UNIVERSIDAD DE GRANADA

Master Universitario en Ingeniería Informática

Sistemas Inteligentes para la Gestión en la Empresa

Práctica 1: Pre-procesamiento de datos y clasificación binaria

Alumno:

Freddy Javier Frere Quintero

ffrere@correo.ugr.es

Curso 2017-2018

Índice

Contenidos 2
Exploración 2
Pre-procesamiento 3
Clasificación 10
R-part 10
Random Forest 12
Redes Neuronales 13
Discusión de resultados 14
Conclusiones 16
Bibliografía 16

Ilustración 1: Resumen de datos 2
Ilustración 2: Dimensiones del conjunto de datos 2
Ilustración 3: Columnas con más del 90% de los valores en 0 3
Ilustración 4: columnas con más del 50% de los valores a NA 3
Ilustración 5: Valores de la variable Loan_status 5
Ilustración 6:Datos de la variable loan_status filtrada 6
Ilustración 7: Equilibrio de las clases 7
Ilustración 8: Matriz de correlación 8
Ilustración 9: Dendograma de agrupaciones 9
Ilustración 10: Modelo R-part 10
Ilustración 11: Modelo R-part “Cross-Validation” 11
Ilustración 12: Árbol Modelo R-part “Cross-Validation” 11
Ilustración 13: Modelo Random Forest 12
Ilustración 14: Modelo Redes Neuronales 13
Ilustración 15: Curva Roc 14
Ilustración 16: f_measure 15
Ilustración 17: Contingencias entre modelos 15

Contenidos

En esta primera práctica de la asignatura Sistemas Inteligentes para la Gestión en la Empresa estudiaremos cómo realizar diversas tareas de pre-procesamiento de datos, como paso previo e imprescindible para el aprendizaje automático.

Exploración

Realizaremos la exploración del conjunto de datos "LoanStats_2017Q4.csv", con el objetivo de predecir el estado de un préstamo (loan_status) a partir del resto de variables. Trataremos el conjunto de datos como un problema de clasificación binaria, con dos posibles salidas: Pago, Impago.

Para obtener un resumen del marco de datos dado, usamos la función "df_status" identificamos de cada variable, la cantidad y porcentaje de ceros (q_zeros y p_zeros respectivamente). Mismas métricas para los valores de NA (q_NA / p_na) y los valores infinitos (q_inf / p_inf). Adicionalmente las dos últimas columnas indican el tipo de datos y la cantidad de valores únicos.

Toda esta información la guardamos en la variable "status".

```
> status <- df_status(data_raw)
```

	variable	q_zeros	p_zeros	q_na	p_na	q_inf	p_inf	type	unique
1	id	0	0.00	118648	100.00	0	0	character	2
2	member_id	0	0.00	118650	100.00	0	0	character	0
3	loan_amnt	0	0.00	2	0.00	0	0	integer	1510
4	funded_amnt	0	0.00	2	0.00	0	0	integer	1510
5	funded_amnt_inv	0	0.00	3	0.00	0	0	integer	1518
6	term	0	0.00	2	0.00	0	0	character	2
7	int_rate	0	0.00	2	0.00	0	0	character	38
8	installment	0	0.00	2	0.00	0	0	numeric	12268
9	grade	0	0.00	2	0.00	0	0	character	7
10	sub_grade	0	0.00	2	0.00	0	0	character	35
11	emp_title	0	0.00	9870	8.32	0	0	character	38288
12	emp_length	0	0.00	9724	8.20	0	0	character	11
13	home_ownership	0	0.00	2	0.00	0	0	character	4
14	annual_inc	212	0.18	2	0.00	0	0	numeric	9455
15	verification_status	0	0.00	2	0.00	0	0	character	3
16	issue_d	0	0.00	2	0.00	0	0	character	3
17	loan_status	0	0.00	2	0.00	0	0	character	7
18	pymnt_plan	0	0.00	2	0.00	0	0	character	2
19	url	0	0.00	118650	100.00	0	0	character	0
20	desc	0	0.00	118650	100.00	0	0	character	0
21	purpose	0	0.00	2	0.00	0	0	character	13
22	title	0	0.00	2	0.00	0	0	character	13
23	zip_code	0	0.00	2	0.00	0	0	character	884
24	addr_state	0	0.00	2	0.00	0	0	character	50
25	dti	182	0.15	224	0.19	0	0	numeric	6324
26	delinq_2yrs	100124	84.39	2	0.00	0	0	integer	26
27	earliest_cr_line	0	0.00	2	0.00	0	0	character	639
28	inq_last_6mths	75940	64.00	2	0.00	0	0	integer	6
29	mths_since_last_delinq	20	0.02	64805	54.62	0	0	integer	133
30	mths_since_last_record	1	0.00	100721	84.89	0	0	integer	122
31	open_acc	7	0.01	2	0.00	0	0	integer	61
32	pub_rec	100719	84.89	2	0.00	0	0	integer	14
33	revol_bal	1203	1.01	2	0.00	0	0	integer	38280
34	revol_util	0	0.00	193	0.16	0	0	character	1079
35	total_acc	0	0.00	2	0.00	0	0	integer	107
36	initial_list_status	0	0.00	2	0.00	0	0	character	2
37	out_prncp	5585	4.71	2	0.00	0	0	numeric	32479
38	out_prncp_inv	5585	4.71	2	0.00	0	0	numeric	34970
39	total_pymnt	175	0.15	2	0.00	0	0	numeric	50326
40	total_pymnt_inv	175	0.15	2	0.00	0	0	numeric	52519
41	total_rec_prncp	184	0.16	2	0.00	0	0	numeric	30895
42	total_rec_int	196	0.17	2	0.00	0	0	numeric	45630
43	total_rec_late_fee	117900	99.37	2	0.00	0	0	numeric	380
44	recoveries	118647	100.00	2	0.00	0	0	numeric	2
45	collection_recovery_fee	118648	100.00	2	0.00	0	0	numeric	1

