Support Vector Machine con e1071 (kernels lineal, radial y polinomial)

Adolfo Sánchez Burón

- Algoritmos empleados: Support Vector Machine (SVM)
- Características del caso
- Proceso
- 1. Entorno
  - 1.1. Instalar librerías
  - 1.2. Importar datos
- 2. Análisis descriptivo
  - 2.1. Análisis inicial
  - 2.2. Tipología de datos
  - 2.3. Análisis descriptivo (gráficos)
- 3. Modelización
  - 3.1. Preparar funciones
  - 3.2. Particiones de training (70%) y test (30%)
- 4. Modelización con Support Vector Machine con e1071
  - 4.1. Linear kernel function
  - 4.2. Radial Basis Function (RBF) Kernel ("Gaussian")
  - 4.3. Modelo SVM Kernel polinomial
- 5. Comparación de los tres modelos

Algoritmos empleados: Support Vector Machine (SVM)

Para un breve resumen del algoritmo de Support Vector Machine (SVM), mirar el post (https://www.ml2projects.com/post/svm_kernlab)

Características del caso

El caso empleado en este análisis es el 'German Credit Data', que puede descargarse el dataset original desde UCI (https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data)). Este dataset ha sido previamente trabajado en cuanto a:

- análisis descriptivo
- limpieza de anomalías, missing y outliers
- peso predictivo de las variables mediante random forest
- discretización de las variables continuas para facilitar la interpretación posterior

Por lo que finalmente se emplea en este caso un dataset preparado para iniciar el análisis, que puede descargarse de GitHub (https://github.com/AdSan-R/MachineLearning_R/tree/main/dataset).

El objetivo del caso es predecir la probabilidad de que un determinado cliente puede incluir un crédito bancario. La explicación de esta conducta estará basada en toda una serie de variables predictoras que se explicarán posteriormente.

Proceso

1. Entorno

El primer punto tratará sobre la preparación del entorno, donde se mostrará la descarga de las librerías empleadas y la importación de datos.

2. Análisis descriptivo

Se mostrarán y explicarán las funciones empleadas en este paso, dividiéndolas en tres grupos: Análisis inicial, Tipología de datos y Análisis descriptivo (gráficos).

3. Preparación de la modelización

Particiones del dataset en dos grupos: training (70%) y test (30%)

4. Modelización

Por motivos didácticos, se dividirá la modelización de los dos algoritmos en una sucesión de pasos.

---

1. Entorno

1.1. Instalar librerías

```
library(dplyr)
library(knitr)       # For Dynamic Report Generation in R
library(ROCR)        # Model Performance and ROC curve
library(caret)       # Classification and Regression Training -  for any machine learn
ing algorithms
library(e1071)       # Support Vector Machine
library(DataExplorer) #para realizar el análisis descriptivo con gráficos
```

1.2. Importar datos

Como el dataset ha sido peviamente trabajado para poder modelizar directamente, si deseas seguir este tutorial, lo puedes descargar de GitHub (https://github.com/AdSan-R).

```
df <- read.csv("CreditBank")
```

---

2. Análisis descriptivo

2.1. Análisis inicial

```r
head(df) #ver la estructura de los primeros 6 casos
```

```
##   X chk_ac_status_1 credit_history_3 duration_month_2 savings_ac_bond_6
## 1 1             A11           04.A34            00-06               A65
## 2 2             A12       03.A32.A33              42+               A61
## 3 3             A14           04.A34            06-12               A61
## 4 4             A11       03.A32.A33            36-42               A61
## 5 5             A11       03.A32.A33            12-24               A61
## 6 6             A14       03.A32.A33            30-36               A65
##   purpose_4 property_type_12 age_in_yrs_13 credit_amount_5 p_employment_since_7
## 1       A43             A121           60+          0-1400                  A75
## 2       A43             A121          0-25           5500+                  A73
## 3       A46             A121         45-50       1400-2500                  A74
## 4       A42             A122         40-45           5500+                  A74
## 5       A40             A124         50-60       4500-5500                  A73
## 6       A46             A124         30-35           5500+                  A73
##   housing_type_15 other_instalment_type_14 personal_status_9 foreign_worker_20
## 1            A152                     A143               A93              A201
## 2            A152                     A143               A92              A201
## 3            A152                     A143               A93              A201
## 4            A153                     A143               A93              A201
## 5            A153                     A143               A93              A201
## 6            A153                     A143               A93              A201
##   other_debtors_or_grantors_10 instalment_pct_8 good_bad_21
## 1                         A101                4        Good
## 2                         A101                2         Bad
## 3                         A101                2        Good
## 4                         A103                2        Good
## 5                         A101                3         Bad
## 6                         A101                2        Good
```

2.2. Tipología de datos

```r
str(df) #mostrar la estructura del dataset y los tipos de variables
```

```
## 'data.frame':    1000 obs. of  17 variables:
##  $ X                           : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ chk_ac_status_1             : chr  "A11" "A12" "A14" "A11" ...
##  $ credit_history_3            : chr  "04.A34" "03.A32.A33" "04.A34" "03.A32.A33"
...
##  $ duration_month_2            : chr  "00-06" "42+" "06-12" "36-42" ...
##  $ savings_ac_bond_6           : chr  "A65" "A61" "A61" "A61" ...
##  $ purpose_4                   : chr  "A43" "A43" "A46" "A42" ...
##  $ property_type_12            : chr  "A121" "A121" "A121" "A122" ...
##  $ age_in_yrs_13               : chr  "60+" "0-25" "45-50" "40-45" ...
##  $ credit_amount_5             : chr  "0-1400" "5500+" "1400-2500" "5500+" ...
##  $ p_employment_since_7        : chr  "A75" "A73" "A74" "A74" ...
##  $ housing_type_15             : chr  "A152" "A152" "A152" "A153" ...
##  $ other_instalment_type_14    : chr  "A143" "A143" "A143" "A143" ...
##  $ personal_status_9           : chr  "A93" "A92" "A93" "A93" ...
##  $ foreign_worker_20           : chr  "A201" "A201" "A201" "A201" ...
##  $ other_debtors_or_grantors_10: chr  "A101" "A101" "A101" "A103" ...
##  $ instalment_pct_8            : int  4 2 2 2 3 2 3 2 2 4 ...
##  $ good_bad_21                 : chr  "Good" "Bad" "Good" "Good" ...
```

Puede observarse que todas son "chr", esto es, "character", por tanto, vamos a pasarlas a Factor. Además, instalment_pct_8 aparece como "entero" cuando es factor. También la transformamos.

```
df <- mutate_if(df, is.character, as.factor) #identifica todas las character y las pas
a a factores
#Sacamos la esructura

df$instalment_pct_8 <- as.factor(df$instalment_pct_8 )

str(df)
```

```
## 'data.frame':    1000 obs. of  17 variables:
##  $ X                            : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ chk_ac_status_1              : Factor w/ 4 levels "A11","A12","A13",..: 1 2 4 1 1
4 4 2 4 2 ...
##  $ credit_history_3             : Factor w/ 4 levels "01.A30","02.A31",..: 4 3 4 3 3
3 3 3 3 4 ...
##  $ duration_month_2             : Factor w/ 7 levels "00-06","06-12",..: 1 7 2 6 3 5
3 5 2 4 ...
##  $ savings_ac_bond_6            : Factor w/ 5 levels "A61","A62","A63",..: 5 1 1 1 1
5 3 1 4 1 ...
##  $ purpose_4                    : Factor w/ 10 levels "A40","A41","A410",..: 5 5 8 4
1 8 4 2 5 1 ...
##  $ property_type_12             : Factor w/ 4 levels "A121","A122",..: 1 1 1 2 4 4 2
3 1 3 ...
##  $ age_in_yrs_13                : Factor w/ 8 levels "0-25","25-30",..: 8 1 6 5 7 3
7 3 8 2 ...
##  $ credit_amount_5              : Factor w/ 6 levels "0-1400","1400-2500",..: 1 6 2
6 5 6 3 6 3 5 ...
##  $ p_employment_since_7         : Factor w/ 5 levels "A71","A72","A73",..: 5 3 4 4 3
3 5 3 4 1 ...
##  $ housing_type_15              : Factor w/ 3 levels "A151","A152",..: 2 2 2 3 3 3 2
1 2 2 ...
##  $ other_instalment_type_14     : Factor w/ 3 levels "A141","A142",..: 3 3 3 3 3 3 3
3 3 3 ...
##  $ personal_status_9            : Factor w/ 4 levels "A91","A92","A93",..: 3 2 3 3 3
3 3 3 1 4 ...
##  $ foreign_worker_20            : Factor w/ 2 levels "A201","A202": 1 1 1 1 1 1 1 1
1 1 ...
##  $ other_debtors_or_grantors_10: Factor w/ 3 levels "A101","A102",..: 1 1 1 3 1 1 1
1 1 1 ...
##  $ instalment_pct_8             : Factor w/ 4 levels "1","2","3","4": 4 2 2 2 3 2 3
2 2 4 ...
##  $ good_bad_21                  : Factor w/ 2 levels "Bad","Good": 2 1 2 2 1 2 2 2 2
1 ...
```

Ahora se puede observar que todas las variables son de tipo "Factor"

Para los siguientes análisis: 1º) Eliminamos a la variable X (número de cliente) del df. 2º) Renombraos la la variable good_bad_21 como "target"

```
#Eliminamos x
df <- select(df,-X)

#Creamos la variable "target"
df$target <- as.factor(df$good_bad_21)

#Eliminamos la variable "good_bad_21"
df <- select(df,-good_bad_21)

str(df)
```

```
## 'data.frame':    1000 obs. of  16 variables:
##  $ chk_ac_status_1            : Factor w/ 4 levels "A11","A12","A13",..: 1 2 4 1 1
4 4 2 4 2 ...
##  $ credit_history_3           : Factor w/ 4 levels "01.A30","02.A31",..: 4 3 4 3 3
3 3 3 3 4 ...
##  $ duration_month_2           : Factor w/ 7 levels "00-06","06-12",..: 1 7 2 6 3 5
3 5 2 4 ...
##  $ savings_ac_bond_6          : Factor w/ 5 levels "A61","A62","A63",..: 5 1 1 1 1
5 3 1 4 1 ...
##  $ purpose_4                  : Factor w/ 10 levels "A40","A41","A410",..: 5 5 8 4
1 8 4 2 5 1 ...
##  $ property_type_12           : Factor w/ 4 levels "A121","A122",..: 1 1 1 2 4 4 2
3 1 3 ...
##  $ age_in_yrs_13              : Factor w/ 8 levels "0-25","25-30",..: 8 1 6 5 7 3
7 3 8 2 ...
##  $ credit_amount_5            : Factor w/ 6 levels "0-1400","1400-2500",..: 1 6 2
6 5 6 3 6 3 5 ...
##  $ p_employment_since_7       : Factor w/ 5 levels "A71","A72","A73",..: 5 3 4 4 3
3 5 3 4 1 ...
##  $ housing_type_15            : Factor w/ 3 levels "A151","A152",..: 2 2 2 3 3 3 2
1 2 2 ...
##  $ other_instalment_type_14   : Factor w/ 3 levels "A141","A142",..: 3 3 3 3 3 3 3
3 3 3 ...
##  $ personal_status_9          : Factor w/ 4 levels "A91","A92","A93",..: 3 2 3 3 3
3 3 3 1 4 ...
##  $ foreign_worker_20          : Factor w/ 2 levels "A201","A202": 1 1 1 1 1 1 1 1
1 1 ...
##  $ other_debtors_or_grantors_10: Factor w/ 3 levels "A101","A102",..: 1 1 1 3 1 1 1
1 1 1 ...
##  $ instalment_pct_8           : Factor w/ 4 levels "1","2","3","4": 4 2 2 2 3 2 3
2 2 4 ...
##  $ target                     : Factor w/ 2 levels "Bad","Good": 2 1 2 2 1 2 2 2 2
1 ...
```

