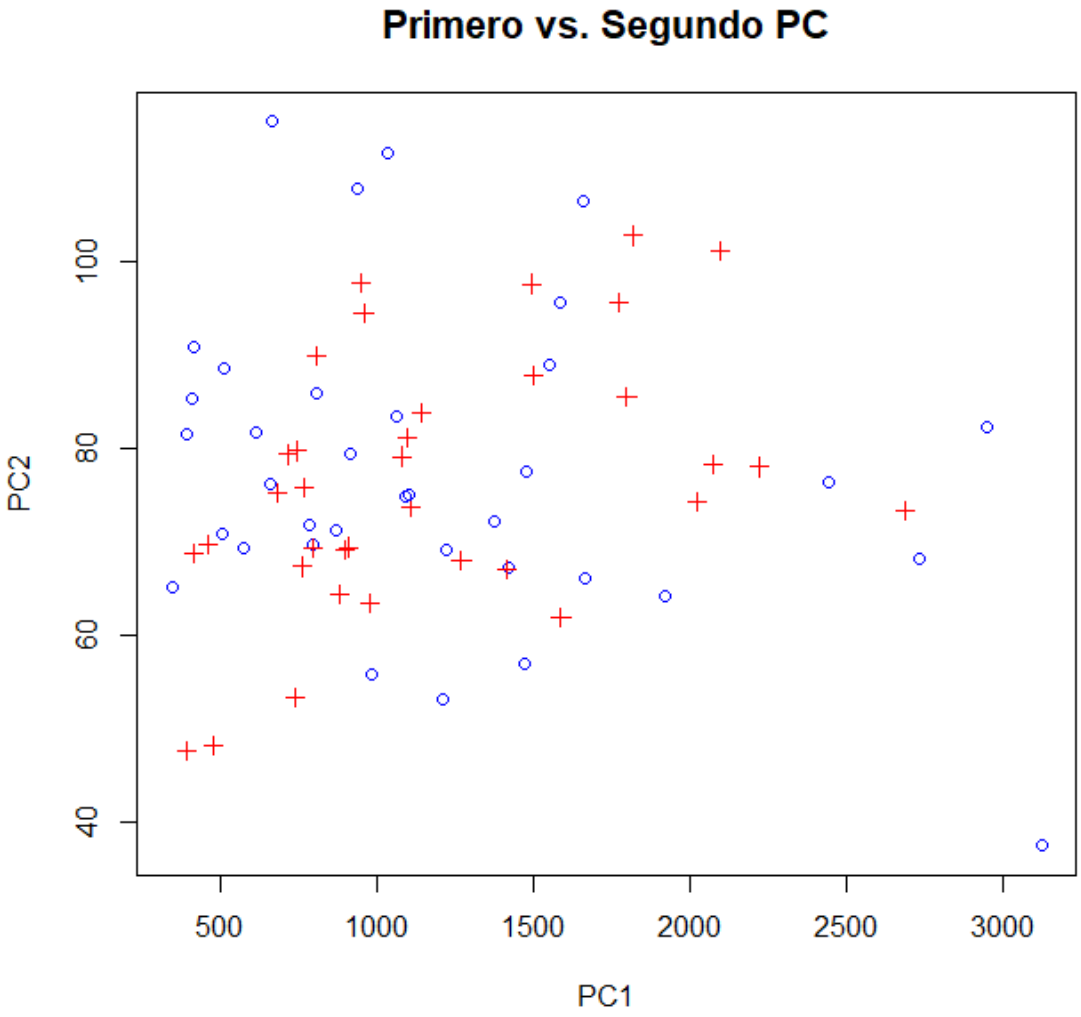

fig 1

Tras el análisis del gráfico de los componentes principales, se aprecia claramente que los datos no son linealmente separables al estar “mezclados” unos con otros.

Problema 2

Para ejercicio de este problema, se tuvo que instalar el paquete “mlbench” para obtener la base de datos con la cual se trabajará.

Primero se tuvo que identificar todos los valores “NA” para reemplazarlos por sus respectivos promedios redondeados al primer decimal en cada una de las 8 categorías para que no incidan de manera significativa en los análisis que se realizarán.