Ilustración 1: Resumen de datos

Con la función "dim" obtuve las dimensiones del conjunto de datos (como los números de filas y columnas) con el objetivo de cerciorarme que se haya cargado correctamente todos los datos.

```
> dim(data_raw)
[1] 118650    145
```

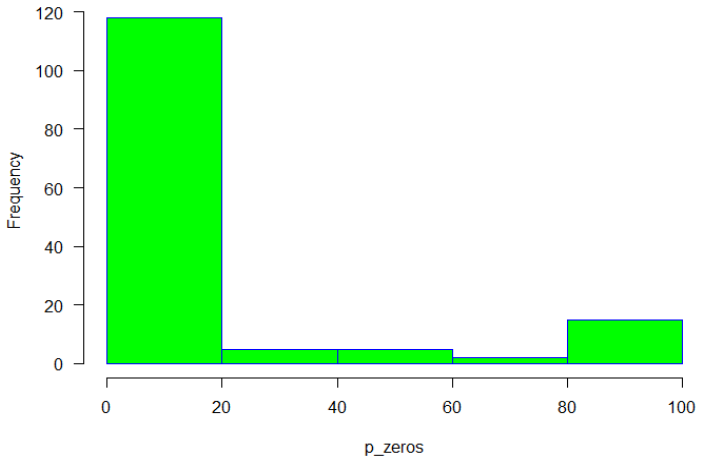
Ilustración 2: Dimensiones del conjunto de datos

Pre-procesamiento

Con los resultados obtenidos procedemos a realizar el Pre-procesamiento con el fin de utilizar técnicas de análisis de datos que me permitan mejorar la calidad del conjunto de datos y así conseguir un mejor porcentaje de clasificación.

Filtramos los diferentes valores de la variable “status” asignándole un nombre a cada una para finalmente eliminar las columnas que no son útiles ya que no nos aporta información.

Identificamos las columnas con más del 90% de los valores en 0.




```
> zero_cols
                      variable
1         total_rec_late_fee
2                  recoveries
3     collection_recovery_fee
4  collections_12_mths_ex_med
5              acc_now_delinq
6    chargeoff_within_12_mths
7                 delinq_amnt
8           num_tl_120dpd_2m
9               num_tl_30dpd
10        num_tl_90g_dpd_24m
11                tax_liens
```

Ilustración 3: Columnas con más del 90% de los valores en 0

Identificamos las columnas con más del 50% de los valores a NA.