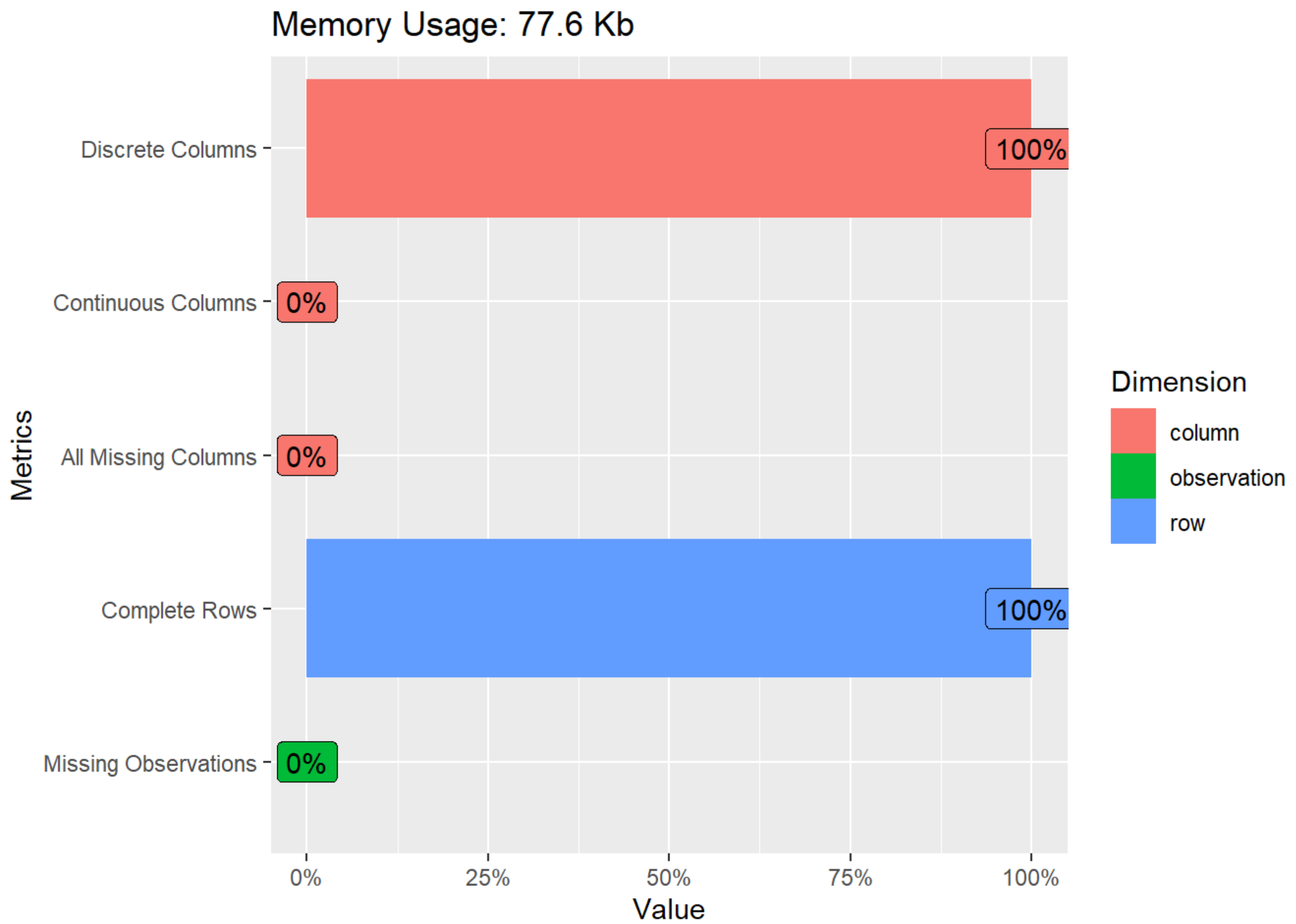
```
lapply(df,summary) #mostrar la distribución de frecuencias en cada categoría de todas
 las variables
```

```
## $chk_ac_status_1
## A11 A12 A13 A14
## 274 269  63 394
##
## $credit_history_3
##     01.A30     02.A31 03.A32.A33     04.A34
##         40         49        618        293
##
## $duration_month_2
## 00-06 06-12 12-24 24-30 30-36 36-42   42+
##    82   277   411    57    86    17    70
##
## $savings_ac_bond_6
## A61 A62 A63 A64 A65
## 603 103  63  48 183
##
## $purpose_4
##  A40  A41 A410  A42  A43  A44  A45  A46  A48  A49
##  234  103   12  181  280   12   22   50    9   97
##
## $property_type_12
## A121 A122 A123 A124
##  282  232  332  154
##
## $age_in_yrs_13
##  0-25 25-30 30-35 35-40 40-45 45-50 50-60   60+
##   190   221   177   138    88    73    68    45
##
## $credit_amount_5
##    0-1400 1400-2500 2500-3500 3500-4500 4500-5500     5500+
##       267       270       149        98        48       168
##
## $p_employment_since_7
## A71 A72 A73 A74 A75
##  62 172 339 174 253
##
## $housing_type_15
## A151 A152 A153
##  179  713  108
##
## $other_instalment_type_14
## A141 A142 A143
##  139   47  814
##
## $personal_status_9
## A91 A92 A93 A94
##  50 310 548  92
##
## $foreign_worker_20
## A201 A202
##  963   37
```

```
##
## $other_debtors_or_grantors_10
## A101 A102 A103
##  907   41   52
##
## $instalment_pct_8
##   1   2   3   4
## 136 231 157 476
##
## $target
##  Bad Good
##  300  700
```

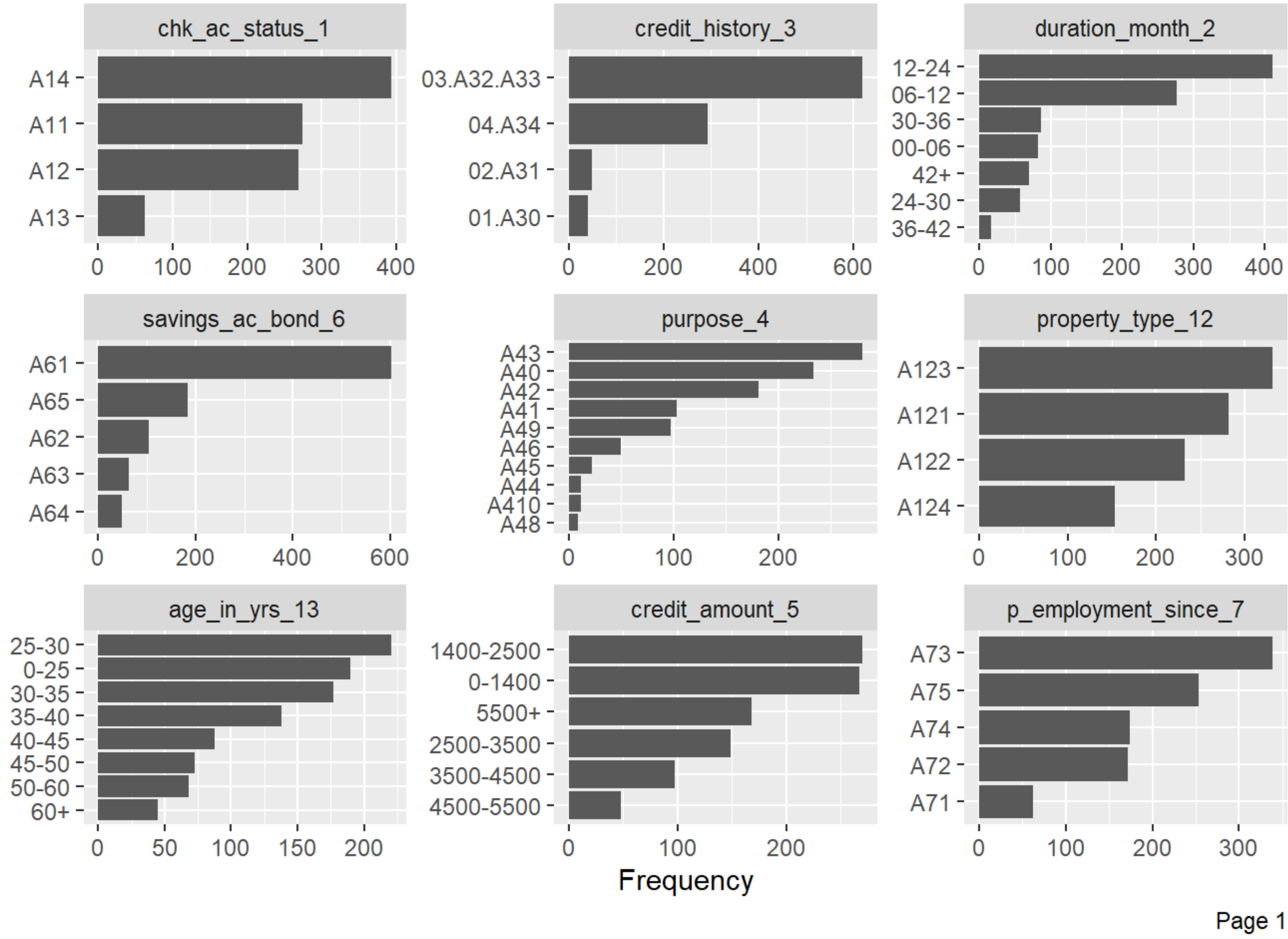
2.3. Análisis descriptivo (gráficos)

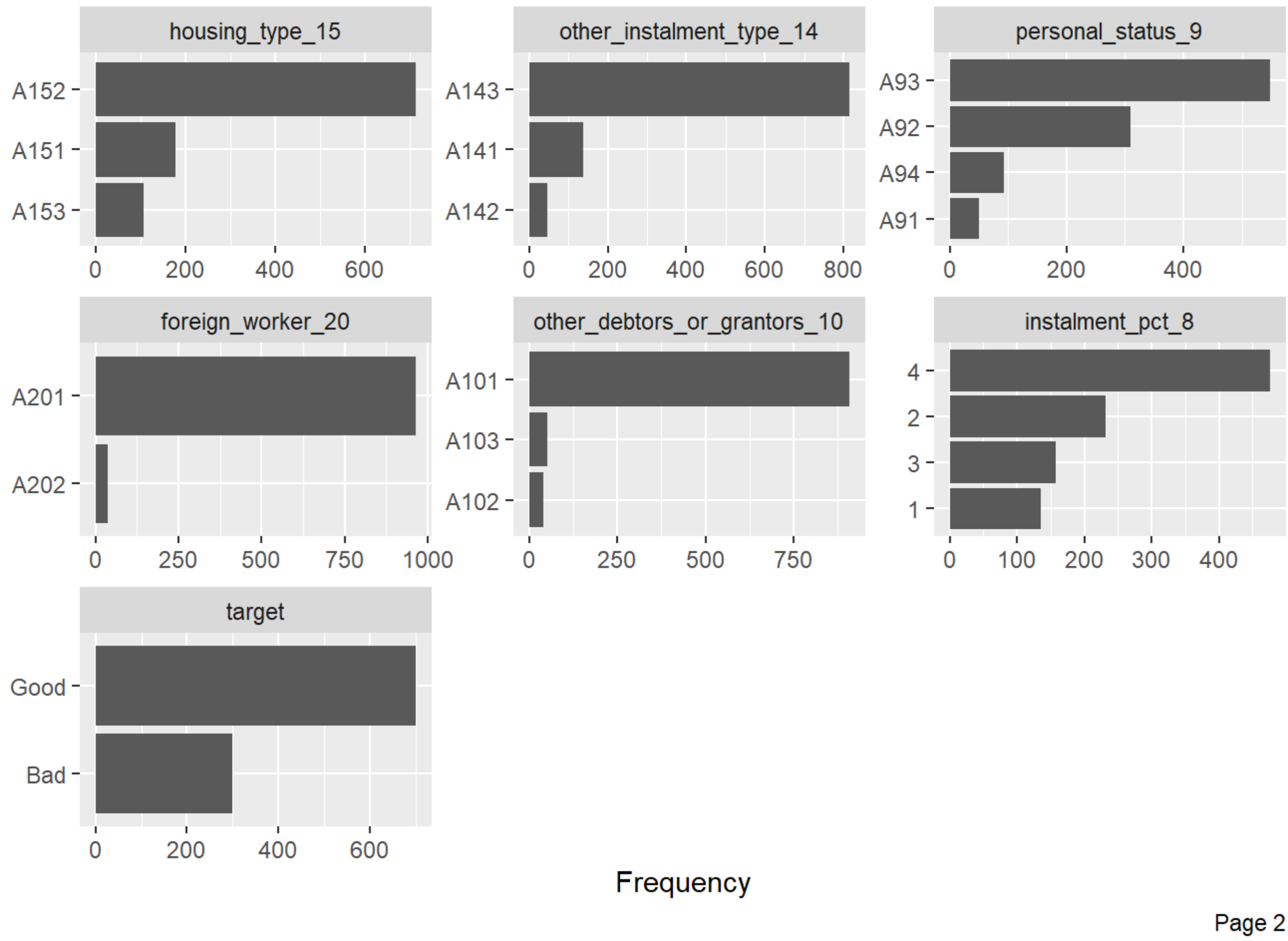
```
plot_intro(df) #gráfico para observar la distribución de variables y los casos missing
por columnas, observaciones y filas
```