Se solicita generar una partición para los datos training y testing en una proporción 80/20, para esto se utilizó “set.seed(123)” con un 80% de los datos training y 20% de los datos testing.

```
################################################################################
# PROBLEMA 2: DIAGNÓSTICO DIABETES TIPO II
################################################################################
install.packages("mlbench")

data("PimaIndiansDiabetes2", package = "mlbench")

which(is.na(PimaIndiansDiabetes2))

PimaIndiansDiabetes2$pregnant[which(is.na(PimaIndiansDiabetes2$pregnant))] =
round(mean(PimaIndiansDiabetes2$pregnant,na.rm=TRUE),1)

PimaIndiansDiabetes2$glucose[which(is.na(PimaIndiansDiabetes2$glucose))] =
round(mean(PimaIndiansDiabetes2$glucose,na.rm =TRUE),1)

PimaIndiansDiabetes2$pressure[which(is.na(PimaIndiansDiabetes2$pressure))] =
round(mean(PimaIndiansDiabetes2$pressure,na.rm =TRUE),1)

PimaIndiansDiabetes2$triceps[which(is.na(PimaIndiansDiabetes2$triceps))] =
round(mean(PimaIndiansDiabetes2$triceps,na.rm =TRUE),1)

PimaIndiansDiabetes2$insulin[which(is.na(PimaIndiansDiabetes2$insulin))] =
round(mean(PimaIndiansDiabetes2$insulin,na.rm =TRUE),1)

PimaIndiansDiabetes2$mass[which(is.na(PimaIndiansDiabetes2$mass))] =
round(mean(PimaIndiansDiabetes2$mass,na.rm =TRUE),1)

PimaIndiansDiabetes2$pedigree[which(is.na(PimaIndiansDiabetes2$pedigree))] =
round(mean(PimaIndiansDiabetes2$pedigree,na.rm =TRUE),1)

PimaIndiansDiabetes2$age[which(is.na(PimaIndiansDiabetes2$age))] =
round(mean(PimaIndiansDiabetes2$age,na.rm =TRUE))

which(is.na(PimaIndiansDiabetes2))
```

```
set.seed(123)
n1=nrow(PimaIndiansDiabetes2)
a1=sample(c(1:n1), floor(n1*0.8),replace=F)
train1=PimaIndiansDiabetes2[a1,]
test1=PimaIndiansDiabetes2[-a1,]
```

Para la construcción de los modelos de clasificadores Naive-Bayes, Regresión Logística y SVM se utilizaron los datos training.

1. Naive-Bayes

Se ejecuta la tabla con los datos entrenados según diabetes, arrojando de esta manera 398 casos negativos y 216 positivos de las observaciones de diabetes, lo cual corresponde a un 65% y 35% respectivamente.

Luego se procede a generar el modelo Naive-Bayes con los datos training. Dado que se está entrenando el modelo con los datos “train”, las predicciones obtenidas serán respecto a los datos test. Obteniendo de esta manera una cantidad de falsos correspondientes a un 25% y la cantidad de verdaderos correspondientes a un 75% según la tabla aproximadamente.

Posterior a esto, se realiza una tabla de confusión según la predicción, obteniendo 84 verdaderos positivos y 31 verdaderos negativos.

Los valores relevantes de la tabla de confusión de este modelo fueron los siguientes:

- Exactitud: 75%
- Precisión: 80%
- Sensibilidad: 82%
- Especificidad: 60%

De acuerdo a los porcentajes entregados, se puede afirmar que el modelo es bastante “bueno” a excepción de su especificidad que está al borde de ser azarosa.

```
# Pregunta 1
# (A) Construya varios modelos de clasificadores: Naive Bayes, Regresión Logística y
# SVM (considere distintos parámetros), con los datos training.

# Datos train
```

```
# Naive-Bayes

table(train1$diabetes)

round(prop.table(table(train1$diabetes))*100)


library(e1071)


model.nb1 = naiveBayes(diabetes ~ ., train1)

pred1=predict(model.nb1,test1)


table(pred1 == test1$diabetes)/length(test1$diabetes)


table(predichos=pred1 , reales=test1$diabetes)


tabla1=table(predichos=pred1 , reales=test1$diabetes)

(TP1=tabla1[1,1])

(TN1=tabla1[2,2])

(FP1=tabla1[1,2])

(FN1=tabla1[2,1])


(exactitud1=(TP1+TN1)/(TP1+TN1+FN1+FP1))

(precision1=TP1/(TP1+FP1))

(sensibilidad1=TP1/(TP1+FN1))

(Especificidad1=TN1/(TN1+FP1))

(NegPredValue1=TN1/(TN1+FN1))

(Prevalence1=(TP1+FN1)/(TP1+FN1+FP1+TN1)) # tasa real de la clase positiva

(DetectionRate1=TP1/(TP1+FN1+FP1+TN1))

(DetectionPrevalence1=(TP1+FP1)/(TP1+FN1+FP1+TN1)) # tasa de detección de la clase positiva
del clasificador

(FalsePositiveRate1=1-Especificidad1)

(FalseNegativeRate1=1-sensibilidad1)

(F1_Score1=2*precision1*sensibilidad1/(precision1+sensibilidad1)) # muy sensible para capturar la
clase positiva

# no hay tantos falsos positivos
```

Tras definir el modelo Naive-Bayes y crear la tabla de confusión, se determinaron los valores de decisión a través de las curvas ROC mediante la librería “pROC”, donde se grafica la sensibilidad vs especificidad (fig 2), donde la cantidad de los datos bajo la curva corresponde a un 83% del total, por lo que el modelo posee un gran acierto y poco error.

Finalmente se ejecuta el comando “confusionmatrix” para efectos de confirmar los valores de la tabla determinada anteriormente.

```
library(pROC)
#Valores de decisión:
valores1=predict(model.nb1,test1,type='raw')
valores1[,1]
val1=valores1[,1]
q_NB1 <- roc(predictor = as.numeric(val1), response = test1$diabetes,
        levels = levels(test1$diabetes))
plot(q_NB1, col = "#9E0142")
# Una medida del desempeño del clasificador. Área bajo la curva (AUC)
# Mientras más cerca de 1 mejor
auc(q_NB1)

library(caret)
library(klaR)
pred=as.factor(pred1)
confusionMatrix(pred1, test1$diabetes)
```