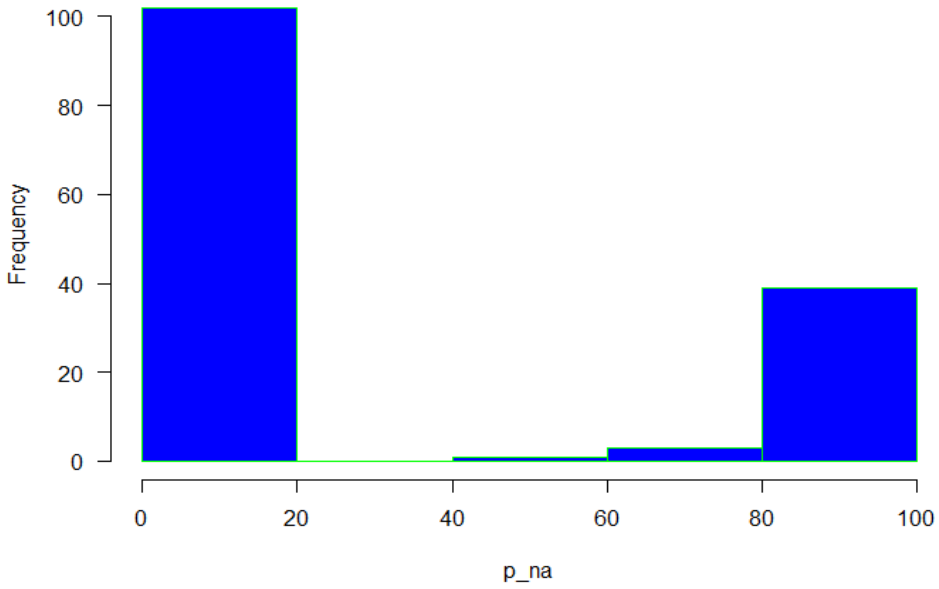


Ilustración 4: columnas con más del 50% de los valores a NA

```
> na_cols
                                    variable
1                                         id
2                                  member_id
3                                        url
4                                       desc
5                      mths_since_last_delinq
6                      mths_since_last_record
7                  mths_since_last_major_derog
8                           annual_inc_joint
9                                  dti_joint
10                  verification_status_joint
11                   mths_since_recent_bc_dlq
12              mths_since_recent_revol_delinq
13                          revol_bal_joint
14                 sec_app_earliest_cr_line
15                   sec_app_inq_last_6mths
16                         sec_app_mort_acc
17                         sec_app_open_acc
18                      sec_app_revol_util
19                     sec_app_open_act_il
20                   sec_app_num_rev_accts
21           sec_app_chargeoff_within_12_mths
22         sec_app_collections_12_mths_ex_med
23        sec_app_mths_since_last_major_derog
24                           hardship_type
25                         hardship_reason
26                         hardship_status
27                           deferral_term
28                         hardship_amount
29                      hardship_start_date
30                        hardship_end_date
31                   payment_plan_start_date
32                         hardship_length
33                           hardship_dpd
34                      hardship_loan_status
35 orig_projected_additional_accrued_interest
36              hardship_payoff_balance_amount
37                hardship_last_payment_amount
38                   debt_settlement_flag_date
39                         settlement_status
40                           settlement_date
41                         settlement_amount
42                     settlement_percentage
43                           settlement_term
```

Identificamos las columnas con valores <= 1 valores diferentes.

```
> eq_cols
                  variable
1                member_id
2                      url
3                     desc
4  collection_recovery_fee
5              policy_code
6            hardship_type
7            deferral_term
8          hardship_length
9        settlement_status
10   settlement_percentage
```

Identificamos las columnas con valores > 75% de valores diferentes.

```
> dif_cols
          variable
1     tot_cur_bal
2 tot_hi_cred_lim
```

Usamos la función “bind_rows” para unir, combinar diferentes tipos conjuntos de datos.

```
# Junta varias de las columnas Inservibles
remove_cols <- bind_rows(
  list(
    zero_cols,
    na_cols,
    eq_cols,
    dif_cols
  )
)
```

Eliminamos las “columnas” que son inservibles, con ayuda de la función “one_of()” seleccionamos las variables deseadas.

```
# Elimina las columnas Inservibles
data <- data_raw %>%
  select(-one_of(remove_cols$variable))
```

Procedemos a tratar la variable loan_status, en la siguiente ilustración observamos los valores que contiene.