Como se ha trabajado previamente, no existen casos missing, por lo que podemos seguir el análisis descriptivo.

```
plot_bar(df) #gráfico para observar la distribución de frecuencias en variables categó
ricas
```





3. Modelización

3.1. Preparar funciones

Tomadas del curso de Machine Learning Predictivo (https://www.datascience4business.com/08_mlc-salespage-b) de DS4B) :

- Matriz de confusión
- Métricas
- Umbrales

Función para la matriz de confusión

En esta función se prepara la matriz de confusión (ver en otro post), donde se observa qué casos coinciden entre la puntuación real (obtenida por cada sujeto) y la puntuación predicha ("scoring") por el modelo, estableciendo previmente un límite ("umbral") para ello.

```
confusion<-function(real,scoring,umbral){
  conf<-table(real,scoring>=umbral)
  if(ncol(conf)==2) return(conf) else return(NULL)
}
```

Funcion para métricas de los modelos

Los indicadores a observar serán:

- Acierto (accuracy) = (TRUE POSITIVE + TRUE NEGATIVE) / TODA LA POBLACIÓN
- Precisión = TRUE POSITIVE / (TRUE POSITIVE + FALSE POSITIVE)
- Cobertura (recall, sensitivity) = TRUE POSITIVE / (TRUE POSITIVE + FALSE NEGATIVE)
- F1 = 2* (precisión * cobertura) (precisión + cobertura)

```
metricas<-function(matriz_conf){
  acierto <- (matriz_conf[1,1] + matriz_conf[2,2]) / sum(matriz_conf) *100
  precision <- matriz_conf[2,2] / (matriz_conf[2,2] + matriz_conf[1,2]) *100
  cobertura <- matriz_conf[2,2] / (matriz_conf[2,2] + matriz_conf[2,1]) *100
  F1 <- 2*precision*cobertura/(precision+cobertura)
  salida<-c(acierto,precision,cobertura,F1)
  return(salida)
}
```

Función para probar distintos umbrales

Con esta función se analiza el efecto que tienen distintos umbrales sobre los indicadores de la matriz de confusión (precisión y cobertura). Lo que buscaremnos será aquél que maximice la relación entre cobertura y precisión (F1).

```
umbrales<-function(real,scoring){
  umbrales<-data.frame(umbral=rep(0,times=19),acierto=rep(0,times=19),precision=rep(0,
times=19),cobertura=rep(0,times=19),F1=rep(0,times=19))
  cont <- 1
  for (cada in seq(0.05,0.95,by = 0.05)){
    datos<-metricas(confusion(real,scoring,cada))
    registro<-c(cada,datos)
    umbrales[cont,]<-registro
    cont <- cont + 1
  }
  return(umbrales)
}
```

3.2. Particiones de training (70%) y test (30%)

Se segmenta la muestra en dos partes (train y test) empleando el programa Caret.

1. Training o entrenamiento (70% de la muestra): servirá para entrenar al modelo de clasificación.
2. Test (30%): servirá para validar el modelo. La característica fundamental es que esta muestra no debe haber tenido contacto previamente con el funcionamiento del modelo.

```
set.seed(100)  # Para reproducir los mismos resultados
partition <- createDataPartition(y = df$target, p = 0.7, list = FALSE)
train <- df[partition,]
test <- df[-partition,]
```

```
#Distribución de la variable TARGET
table(train$target)
```

```
## 
##  Bad Good 
##  210  490
```

```
table(test$target)
```

```
## 
##  Bad Good 
##   90  210
```

4. Modelización con Support Vector Machine con e1071

Emplearemos la librería e1071 para extraer SVM con kernel lineal, radial y polinómico. La función svm tiene una serie de hiperparámetros que deben especificarse en cada tipo de kernel.

- fórmula: especificando la variable dependiente y las predictoras.

- data: dataframe conteniendo los datos.
- type: en este proyecto C-classification (classification problem)
- kernel: tipo de límite de calsificación (en nuestro caso lineal, radial o polinómico)
- cost: parámetro necesario para todos los kernels, controla la severidad permitida de las violaciones de las n observaciones y, por tanto, el equilibrio bias-varianza (default: 1)
- gamma: parámetro necesario para todos los kernels, salvo el lineal (default: 1/(data dimension))
- coef: parámetro necesario para los kernels polinomial y sigmoidal (default: 0)
- degree: parámetro necesario para el kernel polinomial (default: 3)

4.1. Linear kernel function

Paso 1. Ajuste de hiperparámetros

Hallamos el valor de coste mediante tune.svm

```
set.seed(123)
tune_l = tune.svm(target~., data=train, kernel="linear",
                cost = c(0.001, 0.01, 0.1, 1, 5, 10, 50))
summary(tune_l)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.2442857
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-03 0.3000000 0.05753831
## 2 1e-02 0.3000000 0.05753831
## 3 1e-01 0.2542857 0.07184008
## 4 1e+00 0.2514286 0.06068393
## 5 5e+00 0.2457143 0.04655964
## 6 1e+01 0.2442857 0.05104353
## 7 5e+01 0.2471429 0.05041776
```

Observamos el mejor modelo

```
#Mejor modelo
bestmod <- tune_l$best.model
bestmod
```

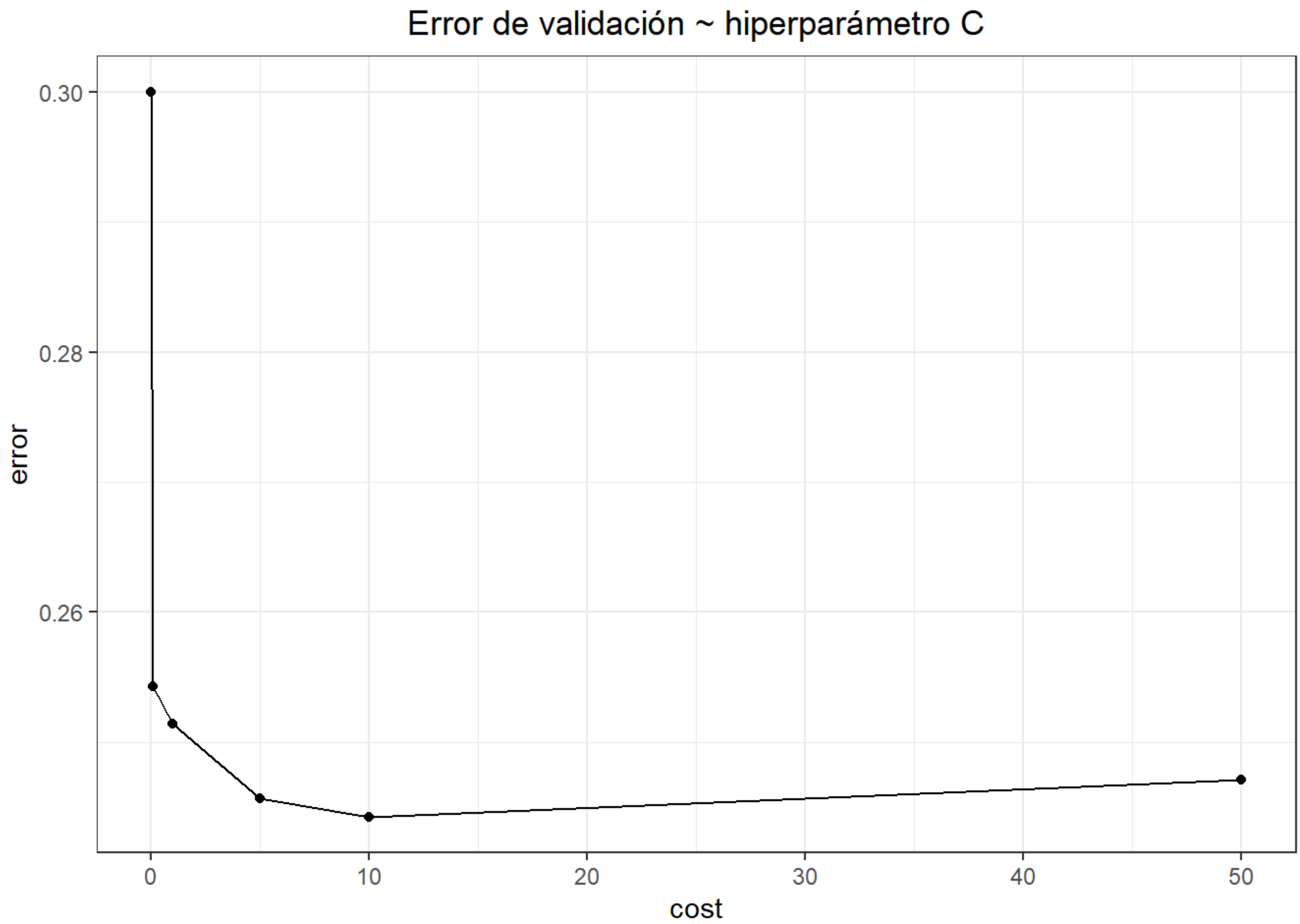
```
##
## Call:
## best.svm(x = target ~ ., data = train, cost = c(0.001, 0.01, 0.1,
##     1, 5, 10, 50), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##
## Number of Support Vectors:  350
```

```
# Mejor valor de coste
tune_l$best.parameters$cost
```

```
## [1] 10
```

El mejor valor para coste es 10. Lo introducimos en la función siguiente

```
# Graficando los costos de tuning
ggplot(data = tune_l$performances, aes(x = cost, y = error)) +
  geom_line() +
  geom_point() +
  labs(title = "Error de validación ~ hiperparámetro C") +
  theme_bw() +
  theme(plot.title = element_text(hjust = 0.5))
```