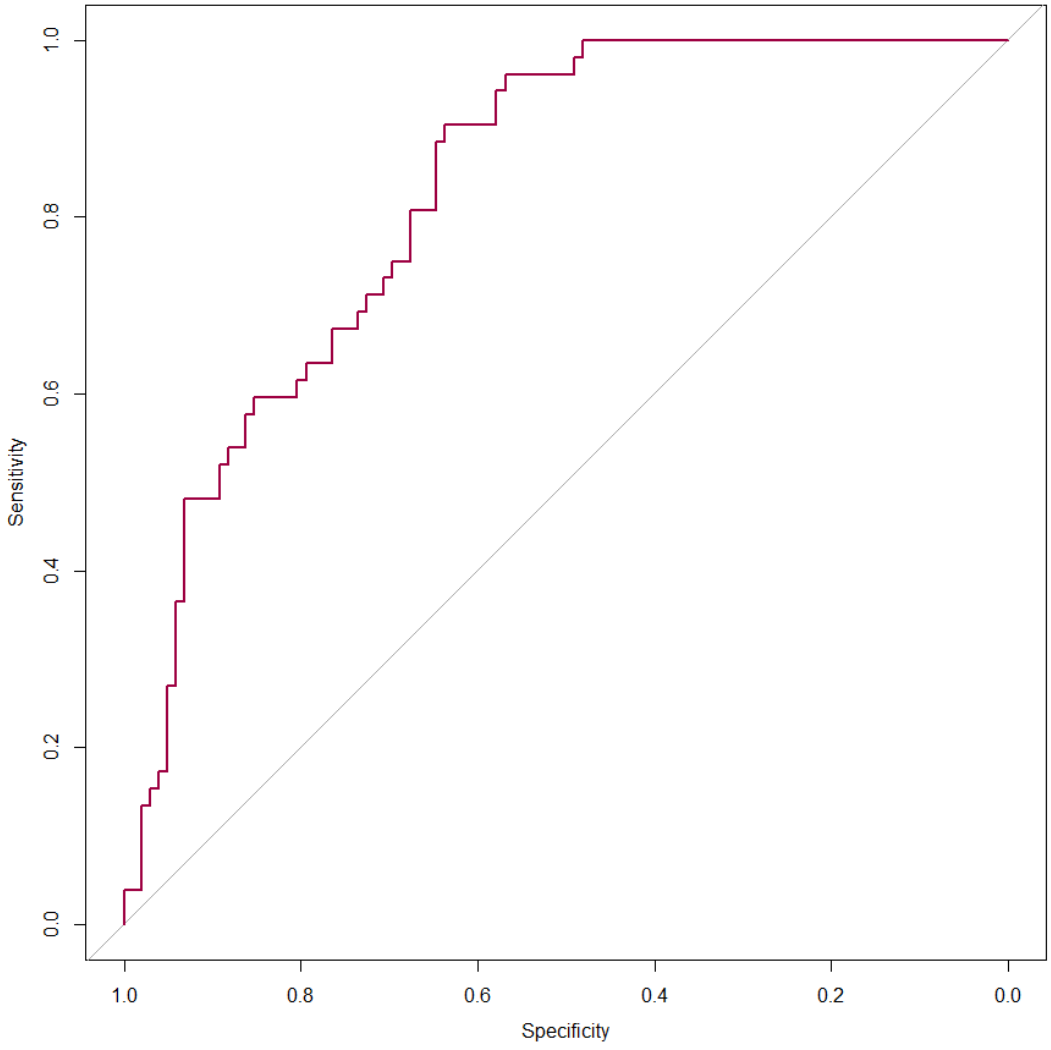

fig 2.

2. Regresión Logística

Este modelo se realizará con la variable dependiente (diabetes) y le asignamos la familia “Binomial”, ya que es de carácter binario (positivo o negativo) de acuerdo con los datos “train”, para realizar la posterior predicción con los datos “test”.

Al ejecutar el modelo se aprecia que las variables más significativas para explicar el modelo de regresión logística son: pregnant, glucosa, mass y pedigree. Dado que el p-valor de estas es menor a 0,05.

Se realiza la predicción de acuerdo con los datos “test” y se define que si la probabilidad de predecir es mayor a un 50% entonces será positivo, en caso contrario será negativo. Obteniendo de esta manera a través del comando “confusionmatrix” los valores de Especificidad (56%) y Sensibilidad (88%).

Finalmente se grafica a través de las curvas ROC para obtener la cantidad de datos bajo la curva de este modelo (fig 3), obteniendo que el modelo es bastante eficaz, dado que abarca gran cantidad de los datos correspondiente al 84%.

```
# Regresion logistica
modelo = glm(diabetes ~ ., train1, family='binomial')
summary(modelo)
pRL = predict(modelo,test1,type='response')
labels = ifelse(pRL > 0.5,'pos','neg')
table(labels==test1$diabetes)/length(test1$diabetes)
table(predichas=labels,reales=test1$diabetes)
predRL=as.factor(labels)
confusionMatrix(predRL, test1$diabetes)

val=pRL
q_logit <- roc(predictor = as.numeric(val), response = test1$diabetes,
               levels = levels(test1$diabetes))
plot(q_logit, col = "#9E0142")
auc(q_logit)
```