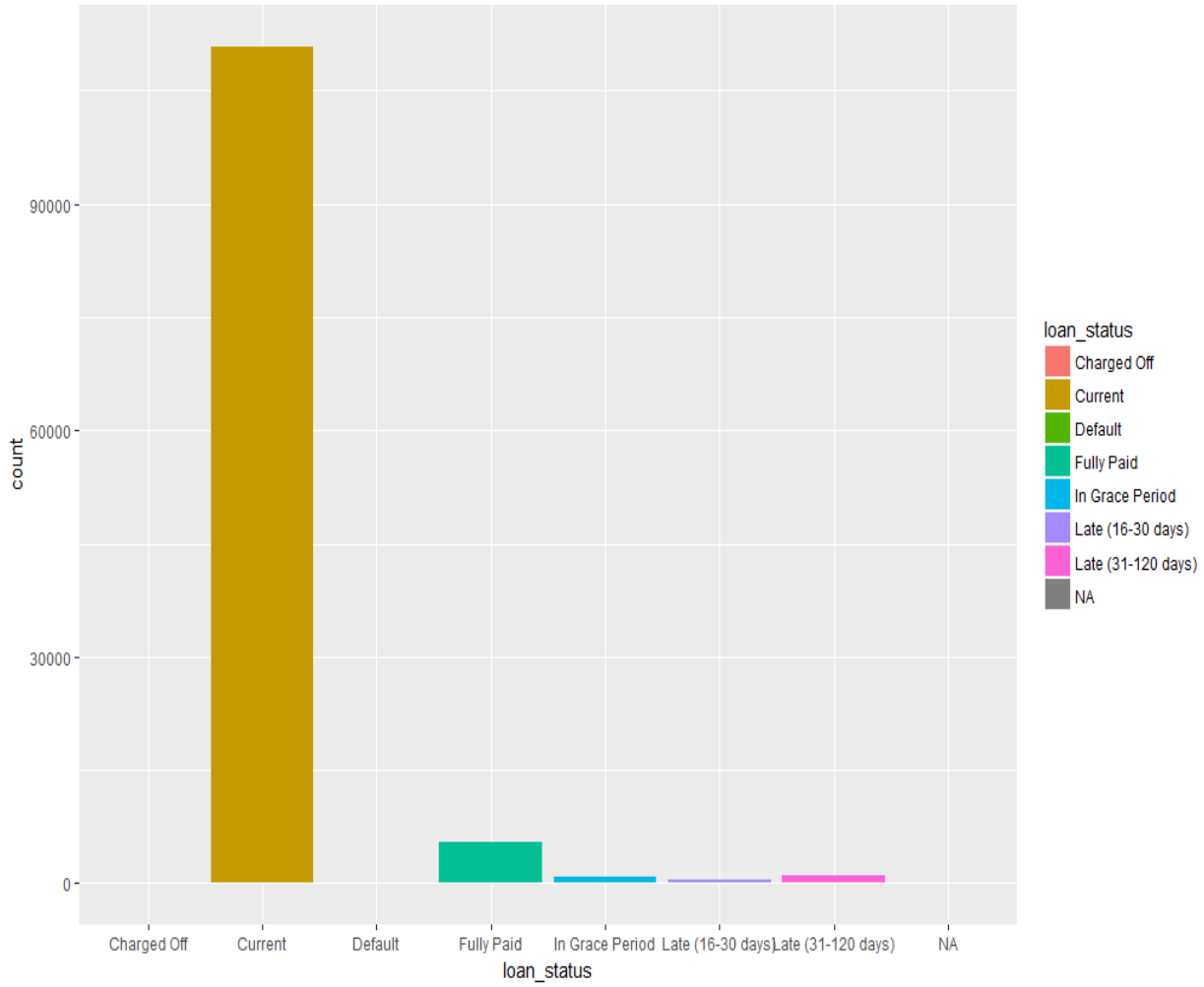


Ilustración 5: Valores de la variable Loan_status

Filtraremos los valores con los que nos quedaremos según lo indica el guion de la práctica.

Utilizando “%in%”, este nos devuelve un vector lógico que indica si hay coincidencia o no para su operando izquierdo. Expresa una condición de filtrado a partir de un conjunto.

```
# Eliminar valores de loan_status no interesantes
data <- data %>%
  filter(loan_status %in% c('Late (16-30 days)', 'Late (31-120 days)', 'In Grace Period', 'Charged Off', 'Current'))
```