Paso 2. Entrenamiento del modelo

Incluimos como cst, cost = tune_l$best.parameters$cost.

```
svm_l<- svm(target ~ .,
            data = train,
            type = "C-classification",
            kernel = "linear",
            cost = tune_l$best.parameters$cost,
            scale = FALSE,
            probability = TRUE
            )

summary(svm_l)
```

```
##
## Call:
## svm(formula = target ~ ., data = train, type = "C-classification",
##     kernel = "linear", cost = tune_l$best.parameters$cost, probability = TRUE,
##     scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##
## Number of Support Vectors:  350
##
##  ( 182 168 )
##
##
## Number of Classes:  2
##
## Levels:
##  Bad Good
```

Paso 3. Predict y matriz de confusión

Sacamos las etiquetas de scores por el resultado del modelo svm lineal.

```
svm_score_l_Response <- predict(svm_l, test, type="response")

#Sacamos los 6 primeros valores
head(svm_score_l_Response)
```

```
##    4    7    9   13   14   23
## Good Good Good Good  Bad Good
## Levels: Bad Good
```

Observamos la matriz de confusión con sus métricas.

```
MC_svm_l <- confusionMatrix(svm_score_l_Response, test$target , positive = 'Good')
MC_svm_l
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction Bad Good
##       Bad   46   30
##       Good  44  180
##                                           
##                Accuracy : 0.7533          
##                  95% CI : (0.7005, 0.8011)
##     No Information Rate : 0.7             
##     P-Value [Acc > NIR] : 0.02388         
##                                           
##                   Kappa : 0.3854          
##                                           
##  Mcnemar's Test P-Value : 0.13073         
##                                           
##             Sensitivity : 0.8571          
##             Specificity : 0.5111          
##          Pos Pred Value : 0.8036          
##          Neg Pred Value : 0.6053          
##              Prevalence : 0.7000          
##          Detection Rate : 0.6000          
##    Detection Prevalence : 0.7467          
##       Balanced Accuracy : 0.6841          
##                                           
##        'Positive' Class : Good            
## 
```

Paso 4. Predict con probabilidades y umbrales

En este paso vamos a sacar, no las etiquetas, sino las probabilidades de que cada cliente devuelva o no un crédito.

1º) En este paso sacamos la probabilidad de cada cliente de devolver el crédito.

```
svm_score_l <- predict(svm_l, test, probability = TRUE)
svm_score_l_Prob <- attr(svm_score_l, "probabilities")[,1]
head(svm_score_l_Prob)
```

```
##         4         7         9        13        14        23 
## 0.9609979 0.8936337 0.9687302 0.6493583 0.4334925 0.7257843
```

2º) Ahora transformamos la probabilidad obtenida en una decisión binaria de si conceder el crédito (Sí lo va a devolver) o no (No lo va a devolver).

Con la función umbrales probamos diferentes cortes

```
umb_svm_l<-umbrales(test$target,svm_score_l_Prob)
umb_svm_l
```

```
##    umbral  acierto precision cobertura        F1
## 1    0.05  0.05000   0.05000  0.050000  0.050000
## 2    0.10  0.10000   0.10000  0.100000  0.100000
## 3    0.15 69.33333  69.79866 99.047619 81.889764
## 4    0.20 69.33333  69.93243 98.571429 81.818182
## 5    0.25 69.66667  70.16949 98.571429 81.980198
## 6    0.30 70.33333  70.93426 97.619048 82.164329
## 7    0.35 71.00000  71.73145 96.666667 82.352941
## 8    0.40 73.00000  73.80074 95.238095 83.160083
## 9    0.45 74.33333  75.28517 94.285714 83.720930
## 10   0.50 75.33333  77.20000 91.904762 83.913043
## 11   0.55 75.00000  78.48101 88.571429 83.221477
## 12   0.60 74.66667  81.30841 82.857143 82.075472
## 13   0.65 74.33333  83.75635 78.571429 81.081081
## 14   0.70 71.00000  85.14286 70.952381 77.402597
## 15   0.75 65.66667  85.43046 61.428571 71.468144
## 16   0.80 58.33333  86.32479 48.095238 61.773700
## 17   0.85 52.33333  93.50649 34.285714 50.174216
## 18   0.90 41.66667  94.87179 17.619048 29.718876
## 19   0.95 33.00000 100.00000  4.285714  8.219178
```

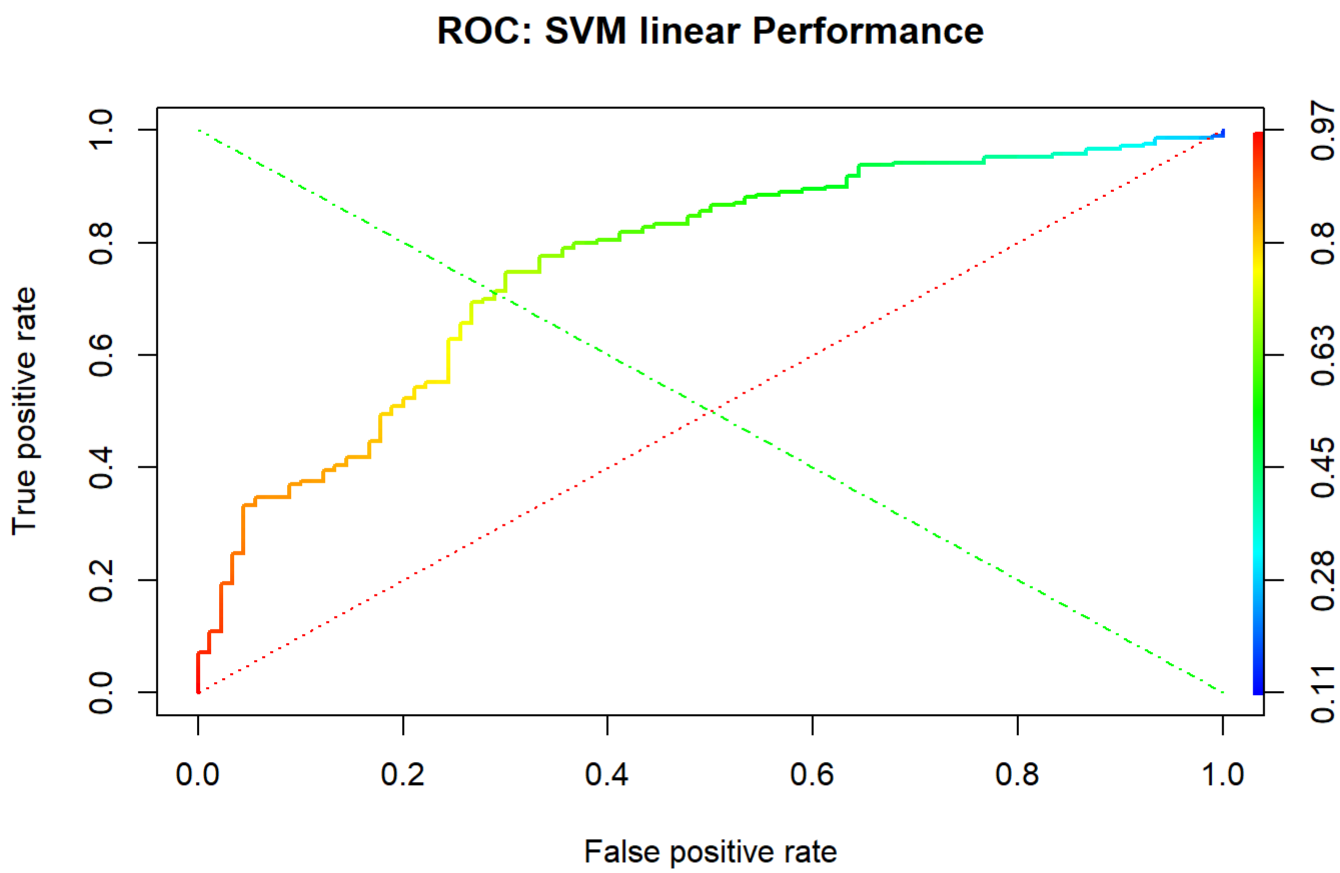
Seleccionamos el umbral que maximiza la F1 (cuando empieza a decaer)

```
umbfinal_svm_l<-umb_svm_l[which.max(umb_svm_l$F1),1]
umbfinal_svm_l
```

```
## [1] 0.5
```

Paso 5. Curva ROC

```
pred_svm_l <- prediction(svm_score_l_Prob, test$target)
perf_svm_l <- performance(pred_svm_l,"tpr","fpr")
#library(ROCR)
plot(perf_svm_l, lwd=2, colorize=TRUE, main="ROC: SVM linear Performance")
lines(x=c(0, 1), y=c(0, 1), col="red", lwd=1, lty=3);
lines(x=c(1, 0), y=c(0, 1), col="green", lwd=1, lty=4)
```



Paso 6. Métricas definitivas

```
#Matriz de confusión con umbral final
score <- ifelse(svm_score_l_Prob > umbfinal_svm_l, "Good", "Bad")
MC <- table(test$target, score)
Acc_svm_l <- round((MC[1,1] + MC[2,2]) / sum(MC) *100, 2)
Sen_svm_l <- round(MC[2,2] / (MC[2,2] + MC[1,2]) *100, 2)
Pr_svm_l <- round(MC[2,2] / (MC[2,2] + MC[2,1]) *100, 2)
F1_svm_l <- round(2*Pr_svm_l*Sen_svm_l/(Pr_svm_l+Sen_svm_l), 2)

#AUC
AUROC_svm_l <- round(performance(pred_svm_l, measure = "auc")@y.values[[1]]*100, 2)

#Métricas finales del modelo
cat("Acc_svm_l: ", Acc_svm_l,"\tSen_svm_l: ", Sen_svm_l, "\tPr_svm_l:", Pr_svm_l, "\tF
1_svm_l:", F1_svm_l, "\tAUROC_svm_l: ", AUROC_svm_l)
```

```
## Acc_svm_l:  75.33    Sen_svm_l:  77.2    Pr_svm_l: 91.9  F1_svm_l: 83.91     AUROC_
svm_l:  76.08
```

Observamos que la AUC tiene un valor de 76.08. Moderadamente aceptable.

4.2. Radial Basis Function (RBF) Kernel ("Gaussian")

Paso 1. Ajuste de hiperparámetros

En el RBF Kernel buscamos ajustar los arámetros de coste y gamma.

```
set.seed(123)
tune_r = tune.svm(target~., data=train, kernel="radial",
                cost = c(0.1,1,10,100,1000),
                gamma = c(0.5,1,2,3,4))
summary(tune_r)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  gamma cost
##    0.5   10
##
## - best performance: 0.2785714
##
## - Detailed performance results:
##    gamma  cost     error dispersion
## 1    0.5 1e-01 0.3000000 0.05753831
## 2    1.0 1e-01 0.3000000 0.05753831
## 3    2.0 1e-01 0.3000000 0.05753831
## 4    3.0 1e-01 0.3000000 0.05753831
## 5    4.0 1e-01 0.3000000 0.05753831
## 6    0.5 1e+00 0.2971429 0.06164782
## 7    1.0 1e+00 0.3000000 0.05753831
## 8    2.0 1e+00 0.3000000 0.05753831
## 9    3.0 1e+00 0.3000000 0.05753831
## 10   4.0 1e+00 0.3000000 0.05753831
## 11   0.5 1e+01 0.2785714 0.06908866
## 12   1.0 1e+01 0.3000000 0.05753831
## 13   2.0 1e+01 0.3000000 0.05753831
## 14   3.0 1e+01 0.3000000 0.05753831
## 15   4.0 1e+01 0.3000000 0.05753831
## 16   0.5 1e+02 0.2785714 0.06908866
## 17   1.0 1e+02 0.3000000 0.05753831
## 18   2.0 1e+02 0.3000000 0.05753831
## 19   3.0 1e+02 0.3000000 0.05753831
## 20   4.0 1e+02 0.3000000 0.05753831
## 21   0.5 1e+03 0.2785714 0.06908866
## 22   1.0 1e+03 0.3000000 0.05753831
## 23   2.0 1e+03 0.3000000 0.05753831
## 24   3.0 1e+03 0.3000000 0.05753831
## 25   4.0 1e+03 0.3000000 0.05753831
```