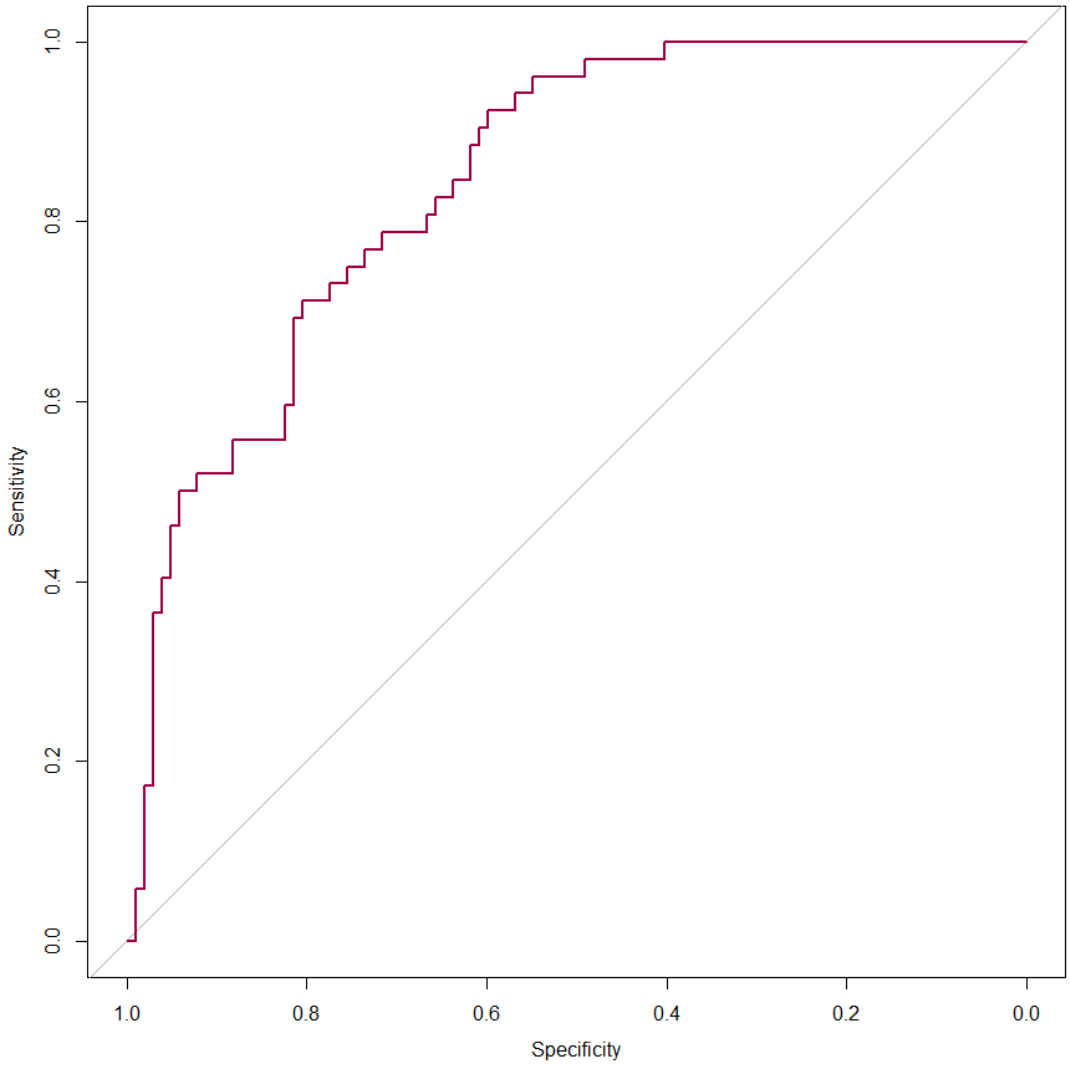

fig 3

3. SVM

Se entrena el modelo SVM con los datos “test” y se rescata la tabla que entrega los atributos y las probabilidades (fig 4) para ver la probabilidad de ser o no categorizada (porcentajes negativos y positivos).