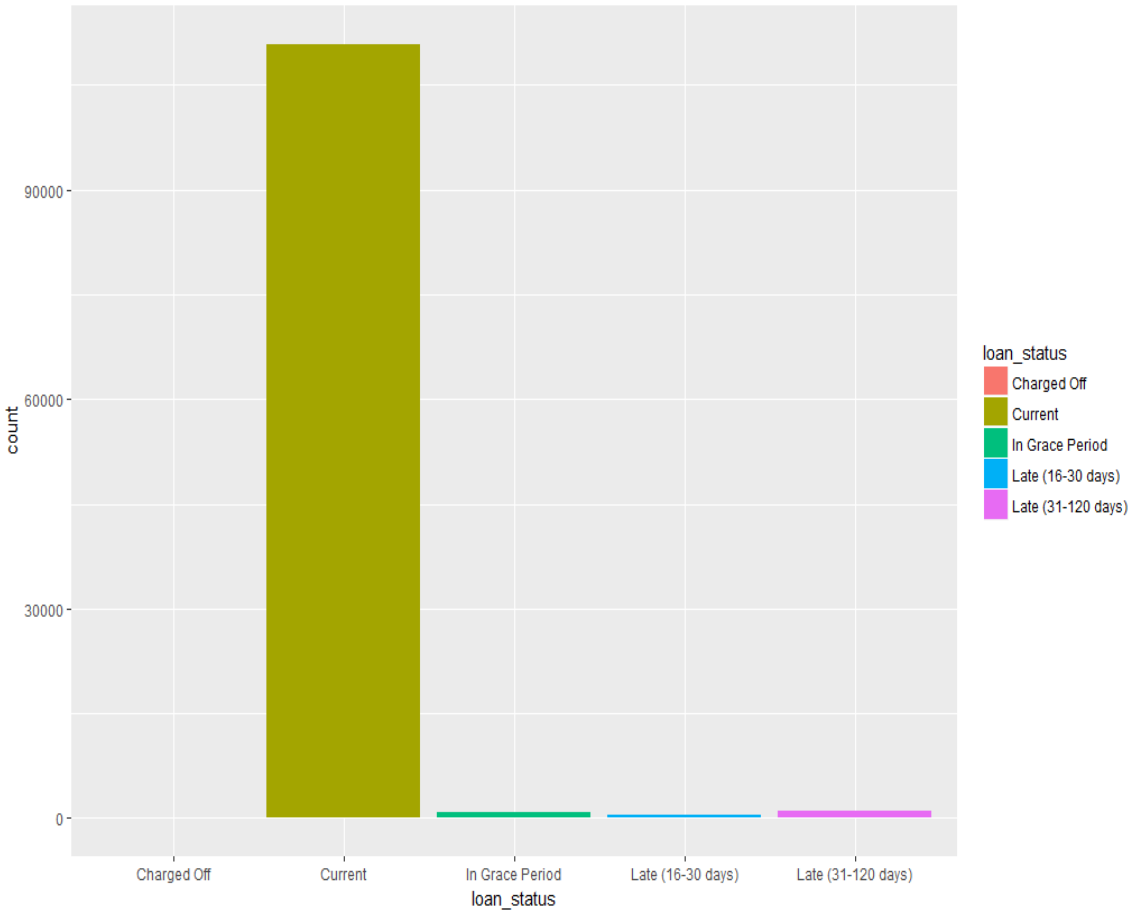


Ilustración 6:Datos de la variable loan_status filtrada.

Procedemos a recodificar la variable loan_status dejándolo con 2 posibles salidas: {Pago, Impago}.

- Impago:
  - Late (16-30 days)
  - Late (31-120 days)
  - In Grace Period
  - Charged Off
- Pago:
  - Current

Con la siguiente función realizamos la recodificación:

```
data <- data %>%
  mutate(loan_status = case_when(
    loan_status == 'Late (16-30 days)'  ~ 'Unpaid',
    loan_status == 'Late (31-120 days)' ~ 'Unpaid',
    loan_status == 'In Grace Period'    ~ 'Unpaid',
    loan_status == 'Charged Off'        ~ 'Unpaid',
    loan_status == 'Current'            ~ 'Paid'))
```

Verificamos cuantos valores de “Paid” y “Unpaid” tenemos y el desequilibro que hay en ellas.

```
> table(data$loan_status)

  Paid Unpaid
110893   2296
```