Observamos el mejor modelo y los valores de coste y gamma.

```
tune_r$best.model
```

```
##
## Call:
## best.svm(x = target ~ ., data = train, gamma = c(0.5, 1, 2, 3, 4),
##     cost = c(0.1, 1, 10, 100, 1000), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10
##
## Number of Support Vectors:  696
```

```
tune_r$best.parameters$cost
```

```
## [1] 10
```

```
tune_r$best.parameters$gamma
```

```
## [1] 0.5
```

Paso 2. Entrenamiento del modelo

Pasamos a entrenar el modelo con los valores obtenidos en coste(tune_r$best. parameters$cost) y gamma (tune_r$best. parameters$gamma).

```
svm_r <- svm(target ~ ., data = train,
             type = "C-classification",
             kernel = "radial",
             cost = tune_r$best.parameters$cost,
             gamma = tune_r$best.parameters$gamma,
             probability = TRUE)

summary(svm_r)
```

```
##
## Call:
## svm(formula = target ~ ., data = train, type = "C-classification",
##     kernel = "radial", cost = tune_r$best.parameters$cost, gamma = tune_r$best.para
meters$gamma,
##     probability = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10
##
## Number of Support Vectors:  696
##
##  ( 486 210 )
##
##
## Number of Classes:  2
##
## Levels:
##  Bad Good
```

Paso 3. Predict y matriz de confusión

Hacemos el predict y obtenemos las métricas de la matriz de confusión.

```
svm_score_r_Response <- predict(svm_r, test, type="response")
head(svm_score_r_Response)
```

```
##    4    7    9   13   14   23
## Good Good Good Good Good Good
## Levels: Bad Good
```

```
MC_svm_r <- confusionMatrix(svm_score_r_Response, test$target , positive = 'Good')
MC_svm_r
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Bad Good
##       Bad    8    5
##       Good  82  205
##
##                Accuracy : 0.71
##                  95% CI : (0.6551, 0.7607)
##     No Information Rate : 0.7
##     P-Value [Acc > NIR] : 0.3793
##
##                   Kappa : 0.0861
##
##  Mcnemar's Test P-Value : 3.698e-16
##
##             Sensitivity : 0.97619
##             Specificity : 0.08889
##          Pos Pred Value : 0.71429
##          Neg Pred Value : 0.61538
##              Prevalence : 0.70000
##          Detection Rate : 0.68333
##    Detection Prevalence : 0.95667
##       Balanced Accuracy : 0.53254
##
##        'Positive' Class : Good
##
```

Paso 4. Predict con probabilidades y umbrales

1º) En este paso sacamos la probabilidad de cada cliente de devolver el crédito.

```
svm_score_r <- predict(svm_r, test, probability = TRUE)
svm_score_r_Prob <- attr(svm_score_r, "probabilities")[,1]
head(svm_score_r_Prob)
```

```
##         4         7         9        13        14        23
## 0.6868303 0.8329394 0.8423351 0.5945678 0.7621789 0.8225202
```

2º) Ahora transformamos la probabilidad obtenida en una decisión binaria de si conceder el crédito (Sí lo va a devolver) o no (No lo va a devolver).

Con la función umbrales probamos diferentes cortes.

```
umb_svm_r<-umbrales(test$target,svm_score_r_Prob)
umb_svm_r
```

```
##    umbral  acierto precision cobertura       F1
## 1    0.05  0.05000   0.05000  0.050000  0.05000
## 2    0.10 71.00000  70.84746 99.523810 82.77228
## 3    0.15 71.33333  71.08844 99.523810 82.93651
## 4    0.20 71.33333  71.08844 99.523810 82.93651
## 5    0.25 71.33333  71.37931 98.571429 82.80000
## 6    0.30 71.00000  71.42857 97.619048 82.49497
## 7    0.35 70.33333  71.22807 96.666667 82.02020
## 8    0.40 70.66667  71.63121 96.190476 82.11382
## 9    0.45 70.33333  72.00000 94.285714 81.64948
## 10   0.50 70.66667  72.93233 92.380952 81.51261
## 11   0.55 70.33333  73.54086 90.000000 80.94218
## 12   0.60 69.00000  74.07407 85.714286 79.47020
## 13   0.65 69.00000  76.47059 80.476190 78.42227
## 14   0.70 67.66667  80.87432 70.476190 75.31807
## 15   0.75 62.33333  83.44828 57.619048 68.16901
## 16   0.80 57.33333  85.96491 46.666667 60.49383
## 17   0.85 47.66667  86.30137 30.000000 44.52297
## 18   0.90 42.00000  92.85714 18.571429 30.95238
## 19   0.95 34.33333 100.00000  6.190476 11.65919
```

Seleccionamos el umbral que maximiza la F1 (cuando empieza a decaer)

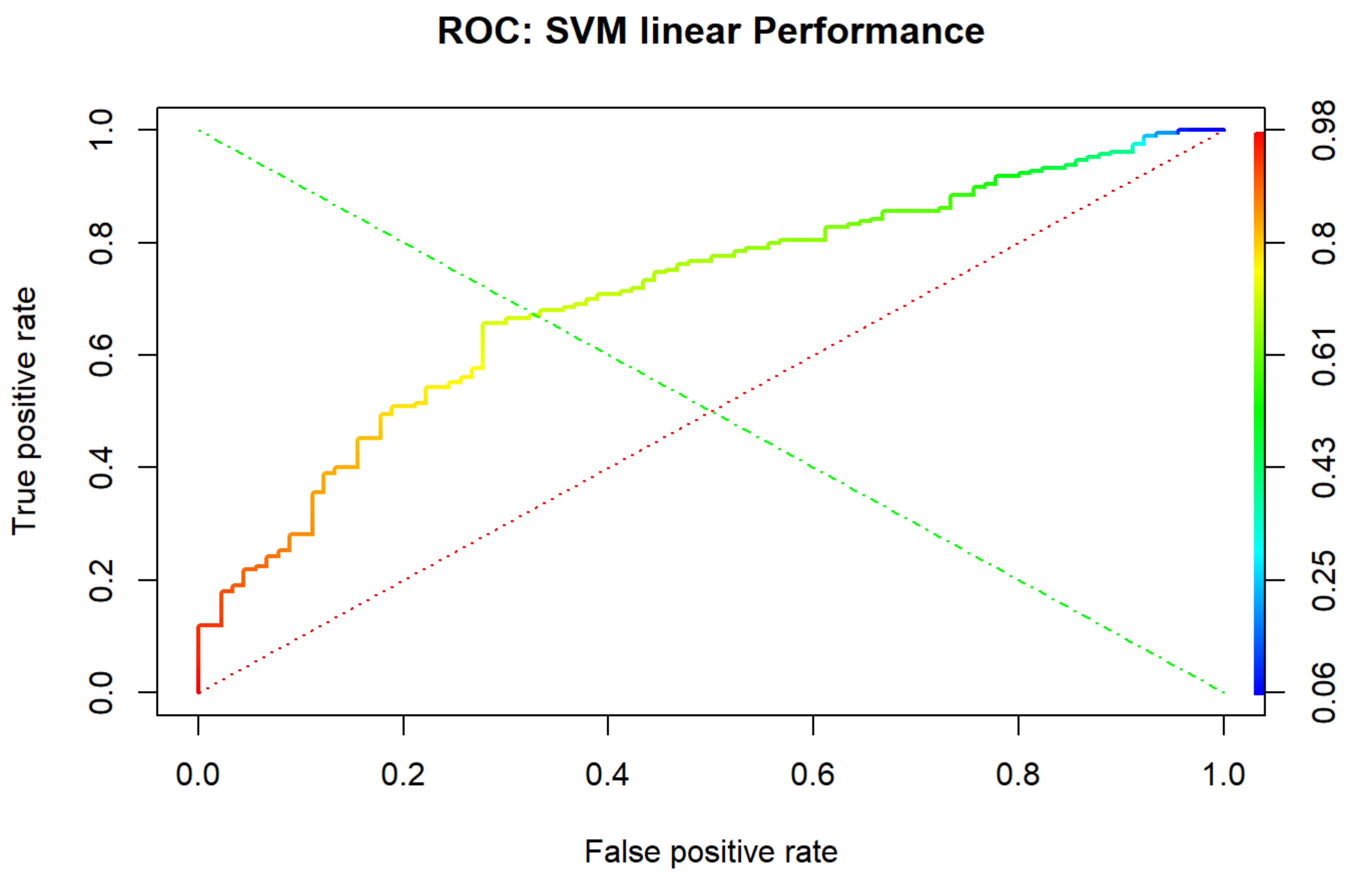
```
umbfinal_svm_r<-umb_svm_r[which.max(umb_svm_r$F1),1]
umbfinal_svm_r
```

```
## [1] 0.15
```

Vemos que el umbral de corte que maximiza la F1 es 0.15.

Paso 5. Curva ROC

```
pred_svm_r <- prediction(svm_score_r_Prob, test$target)
perf_svm_r <- performance(pred_svm_r,"tpr","fpr")
#library(ROCR)
plot(perf_svm_r, lwd=2, colorize=TRUE, main="ROC: SVM linear Performance")
lines(x=c(0, 1), y=c(0, 1), col="red", lwd=1, lty=3);
lines(x=c(1, 0), y=c(0, 1), col="green", lwd=1, lty=4)
```



Paso 6. Métricas definitivas

```
#Matriz de confusión con umbral final
score <- ifelse(svm_score_r_Prob > umbfinal_svm_r, "Good", "Bad")
MC <- table(test$target, score)
Acc_svm_r <- round((MC[1,1] + MC[2,2]) / sum(MC) *100, 2)
Sen_svm_r <- round(MC[2,2] / (MC[2,2] + MC[1,2]) *100, 2)
Pr_svm_r <- round(MC[2,2] / (MC[2,2] + MC[2,1]) *100, 2)
F1_svm_r <- round(2*Pr_svm_r*Sen_svm_r/(Pr_svm_r+Sen_svm_r), 2)

#AUC
AUROC_svm_r <- round(performance(pred_svm_r, measure = "auc")@y.values[[1]]*100, 2)

#Métricas finales del modelo
cat("Acc_svm_r: ", Acc_svm_r,"\tSen_svm_r: ", Sen_svm_r, "\tPr_svm_r:", Pr_svm_r, "\tF
1_svm_r:", F1_svm_r, "\tAUROC_svm_r: ", AUROC_svm_r)
```

```
## Acc_svm_r:  71.33    Sen_svm_r:  71.09    Pr_svm_r: 99.52     F1_svm_r: 82.94     AU
ROC_svm_r:  70.63
```

Se obtiene un modelo con una AUC = 70.63. El modelo es moderadamente aceptable.