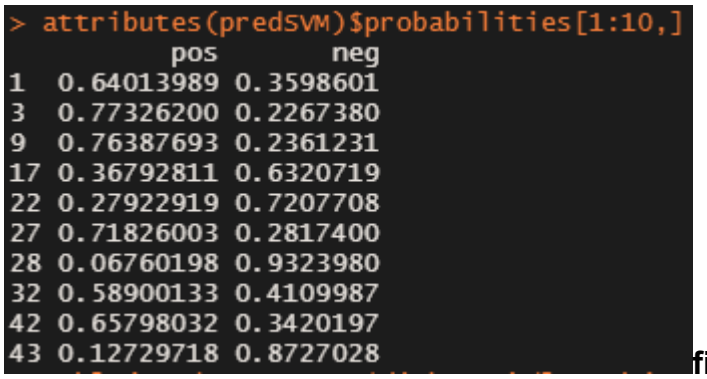

fig 4.

Luego se determina el porcentaje correspondiente a la validación del modelo (77% de validez) para después armar la tabla de confusión y luego confirmarla con el comando “confusionmatrix”.

 # SVM

model.svm = svm(diabetes ~ ., train1, kernel='linear',decision.values=T,probability = T)

predSVM=predict(model.svm,test1,decision.values=T, probability = T)

attributes(predSVM)$decision.values[1:10,]

attributes(predSVM)$probabilities[1:10,]

table(predSVM==test1$diabetes)/length(test1$diabetes)

table(predichas=predSVM,reales=test1$diabetes)

predSVM=as.factor(labels)

confusionMatrix(predSVM, test1$diabetes)

- Especificidad: 56%
- Sensibilidad: 88%

Finalmente se define el mejor costo para el modelo SVM con kernel lineal, el cual fue el costo C = 0,25 y se iteró 10 veces para encontrar el mejor costo, posterior se procede a generar la tabla donde se observa que tiende a ser efectivo un 77% de

las veces y un 23% a equivocarse aproximadamente, luego se confirma mediante el comando “confusionmatrix” con tal de obtener los valores de sensibilidad y especificidad.

- Sensibilidad: 87%
- Especificidad: 54%

Como última instancia se define el área bajo la curva mediante las curvas ROC en el gráfico (fig 5) para determinar la cantidad de datos almacenados, el cual fue de un 84%, por lo que el modelo es efectivo.

```
# Usamos el "costo" C para regular los errores
tune.out1.1=tune(svm,diabetes~.,data=test1,kernel="linear",
                 ranges=list(cost=c(0.01,0.25,0.1,1.0,10)))
summary(tune.out1.1)
bestmod1.1=tune.out1.1$best.model
summary(bestmod1.1)
ypred1.1=predict(bestmod1.1,train1)
model.svm.cost1.1 = svm(diabetes ~ .,test1,kernel='linear',cost=0.25,decision.values=T,probability
= T)
pred1.1=predict(model.svm.cost1.1,train1,decision.values=T,probability = T)
table(train1$diabetes==labels1)/length(train1$diabetes)
table(predichos=pred1.1,reales=train1$diabetes)

confusionMatrix(pred1.1, train1$diabetes)
val1.1=attributes(pred1.1)$decision.values
q_SVM1.1 <- roc(predictor = as.numeric(val1.1), response = train1$diabetes,
                levels = levels(train1$diabetes))
plot(q_SVM1.1, col = "#9E0142")
auc(q_SVM1.1)
```