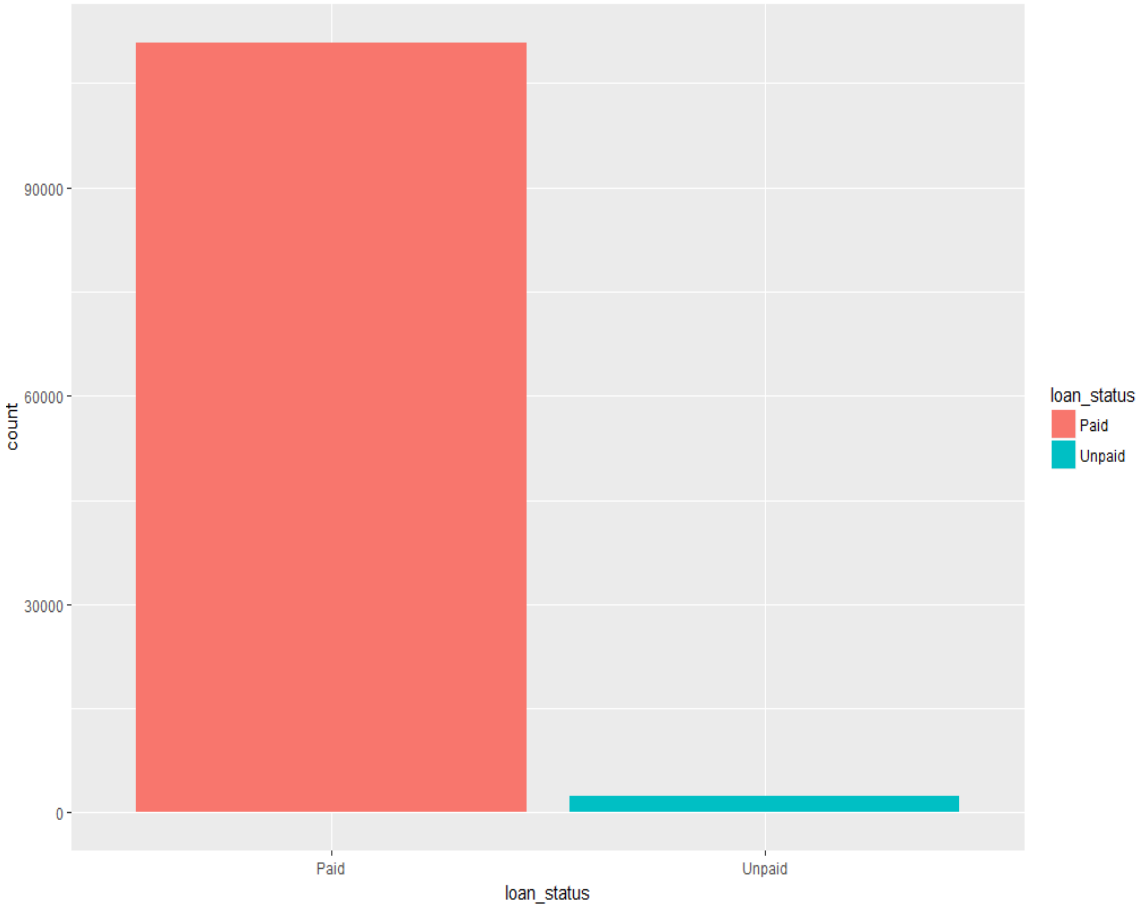


Ilustración 7: Equilibrio de las clases

Ya se hemos quitado las variables que no son de interés, procedemos a estudiar las variables con alta correlación con la variable objetivo y con alta correlación entre sí.

Verificamos las columnas con alta correlación con la variable “loan_status”, para ellos debemos transformar todas las variables con valores que no son numéricos a valores numéricos, con ayuda de la función “mutate_if”, iterara por las columnas y en el caso de que se cumpla la condición se realizara el cambio que nosotros deseamos.

```
data_num <- data %>%
  na.exclude() %>% #quitamos las filas que tenga NA
  mutate_if(is.character, as.factor) %>%
  mutate_if(is.factor, as.numeric)
cor_target <- correlation_table(data_num, target='loan_status')

important_vars <- cor_target %>%
  filter(abs(loan_status) >= 0.02) #Creamos un umbral
data <- data %>%
  select(one_of(important_vars$Variable))
```

Verificamos las columnas con alta correlación entre sí.

```
data_num <- data %>%
  na.exclude() %>%
  mutate_if(is.character, as.factor) %>%
  mutate_if(is.factor, as.numeric)
rcorr_result <- rcorr(as.matrix(data_num))
cor_matrix <- as.tibble(rcorr_result$r, rownames = "variable")
corrplot(rcorr_result$r, type = "upper", order = "original", tl.col = "black", tl.srt = 45)
```

En la siguiente ilustración podemos apreciar la correlación que hay entre cada variable.