4.3. Modelo SVM Kernel polinomial

Paso 1. Ajuste de hiperparámetros

En el kernel polinomial vamos a ajustar los parámetros de coste, gamma, degree y coef0.

```
set.seed(123)
tune_p = tune.svm(target~., data=train, kernel="polynomial",
                  cost = c(0.1,1,10,100,1000), gamma = c(0.1,1,10),
                  degree=c(2,3,4,5), coef0=c(0.1,0.5,1,2,3))
summary(tune_p)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  degree gamma coef0 cost
##       2   0.1     2    1
##
## - best performance: 0.2385714
##
## - Detailed performance results:
##     degree gamma coef0  cost     error dispersion
## 1        2   0.1   0.1 1e-01 0.3000000 0.05753831
## 2        3   0.1   0.1 1e-01 0.3000000 0.05753831
## 3        4   0.1   0.1 1e-01 0.3000000 0.05753831
## 4        5   0.1   0.1 1e-01 0.3000000 0.05753831
## 5        2   1.0   0.1 1e-01 0.3000000 0.04151332
## 6        3   1.0   0.1 1e-01 0.2928571 0.05313313
## 7        4   1.0   0.1 1e-01 0.2857143 0.06353173
## 8        5   1.0   0.1 1e-01 0.2757143 0.06425925
## 9        2  10.0   0.1 1e-01 0.3157143 0.05014718
## 10       3  10.0   0.1 1e-01 0.2942857 0.05437753
## 11       4  10.0   0.1 1e-01 0.2842857 0.06579360
## 12       5  10.0   0.1 1e-01 0.2742857 0.06452337
## 13       2   0.1   0.5 1e-01 0.2985714 0.05489632
## 14       3   0.1   0.5 1e-01 0.2814286 0.06390540
## 15       4   0.1   0.5 1e-01 0.2628571 0.06466379
## 16       5   0.1   0.5 1e-01 0.2542857 0.06274160
## 17       2   1.0   0.5 1e-01 0.2985714 0.03894877
## 18       3   1.0   0.5 1e-01 0.2957143 0.05261851
## 19       4   1.0   0.5 1e-01 0.2842857 0.05652443
## 20       5   1.0   0.5 1e-01 0.2728571 0.06782999
## 21       2  10.0   0.5 1e-01 0.3157143 0.05014718
## 22       3  10.0   0.5 1e-01 0.2942857 0.05437753
## 23       4  10.0   0.5 1e-01 0.2842857 0.06579360
## 24       5  10.0   0.5 1e-01 0.2757143 0.06425925
## 25       2   0.1   1.0 1e-01 0.2857143 0.06632566
## 26       3   0.1   1.0 1e-01 0.2557143 0.07267172
## 27       4   0.1   1.0 1e-01 0.2442857 0.05364281
## 28       5   0.1   1.0 1e-01 0.2542857 0.03915201
## 29       2   1.0   1.0 1e-01 0.2985714 0.03894877
## 30       3   1.0   1.0 1e-01 0.2928571 0.05005666
## 31       4   1.0   1.0 1e-01 0.2900000 0.05431495
## 32       5   1.0   1.0 1e-01 0.2685714 0.07024415
## 33       2  10.0   1.0 1e-01 0.3171429 0.05075395
## 34       3  10.0   1.0 1e-01 0.2928571 0.05313313
## 35       4  10.0   1.0 1e-01 0.2857143 0.06353173
## 36       5  10.0   1.0 1e-01 0.2757143 0.06425925
## 37       2   0.1   2.0 1e-01 0.2642857 0.06908866
## 38       3   0.1   2.0 1e-01 0.2457143 0.04248516
```

```
## 39        4   0.1   2.0 1e-01 0.2485714 0.04374737
## 40        5   0.1   2.0 1e-01 0.2957143 0.04619293
## 41        2   1.0   2.0 1e-01 0.2942857 0.04216370
## 42        3   1.0   2.0 1e-01 0.3000000 0.04416009
## 43        4   1.0   2.0 1e-01 0.2942857 0.05682451
## 44        5   1.0   2.0 1e-01 0.2800000 0.07383256
## 45        2  10.0   2.0 1e-01 0.3185714 0.04858543
## 46        3  10.0   2.0 1e-01 0.2942857 0.05520524
## 47        4  10.0   2.0 1e-01 0.2871429 0.06151894
## 48        5  10.0   2.0 1e-01 0.2742857 0.06487385
## 49        2   0.1   3.0 1e-01 0.2528571 0.07064651
## 50        3   0.1   3.0 1e-01 0.2400000 0.04799849
## 51        4   0.1   3.0 1e-01 0.2757143 0.03929654
## 52        5   0.1   3.0 1e-01 0.3114286 0.04194803
## 53        2   1.0   3.0 1e-01 0.2971429 0.04353954
## 54        3   1.0   3.0 1e-01 0.3057143 0.04865539
## 55        4   1.0   3.0 1e-01 0.2942857 0.05395892
## 56        5   1.0   3.0 1e-01 0.2857143 0.06317381
## 57        2  10.0   3.0 1e-01 0.3171429 0.05545115
## 58        3  10.0   3.0 1e-01 0.2942857 0.05520524
## 59        4  10.0   3.0 1e-01 0.2842857 0.05888226
## 60        5  10.0   3.0 1e-01 0.2728571 0.06782999
## 61        2   0.1   0.1 1e+00 0.2442857 0.04783285
## 62        3   0.1   0.1 1e+00 0.2500000 0.04960159
## 63        4   0.1   0.1 1e+00 0.2585714 0.05732115
## 64        5   0.1   0.1 1e+00 0.2600000 0.06127889
## 65        2   1.0   0.1 1e+00 0.3171429 0.05075395
## 66        3   1.0   0.1 1e+00 0.2928571 0.05313313
## 67        4   1.0   0.1 1e+00 0.2857143 0.06353173
## 68        5   1.0   0.1 1e+00 0.2757143 0.06425925
## 69        2  10.0   0.1 1e+00 0.3157143 0.05014718
## 70        3  10.0   0.1 1e+00 0.2942857 0.05437753
## 71        4  10.0   0.1 1e+00 0.2842857 0.06579360
## 72        5  10.0   0.1 1e+00 0.2742857 0.06452337
## 73        2   0.1   0.5 1e+00 0.2457143 0.04752371
## 74        3   0.1   0.5 1e+00 0.2528571 0.03566663
## 75        4   0.1   0.5 1e+00 0.2771429 0.05048517
## 76        5   0.1   0.5 1e+00 0.2814286 0.04996597
## 77        2   1.0   0.5 1e+00 0.3214286 0.05005666
## 78        3   1.0   0.5 1e+00 0.2957143 0.05261851
## 79        4   1.0   0.5 1e+00 0.2842857 0.05652443
## 80        5   1.0   0.5 1e+00 0.2728571 0.06782999
## 81        2  10.0   0.5 1e+00 0.3157143 0.05014718
## 82        3  10.0   0.5 1e+00 0.2942857 0.05437753
## 83        4  10.0   0.5 1e+00 0.2842857 0.06579360
## 84        5  10.0   0.5 1e+00 0.2757143 0.06425925
## 85        2   0.1   1.0 1e+00 0.2400000 0.04507489
## 86        3   0.1   1.0 1e+00 0.2585714 0.04439056
## 87        4   0.1   1.0 1e+00 0.2871429 0.04638887
## 88        5   0.1   1.0 1e+00 0.2900000 0.04261838
## 89        2   1.0   1.0 1e+00 0.3214286 0.05005666
## 90        3   1.0   1.0 1e+00 0.2928571 0.05005666
```

```
## 91        4   1.0   1.0 1e+00 0.2900000 0.05431495
## 92        5   1.0   1.0 1e+00 0.2685714 0.07024415
## 93        2  10.0   1.0 1e+00 0.3171429 0.05075395
## 94        3  10.0   1.0 1e+00 0.2928571 0.05313313
## 95        4  10.0   1.0 1e+00 0.2857143 0.06353173
## 96        5  10.0   1.0 1e+00 0.2757143 0.06425925
## 97        2   0.1   2.0 1e+00 0.2385714 0.04314717
## 98        3   0.1   2.0 1e+00 0.2828571 0.04704415
## 99        4   0.1   2.0 1e+00 0.3114286 0.04194803
## 100       5   0.1   2.0 1e+00 0.3057143 0.04865539
## 101       2   1.0   2.0 1e+00 0.3200000 0.04911923
## 102       3   1.0   2.0 1e+00 0.3000000 0.04416009
## 103       4   1.0   2.0 1e+00 0.2942857 0.05682451
## 104       5   1.0   2.0 1e+00 0.2800000 0.07383256
## 105       2  10.0   2.0 1e+00 0.3185714 0.04858543
## 106       3  10.0   2.0 1e+00 0.2942857 0.05520524
## 107       4  10.0   2.0 1e+00 0.2871429 0.06151894
## 108       5  10.0   2.0 1e+00 0.2742857 0.06487385
## 109       2   0.1   3.0 1e+00 0.2442857 0.04589745
## 110       3   0.1   3.0 1e+00 0.2957143 0.03812499
## 111       4   0.1   3.0 1e+00 0.3185714 0.04811645
## 112       5   0.1   3.0 1e+00 0.3114286 0.04194803
## 113       2   1.0   3.0 1e+00 0.3185714 0.04858543
## 114       3   1.0   3.0 1e+00 0.3057143 0.04865539
## 115       4   1.0   3.0 1e+00 0.2942857 0.05395892
## 116       5   1.0   3.0 1e+00 0.2857143 0.06317381
## 117       2  10.0   3.0 1e+00 0.3171429 0.05545115
## 118       3  10.0   3.0 1e+00 0.2942857 0.05520524
## 119       4  10.0   3.0 1e+00 0.2842857 0.05888226
## 120       5  10.0   3.0 1e+00 0.2728571 0.06782999
## 121       2   0.1   0.1 1e+01 0.2985714 0.03894877
## 122       3   0.1   0.1 1e+01 0.2914286 0.04771419
## 123       4   0.1   0.1 1e+01 0.2900000 0.05431495
## 124       5   0.1   0.1 1e+01 0.2685714 0.07024415
## 125       2   1.0   0.1 1e+01 0.3171429 0.05075395
## 126       3   1.0   0.1 1e+01 0.2928571 0.05313313
## 127       4   1.0   0.1 1e+01 0.2857143 0.06353173
## 128       5   1.0   0.1 1e+01 0.2757143 0.06425925
## 129       2  10.0   0.1 1e+01 0.3157143 0.05014718
## 130       3  10.0   0.1 1e+01 0.2942857 0.05437753
## 131       4  10.0   0.1 1e+01 0.2842857 0.06579360
## 132       5  10.0   0.1 1e+01 0.2742857 0.06452337
## 133       2   0.1   0.5 1e+01 0.3042857 0.04366955
## 134       3   0.1   0.5 1e+01 0.3100000 0.04366955
## 135       4   0.1   0.5 1e+01 0.2928571 0.04530071
## 136       5   0.1   0.5 1e+01 0.2900000 0.05676462
## 137       2   1.0   0.5 1e+01 0.3214286 0.05005666
## 138       3   1.0   0.5 1e+01 0.2957143 0.05261851
## 139       4   1.0   0.5 1e+01 0.2842857 0.05652443
## 140       5   1.0   0.5 1e+01 0.2728571 0.06782999
## 141       2  10.0   0.5 1e+01 0.3157143 0.05014718
## 142       3  10.0   0.5 1e+01 0.2942857 0.05437753
```

```