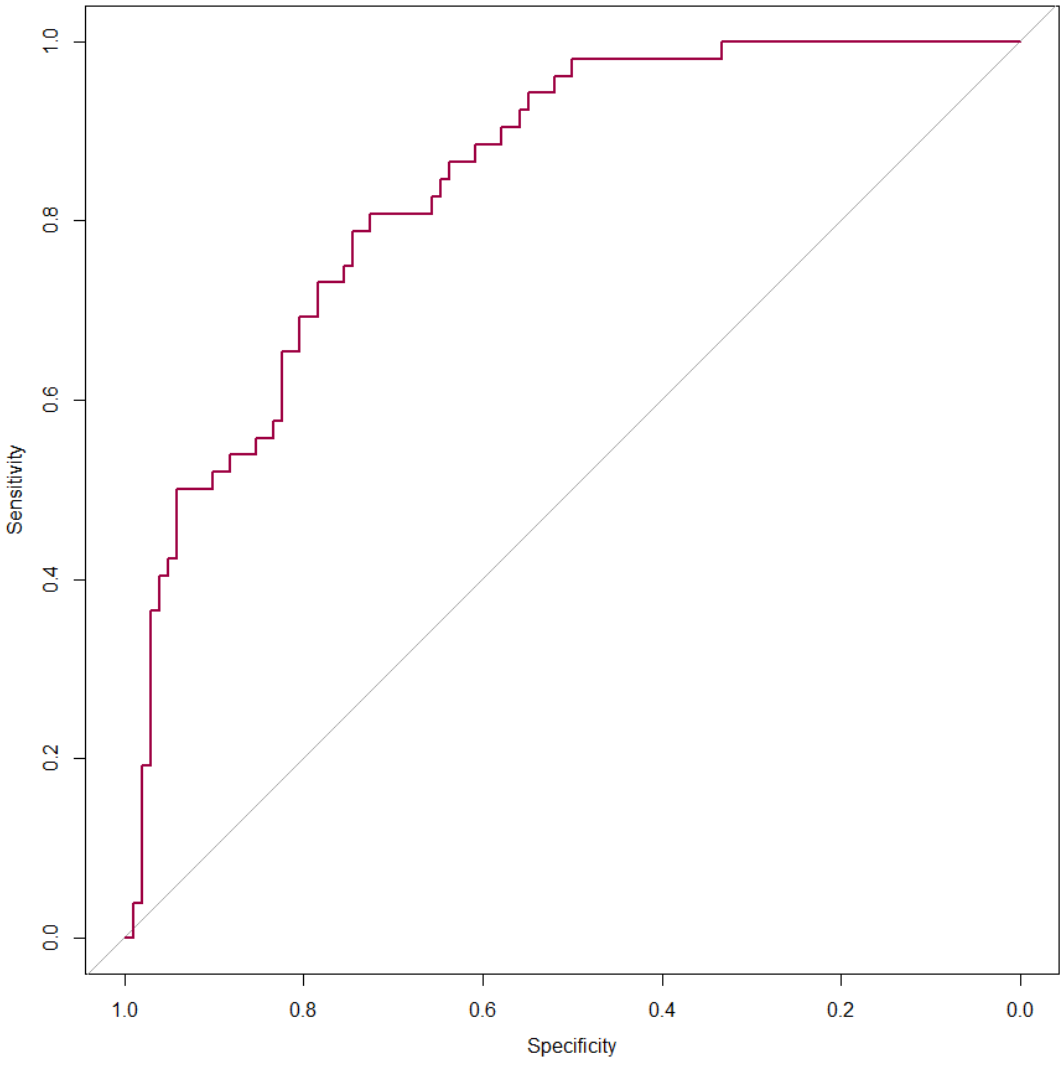

fig 5

Problema 3

Para ejercicio de este problema, se tuvo que instalar el paquete "ISLR" para obtener los datos a trabajar y se utilizaron las librerías "caret", "klaR" y "e1071".

Para comenzar, se tuvo que ubicar todos los valores “NA” de la columna "Salary" con el fin de eliminarlos y posteriormente se buscó determinar los cuantiles 25% y 75% de esta columna para asociar a cada jugador en un rango de sueldo "bajo", "medio", "alto", en una columna nueva llamada "label".

Una vez ubicados en el rango de sueldo en la columna "label", se eliminó la columna de "Salary" de la base de datos para generar una partición para los datos training y testing en una proporción 70/30, para esto se utilizó “set.seed(111)” con un 70% de los datos training y 30% de los datos testing.

Se construyó un modelo de clasificación “LDA” utilizando los datos training y posteriormente analizamos la calidad del modelo con los datos testing. Luego se procedió a realizar la tabla de confusión para esta base de datos obteniendo los siguientes valores:

	reales		
predichos	bajo	medio	alto
bajo	19	5	1
medio	4	28	6
alto	1	4	11

Obteniendo así una sensibilidad del 79% para la clase “bajo”, 76% para la clase “medio” y 61% para la clase “alto” aproximadamente.