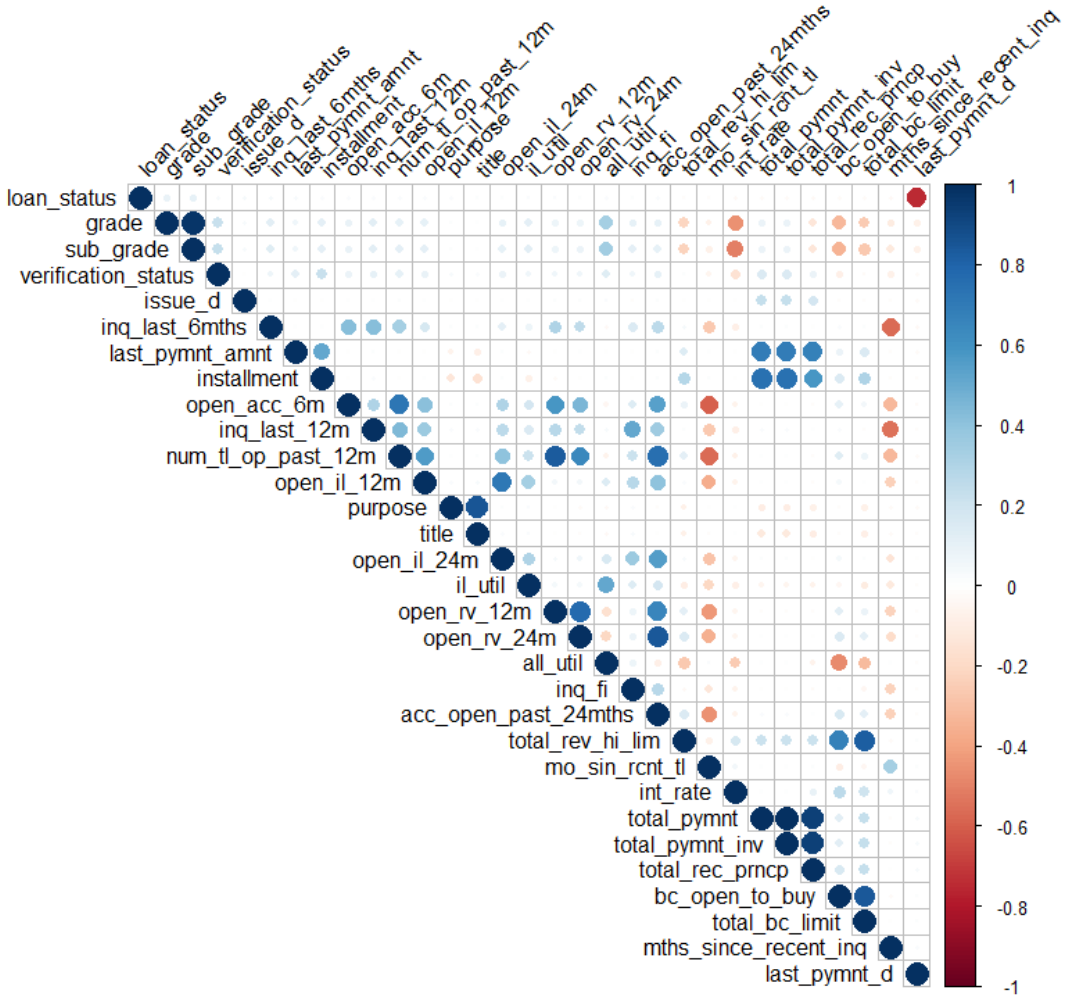


Ilustración 8: Matriz de correlación

Procedemos agrupar con la ayuda de la función “Varclus” a partir de la correlación, que existen en las variables especificando que medida de similaridad queremos. En este caso usaremos la de “Pearson”.

Podemos observar en la siguiente ilustración el dendograma según el criterio de correlación del coeficiente de Pearson.

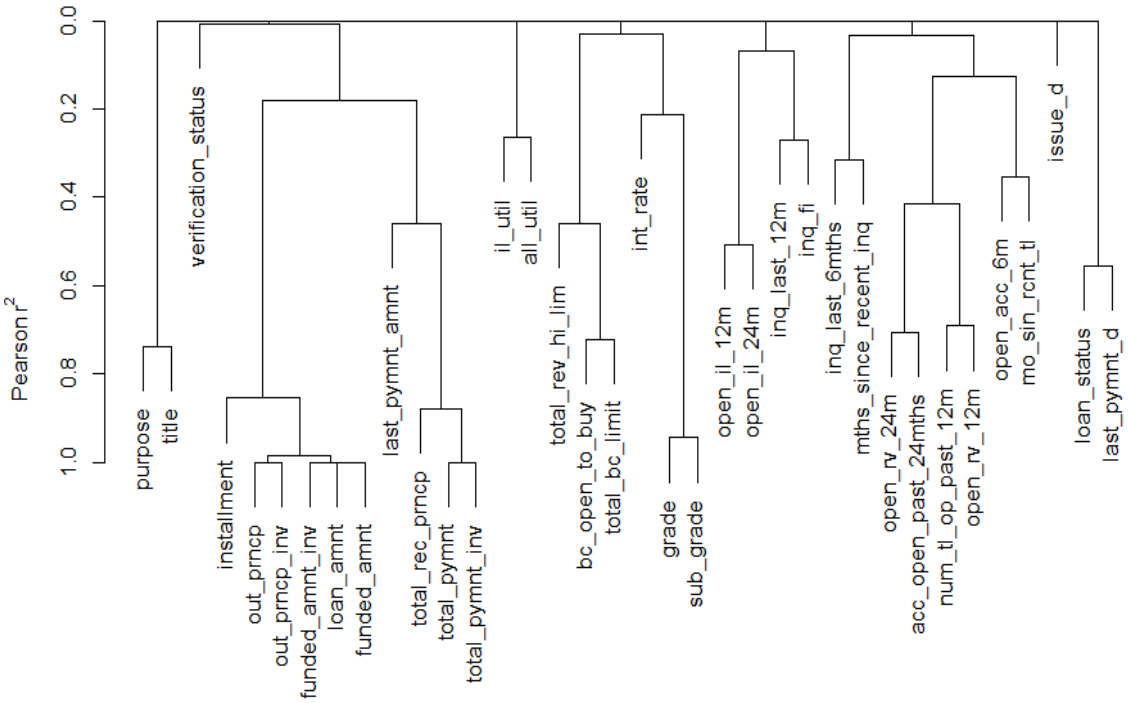


Ilustración 9: Dendograma de agrupaciones

Podemos observar que prácticamente a ciegas hemos conseguido realizar agrupaciones con variables que según por sus nombres tienen sentido que se hayan agrupado.