## 143       4  10.0   0.5 1e+01 0.2842857 0.06579360
## 144       5  10.0   0.5 1e+01 0.2757143 0.06425925
## 145       2   0.1   1.0 1e+01 0.2985714 0.04335687
## 146       3   0.1   1.0 1e+01 0.3142857 0.04416009
## 147       4   0.1   1.0 1e+01 0.3000000 0.04665695
## 148       5   0.1   1.0 1e+01 0.2900000 0.04261838
## 149       2   1.0   1.0 1e+01 0.3214286 0.05005666
## 150       3   1.0   1.0 1e+01 0.2928571 0.05005666
## 151       4   1.0   1.0 1e+01 0.2900000 0.05431495
## 152       5   1.0   1.0 1e+01 0.2685714 0.07024415
## 153       2  10.0   1.0 1e+01 0.3171429 0.05075395
## 154       3  10.0   1.0 1e+01 0.2928571 0.05313313
## 155       4  10.0   1.0 1e+01 0.2857143 0.06353173
## 156       5  10.0   1.0 1e+01 0.2757143 0.06425925
## 157       2   0.1   2.0 1e+01 0.2942857 0.03512207
## 158       3   0.1   2.0 1e+01 0.3228571 0.04957872
## 159       4   0.1   2.0 1e+01 0.3114286 0.04194803
## 160       5   0.1   2.0 1e+01 0.3057143 0.04865539
## 161       2   1.0   2.0 1e+01 0.3200000 0.04911923
## 162       3   1.0   2.0 1e+01 0.3000000 0.04416009
## 163       4   1.0   2.0 1e+01 0.2942857 0.05682451
## 164       5   1.0   2.0 1e+01 0.2800000 0.07383256
## 165       2  10.0   2.0 1e+01 0.3185714 0.04858543
## 166       3  10.0   2.0 1e+01 0.2942857 0.05520524
## 167       4  10.0   2.0 1e+01 0.2871429 0.06151894
## 168       5  10.0   2.0 1e+01 0.2742857 0.06487385
## 169       2   0.1   3.0 1e+01 0.2957143 0.03629684
## 170       3   0.1   3.0 1e+01 0.3242857 0.04811645
## 171       4   0.1   3.0 1e+01 0.3185714 0.04811645
## 172       5   0.1   3.0 1e+01 0.3114286 0.04194803
## 173       2   1.0   3.0 1e+01 0.3185714 0.04858543
## 174       3   1.0   3.0 1e+01 0.3057143 0.04865539
## 175       4   1.0   3.0 1e+01 0.2942857 0.05395892
## 176       5   1.0   3.0 1e+01 0.2857143 0.06317381
## 177       2  10.0   3.0 1e+01 0.3171429 0.05545115
## 178       3  10.0   3.0 1e+01 0.2942857 0.05520524
## 179       4  10.0   3.0 1e+01 0.2842857 0.05888226
## 180       5  10.0   3.0 1e+01 0.2728571 0.06782999
## 181       2   0.1   0.1 1e+02 0.3214286 0.05005666
## 182       3   0.1   0.1 1e+02 0.2928571 0.05005666
## 183       4   0.1   0.1 1e+02 0.2900000 0.05431495
## 184       5   0.1   0.1 1e+02 0.2685714 0.07024415
## 185       2   1.0   0.1 1e+02 0.3171429 0.05075395
## 186       3   1.0   0.1 1e+02 0.2928571 0.05313313
## 187       4   1.0   0.1 1e+02 0.2857143 0.06353173
## 188       5   1.0   0.1 1e+02 0.2757143 0.06425925
## 189       2  10.0   0.1 1e+02 0.3157143 0.05014718
## 190       3  10.0   0.1 1e+02 0.2942857 0.05437753
## 191       4  10.0   0.1 1e+02 0.2842857 0.06579360
## 192       5  10.0   0.1 1e+02 0.2742857 0.06452337
## 193       2   0.1   0.5 1e+02 0.3185714 0.05261851
## 194       3   0.1   0.5 1e+02 0.3100000 0.04366955
```

```
## 195       4   0.1   0.5 1e+02 0.2928571 0.04530071
## 196       5   0.1   0.5 1e+02 0.2900000 0.05676462
## 197       2   1.0   0.5 1e+02 0.3214286 0.05005666
## 198       3   1.0   0.5 1e+02 0.2957143 0.05261851
## 199       4   1.0   0.5 1e+02 0.2842857 0.05652443
## 200       5   1.0   0.5 1e+02 0.2728571 0.06782999
## 201       2  10.0   0.5 1e+02 0.3157143 0.05014718
## 202       3  10.0   0.5 1e+02 0.2942857 0.05437753
## 203       4  10.0   0.5 1e+02 0.2842857 0.06579360
## 204       5  10.0   0.5 1e+02 0.2757143 0.06425925
## 205       2   0.1   1.0 1e+02 0.3171429 0.05379056
## 206       3   0.1   1.0 1e+02 0.3142857 0.04416009
## 207       4   0.1   1.0 1e+02 0.3000000 0.04665695
## 208       5   0.1   1.0 1e+02 0.2900000 0.04261838
## 209       2   1.0   1.0 1e+02 0.3214286 0.05005666
## 210       3   1.0   1.0 1e+02 0.2928571 0.05005666
## 211       4   1.0   1.0 1e+02 0.2900000 0.05431495
## 212       5   1.0   1.0 1e+02 0.2685714 0.07024415
## 213       2  10.0   1.0 1e+02 0.3171429 0.05075395
## 214       3  10.0   1.0 1e+02 0.2928571 0.05313313
## 215       4  10.0   1.0 1e+02 0.2857143 0.06353173
## 216       5  10.0   1.0 1e+02 0.2757143 0.06425925
## 217       2   0.1   2.0 1e+02 0.3185714 0.05218578
## 218       3   0.1   2.0 1e+02 0.3228571 0.04957872
## 219       4   0.1   2.0 1e+02 0.3114286 0.04194803
## 220       5   0.1   2.0 1e+02 0.3057143 0.04865539
## 221       2   1.0   2.0 1e+02 0.3200000 0.04911923
## 222       3   1.0   2.0 1e+02 0.3000000 0.04416009
## 223       4   1.0   2.0 1e+02 0.2942857 0.05682451
## 224       5   1.0   2.0 1e+02 0.2800000 0.07383256
## 225       2  10.0   2.0 1e+02 0.3185714 0.04858543
## 226       3  10.0   2.0 1e+02 0.2942857 0.05520524
## 227       4  10.0   2.0 1e+02 0.2871429 0.06151894
## 228       5  10.0   2.0 1e+02 0.2742857 0.06487385
## 229       2   0.1   3.0 1e+02 0.3214286 0.04821061
## 230       3   0.1   3.0 1e+02 0.3242857 0.04811645
## 231       4   0.1   3.0 1e+02 0.3185714 0.04811645
## 232       5   0.1   3.0 1e+02 0.3114286 0.04194803
## 233       2   1.0   3.0 1e+02 0.3185714 0.04858543
## 234       3   1.0   3.0 1e+02 0.3057143 0.04865539
## 235       4   1.0   3.0 1e+02 0.2942857 0.05395892
## 236       5   1.0   3.0 1e+02 0.2857143 0.06317381
## 237       2  10.0   3.0 1e+02 0.3171429 0.05545115
## 238       3  10.0   3.0 1e+02 0.2942857 0.05520524
## 239       4  10.0   3.0 1e+02 0.2842857 0.05888226
## 240       5  10.0   3.0 1e+02 0.2728571 0.06782999
## 241       2   0.1   0.1 1e+03 0.3214286 0.05005666
## 242       3   0.1   0.1 1e+03 0.2928571 0.05005666
## 243       4   0.1   0.1 1e+03 0.2900000 0.05431495
## 244       5   0.1   0.1 1e+03 0.2685714 0.07024415
## 245       2   1.0   0.1 1e+03 0.3171429 0.05075395
## 246       3   1.0   0.1 1e+03 0.2928571 0.05313313
```

```
## 247     4   1.0   0.1 1e+03 0.2857143 0.06353173
## 248     5   1.0   0.1 1e+03 0.2757143 0.06425925
## 249     2  10.0   0.1 1e+03 0.3157143 0.05014718
## 250     3  10.0   0.1 1e+03 0.2942857 0.05437753
## 251     4  10.0   0.1 1e+03 0.2842857 0.06579360
## 252     5  10.0   0.1 1e+03 0.2742857 0.06452337
## 253     2   0.1   0.5 1e+03 0.3185714 0.05261851
## 254     3   0.1   0.5 1e+03 0.3100000 0.04366955
## 255     4   0.1   0.5 1e+03 0.2928571 0.04530071
## 256     5   0.1   0.5 1e+03 0.2900000 0.05676462
## 257     2   1.0   0.5 1e+03 0.3214286 0.05005666
## 258     3   1.0   0.5 1e+03 0.2957143 0.05261851
## 259     4   1.0   0.5 1e+03 0.2842857 0.05652443
## 260     5   1.0   0.5 1e+03 0.2728571 0.06782999
## 261     2  10.0   0.5 1e+03 0.3157143 0.05014718
## 262     3  10.0   0.5 1e+03 0.2942857 0.05437753
## 263     4  10.0   0.5 1e+03 0.2842857 0.06579360
## 264     5  10.0   0.5 1e+03 0.2757143 0.06425925
## 265     2   0.1   1.0 1e+03 0.3171429 0.05379056
## 266     3   0.1   1.0 1e+03 0.3142857 0.04416009
## 267     4   0.1   1.0 1e+03 0.3000000 0.04665695
## 268     5   0.1   1.0 1e+03 0.2900000 0.04261838
## 269     2   1.0   1.0 1e+03 0.3214286 0.05005666
## 270     3   1.0   1.0 1e+03 0.2928571 0.05005666
## 271     4   1.0   1.0 1e+03 0.2900000 0.05431495
## 272     5   1.0   1.0 1e+03 0.2685714 0.07024415
## 273     2  10.0   1.0 1e+03 0.3171429 0.05075395
## 274     3  10.0   1.0 1e+03 0.2928571 0.05313313
## 275     4  10.0   1.0 1e+03 0.2857143 0.06353173
## 276     5  10.0   1.0 1e+03 0.2757143 0.06425925
## 277     2   0.1   2.0 1e+03 0.3185714 0.05218578
## 278     3   0.1   2.0 1e+03 0.3228571 0.04957872
## 279     4   0.1   2.0 1e+03 0.3114286 0.04194803
## 280     5   0.1   2.0 1e+03 0.3057143 0.04865539
## 281     2   1.0   2.0 1e+03 0.3200000 0.04911923
## 282     3   1.0   2.0 1e+03 0.3000000 0.04416009
## 283     4   1.0   2.0 1e+03 0.2942857 0.05682451
## 284     5   1.0   2.0 1e+03 0.2800000 0.07383256
## 285     2  10.0   2.0 1e+03 0.3185714 0.04858543
## 286     3  10.0   2.0 1e+03 0.2942857 0.05520524
## 287     4  10.0   2.0 1e+03 0.2871429 0.06151894
## 288     5  10.0   2.0 1e+03 0.2742857 0.06487385
## 289     2   0.1   3.0 1e+03 0.3214286 0.04821061
## 290     3   0.1   3.0 1e+03 0.3242857 0.04811645
## 291     4   0.1   3.0 1e+03 0.3185714 0.04811645
## 292     5   0.1   3.0 1e+03 0.3114286 0.04194803
## 293     2   1.0   3.0 1e+03 0.3185714 0.04858543
## 294     3   1.0   3.0 1e+03 0.3057143 0.04865539
## 295     4   1.0   3.0 1e+03 0.2942857 0.05395892
## 296     5   1.0   3.0 1e+03 0.2857143 0.06317381
## 297     2  10.0   3.0 1e+03 0.3171429 0.05545115
## 298     3  10.0   3.0 1e+03 0.2942857 0.05520524
```

```
## 299      4  10.0   3.0 1e+03 0.2842857 0.05888226
## 300      5  10.0   3.0 1e+03 0.2728571 0.06782999
```