Además, cabe mencionar que la especificidad de esta tabla de confusión fue de un 89% para la clase “bajo”, 76% para la clase “medio” y 92% para la clase “alto” aproximadamente.

Para finalizar, se descargó la base de datos "jugadores.csv" de tres observaciones de jugadores "Pedro”, “Juan" y "Diego" para implementar el clasificador “LDA” construido anteriormente y determinar a qué rango de "label" pertenece cada uno de ellos. Tras esto, se obtuvo la siguiente tabla:

	reales		
predichos	Diego	Juan	Pedro
bajo	0	0	0
medio	0	1	1
alto	1	0	0

Como se puede observar, el clasificador “LDA” optó por categorizar a Juan y Pedro con un sueldo “medio” de acuerdo a sus datos de sus variables y a Diego como que tenía un sueldo “alto”.

Finalmente, se definen las probabilidades “a posteriori” solicitadas, obteniendo así la siguiente tabla:

	bajo	medio	alto
1	0.063870303	0.91704867	0.01908103
2	0.118865221	0.87581105	0.00532373
3	0.005730547	0.07946056	0.91480889

De acuerdo a esta tabla, podemos afirmar que, precisamente, Juan y Pedro tienen la mayor probabilidad de ser categorizados como que tienen un sueldo “medio”, con un 92% y 87.5% aproximadamente. Por otro lado, podemos afirmar que la

probabilidad de que Diego sea categorizado con un sueldo “alto” es de un 79.5% aproximadamente, lo cual reafirma la validez de la tabla realizada de acuerdo al clasificador escogido.

```
# PROBLEMA 3: PREDICCIÓN DE SALARIOS DE JUGADORES DE BASEBALL.
################################################################################
#install.packages("caret")
#install.packages("KlaR")
#install.packages("ISLR")
#install.packages("e1071")
library(e1071)
library(caret)
library(klaR)
library(ISLR)
help(Hitters) # le permitirá saber qué significan las variables.
dim(Hitters)
sum(is.na(Hitters$Salary))
Hitters <- na.omit(Hitters) # eliminamos los NA
dim(Hitters)
sum(is.na(Hitters))
```

```
#Pregunta 1

quantile(Hitters$Salary) # de aqui obtenemos los cuantiles 25% y 75%

a <- 67
b <- 190
c <- 750
d <- 2500

particion=cut(Hitters$Salary, breaks = c(a,b,c,d)) #Aquí hacemos la partición
```

```
table(particion)
intervalo1='(67,190]'
intervalo2='(190,750]'
intervalo3='(750,2.5e+03]'
unique(particion)
label=factor(particion,levels=c(intervalo1, intervalo2, intervalo3),
        labels=c("bajo","medio","alto"))
unique(label)
Hitters$label=label #Crea una nueva columna en el dataset
View(Hitters)
#Eliminamos la variable Salary para trabajar solo con label
datos=Hitters[,-19]
View(datos)
# Pregunta 2

# (A) Divida los datos en training y testing en la razón 70/30% (coloque set.seed(111) antes)
# Construya un modelo de clasificación LDA o Naive Bayes que clasifique
# los salarios en bajos, medios y altos de los jugadores de baseball usando
# los datos training y haga predicciones empleando los datos testing.
set.seed(111)
n=nrow(datos)
a3=sample(c(1:n), floor(n*0.7),replace=F)
train3=datos[a3,]
test3=datos[-a3,]



modelo3=lda(label~.,data=train3)
modelo3
pred3=predict(modelo3,test3)$class
tabla3=table(predichos=pred3 , reales=test3$label)
tabla3
```

```
confusionMatrix(tabla3)


# C

dat <- read.csv('C:/Users/pc/Desktop/bases de datos csv y excel/jugadores.csv',sep=',',dec = ".",header = TRUE)

pred3.1 <- predict(modelo3,dat)$class

tabla3.1 <- table(predichos = pred3.1, reales = dat$X)

tabla3.1

# D

p=predict(modelo3, dat)

p

names(p)

p.class=p$class

p.class

tabla <- table(p.class,dat$X)

tabla

p$posterior
```