Observamos el mejor modelo y sus valores

```
tune_p$best.model
```

```
##
## Call:
## best.svm(x = target ~ ., data = train, degree = c(2, 3, 4, 5), gamma = c(0.1,
##     1, 10), coef0 = c(0.1, 0.5, 1, 2, 3), cost = c(0.1, 1, 10, 100,
##     1000), kernel = "polynomial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  2
##      coef.0:  2
##
## Number of Support Vectors:  411
```

```
#Mejores valores
tune_p$best.parameters$cost
```

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

```
tune_p$best.parameters$gamma
```

```
## [1] 0.1
```

```
tune_p$best.parameters$degree
```

```
## [1] 2
```

```
tune_p$best.parameters$coef0
```

```
## [1] 2
```

Paso 2. Entrenamiento del modelo

Entrenamos el modelo con los hiperparámetros establecidos previamente.

```
svm_p <- svm(target ~ ., data = train, type = "C-classification", kernel = "polynomia
l",
                 degree = tune_p$best.parameters$degree,
                 cost = tune_p$best.parameters$cost,
                 gamma = tune_p$best.parameters$gamma,
                 coef0 = tune_p$best.parameters$coef0,
             probability = TRUE)

summary(svm_p)
```

```
##
## Call:
## svm(formula = target ~ ., data = train, type = "C-classification",
##     kernel = "polynomial", degree = tune_p$best.parameters$degree,
##     cost = tune_p$best.parameters$cost, gamma = tune_p$best.parameters$gamma,
##     coef0 = tune_p$best.parameters$coef0, probability = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  2
##      coef.0:  2
##
## Number of Support Vectors:  411
##
##  ( 227 184 )
##
##
## Number of Classes:  2
##
## Levels:
##  Bad Good
```

Paso 3. Predict y matriz de confusión

Sacamos la matriz de confusión.

```
svm_score_p_Response <- predict(svm_p, test, type="response")
head(svm_score_p_Response)
```

```
##    4    7    9   13   14   23
## Good Good Good Good Good Good
## Levels: Bad Good
```

```
MC_svm_p <- confusionMatrix(svm_score_p_Response, test$target , positive = 'Good')
MC_svm_p
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Bad Good
##       Bad   37   26
##       Good  53  184
##
##                Accuracy : 0.7367
##                  95% CI : (0.683, 0.7856)
##     No Information Rate : 0.7
##     P-Value [Acc > NIR] : 0.091793
##
##                   Kappa : 0.3142
##
##  Mcnemar's Test P-Value : 0.003442
##
##             Sensitivity : 0.8762
##             Specificity : 0.4111
##          Pos Pred Value : 0.7764
##          Neg Pred Value : 0.5873
##              Prevalence : 0.7000
##          Detection Rate : 0.6133
##    Detection Prevalence : 0.7900
##       Balanced Accuracy : 0.6437
##
##        'Positive' Class : Good
##
```

Paso 4. Predict con probabilidades y umbrales

1º) En este paso sacamos la probabilidad de cada cliente de devolver el crédito.

```
svm_score_p <- predict(svm_p, test, probability = TRUE)
svm_score_p_Prob <- attr(svm_score_p, "probabilities")[,1]
head(svm_score_p_Prob)
```

```
##         4         7         9        13        14        23
## 0.8836713 0.8626002 0.9629161 0.7366813 0.6054464 0.7682053
```

2º) Ahora transformamos la probabilidad obtenida en una decisión binaria de si conceder el crédito (Sí lo va a devolver) o no (No lo va a devolver).

Con la función umbrales probamos diferentes cortes

```
umb_svm_p<-umbrales(test$target,svm_score_p_Prob)
umb_svm_p
```

```
##    umbral  acierto precision cobertura        F1
## 1    0.05  0.05000   0.05000  0.050000  0.050000
## 2    0.10 69.66667  69.89967 99.523810 82.121807
## 3    0.15 69.33333  69.79866 99.047619 81.889764
## 4    0.20 69.66667  70.16949 98.571429 81.980198
## 5    0.25 70.33333  70.64846 98.571429 82.306163
## 6    0.30 71.66667  71.62630 98.571429 82.965932
## 7    0.35 70.66667  71.94245 95.238095 81.967213
## 8    0.40 70.33333  72.00000 94.285714 81.649485
## 9    0.45 72.00000  74.04580 92.380952 82.203390
## 10   0.50 73.66667  76.09562 90.952381 82.863341
## 11   0.55 73.00000  76.76349 88.095238 82.039911
## 12   0.60 73.66667  78.60262 85.714286 82.004556
## 13   0.65 73.33333  80.95238 80.952381 80.952381
## 14   0.70 70.66667  83.15217 72.857143 77.664975
## 15   0.75 66.00000  82.53012 65.238095 72.872340
## 16   0.80 59.66667  85.60000 50.952381 63.880597
## 17   0.85 53.33333  89.77273 37.619048 53.020134
## 18   0.90 41.66667  94.87179 17.619048 29.718876
## 19   0.95 31.00000 100.00000  1.428571  2.816901
```

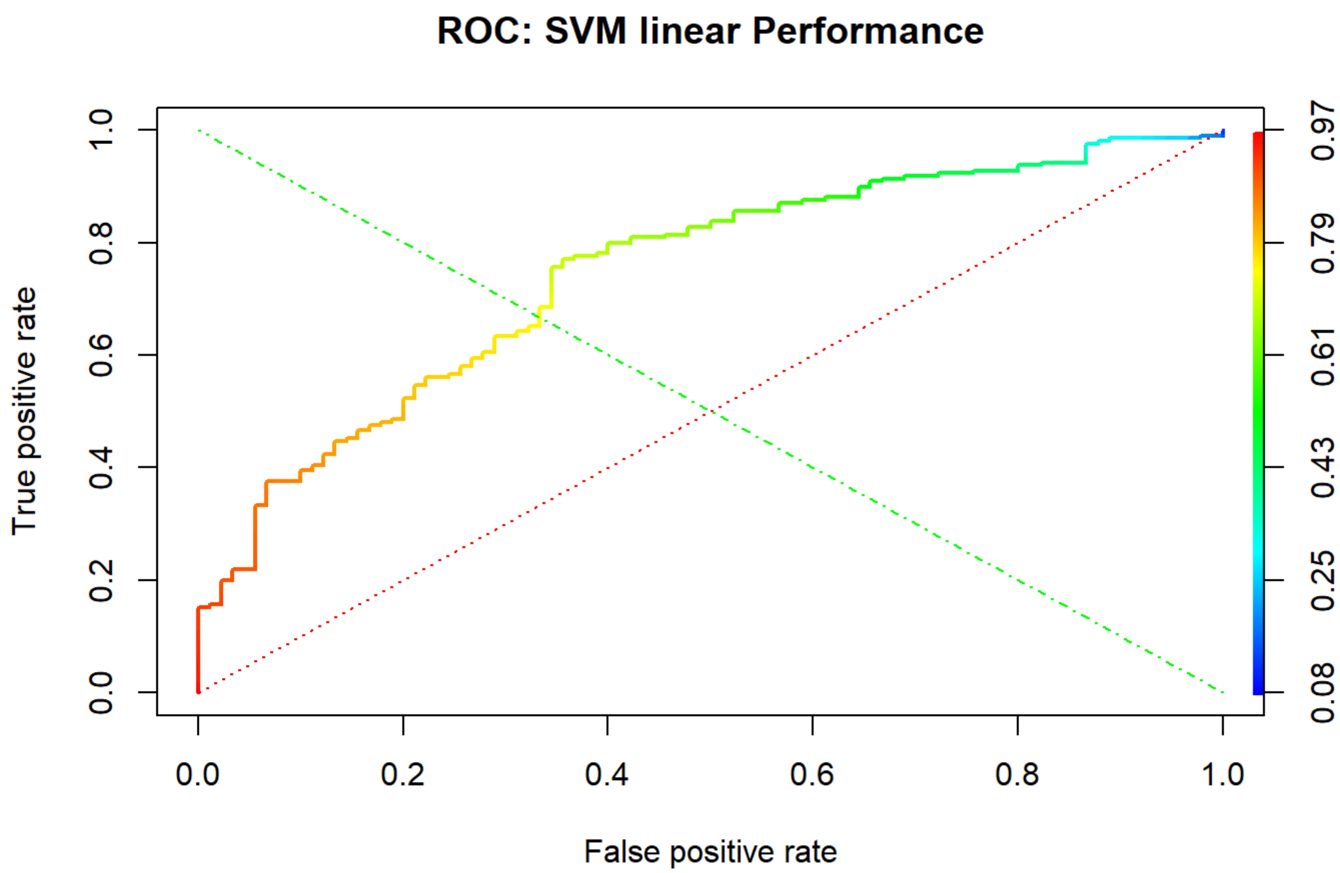
Seleccionamos el umbral que maximiza la F1 (cuando empieza a decaer)

```
umbfinal_svm_p<-umb_svm_p[which.max(umb_svm_p$F1),1]
umbfinal_svm_p
```

```
## [1] 0.3
```

Paso 5. Curva ROC

```
pred_svm_p <- prediction(svm_score_p_Prob, test$target)
perf_svm_p <- performance(pred_svm_p,"tpr","fpr")
#Library(ROCR)
plot(perf_svm_p, lwd=2, colorize=TRUE, main="ROC: SVM linear Performance")
lines(x=c(0, 1), y=c(0, 1), col="red", lwd=1, lty=3);
lines(x=c(1, 0), y=c(0, 1), col="green", lwd=1, lty=4)
```



Paso 6. Métricas definitivas

```
#Matriz de confusión con umbral final
score <- ifelse(svm_score_p_Prob > umbfinal_svm_p, "Good", "Bad")
MC <- table(test$target, score)
Acc_svm_p <- round((MC[1,1] + MC[2,2]) / sum(MC) *100, 2)
Sen_svm_p <- round(MC[2,2] / (MC[2,2] + MC[1,2]) *100, 2)
Pr_svm_p <- round(MC[2,2] / (MC[2,2] + MC[2,1]) *100, 2)
F1_svm_p <- round(2*Pr_svm_p*Sen_svm_p/(Pr_svm_p+Sen_svm_p), 2)

#AUC
AUROC_svm_p <- round(performance(pred_svm_p, measure = "auc")@y.values[[1]]*100, 2)

#Métricas finales del modelo
cat("Acc_svm_p: ", Acc_svm_p,"\tSen_svm_p: ", Sen_svm_p, "\tPr_svm_p:", Pr_svm_p, "\tF
1_svm_p:", F1_svm_p, "\tAUROC_svm_p: ", AUROC_svm_p)
```

```
## Acc_svm_p:  71.67    Sen_svm_p:  71.63    Pr_svm_p: 98.57     F1_svm_p: 82.97     AU
ROC_svm_p:  74.45
```

Se obtiene un modelo con una AUC = 74.45. Moderadamente aceptable.

5. Comparación de los tres modelos

```r
# Etiquetas de filas
models <- c('SVM_l', 'SVM_r', 'SVM_p')

#Accuracy
models_Acc <- c(Acc_svm_l, Acc_svm_r, Acc_svm_p)

#Sensibilidad
models_Sen <- c(Sen_svm_l, Sen_svm_r, Sen_svm_p)

#Precisión
models_Pr <- c(Pr_svm_l, Pr_svm_r, Pr_svm_p)

#F1
models_F1 <- c(F1_svm_l, F1_svm_r, F1_svm_p)

# AUC
models_AUC <- c(AUROC_svm_l, AUROC_svm_r, AUROC_svm_p)
```

```r
# Combinar métricas
metricas <- as.data.frame(cbind(models, models_Acc, models_Sen, models_Pr, models_F1,
 models_AUC))
```

```r
# Colnames
colnames(metricas) <- c("Model", "Acc", "Sen", "Pr", "F1", "AUC")
```

```r
# Tabla final de métricas
kable(metricas, caption ="Comparision of Model Performances")
```

Comparision of Model Performances

Model	Acc	Sen	Pr	F1	AUC
SVM_l	75.33	77.2	91.9	83.91	76.08
SVM_r	71.33	71.09	99.52	82.94	70.63
SVM_p	71.67	71.63	98.57	82.97	74.45

Se observa que el modelo SVM lineal obtiene unos resultados mejores que los otros dos.