Con la función “glimpse” observamos cuantas variables, que tipos de datos que tiene y que variables tienen cada una.

```
> glimpse(data)
Observations: 113,189
Variables: 25
$ loan_status           <chr> "Paid", "Paid", "Paid", "Paid", "Paid", "Paid", "Paid", "Paid", "Paid", "Paid", "Unpaid", "P...
$ grade                 <chr> "A", "C", "B", "A", "A", "B", "C", "B", "A", "B", "F", "A", "B", "A", "B", "C", "A", "A", "E...
$ verification_status   <chr> "Not Verified", "Verified", "Source Verified", "Not Verified", "Not Verified", "Source Verif...
$ issue_d               <chr> "Dec-2017", "Dec-2017", "Dec-2017", "Dec-2017", "Dec-2017", "Dec-2017", "Dec-2017", "Dec-201...
$ inq_last_6mths        <int> 0, 0, 1, 1, 0, 0, 0, 2, 0, 0, 1, 3, 0, 0, 1, 0, 0, 0, 2, 1, 0, 0, 0, 0, 0, 2, 0, 0, 0, 2, 0,...
$ funded_amnt_inv       <int> 4800, 16000, 32600, 10000, 9000, 35000, 30000, 2700, 8400, 15000, 11200, 1200, 35000, 40000,...
$ last_pymnt_amnt       <dbl> 144.56, 389.26, 1082.64, 304.59, 271.04, 778.38, 729.87, 87.04, 260.72, 486.98, 366.85, 37.2...
$ open_acc_6m           <int> 0, 0, 4, 1, 0, 0, 1, 2, 0, 0, 1, 2, 0, 0, 1, 0, 0, 0, 2, 2, 0, 1, 0, 0, 2, 3, 1, 2, 0, 2, 2,...
$ open_il_12m           <int> 1, 0, 2, 0, 0, 0, 2, 3, 0, 0, 1, 2, 0, 0, 0, 0, 0, 0, 2, 0, 0, 1, 1, 0, 1, 2, 0, 2, 1, 0, 1,...
$ inq_last_12m          <int> 0, 0, 2, 1, 3, 0, 9, 7, 0, 1, 8, 4, 2, 2, 1, 0, 0, 0, 9, 1, 0, 0, 0, 1, 7, 0, 4, 0, 0, 0, 3, 0,...
$ num_tl_op_past_12m    <int> 1, 0, 5, 2, 1, 0, 3, 4, 0, 0, 3, 3, 0, 0, 3, 1, 1, 0, 4, 2, 0, 2, 1, 0, 2, 4, 1, 3, 1, 2, 3,...
$ title                 <chr> "Medical expenses", "Debt consolidation", "Home buying", "Car financing", "Credit card refin...
$ open_il_24m           <int> 1, 2, 2, 0, 1, 2, 2, 4, 0, 1, 2, 4, 2, 1, 1, 0, 0, 1, 4, 1, 0, 1, 2, 0, 3, 2, 1, 2, 1, 1, 3,...
$ il_util               <int> 45, 87, 75, NA, 54, 73, 69, 72, 34, 103, 41, 102, 55, 58, 76, NA, NA, 69, 93, 46, NA, 36, 84...
$ open_rv_12m           <int> 0, 0, 3, 2, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 3, 1, 1, 0, 2, 2, 0, 1, 0, 0, 1, 1, 1, 1, 0, 2, 2,...
$ acc_open_past_24mths  <int> 2, 2, 7, 2, 4, 2, 4, 6, 1, 3, 5, 5, 5, 2, 4, 1, 3, 1, 7, 4, 3, 2, 2, 1, 7, 5, 3, 3, 2, 3, 6,...
$ all_util              <int> 29, 80, 59, 0, 24, 69, 61, 64, 43, 95, 65, 66, 33, 46, 46, 56, 46, 43, 90, 42, 41, 64, 82, 4...
$ inq_fi                <int> 0, 0, 1, 0, 2, 0, 11, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 4, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 1, 0...
$ total_rev_hi_lim      <int> 42900, 14600, 25500, 51900, 113000, 33000, 9100, 28000, 21600, 17200, 38000, 28000, 30100, 6...
$ mo_sin_rcnt_tl        <int> 7, 14, 1, 5, 11, 17, 1, 4, 22, 17, 2, 5, 14, 14, 4, 9, 10, 18, 4, 6, 18, 5, 8, 17, 4, 1, 5, ...
$ int_rate              <chr> "5.32%", "16.02%", "11.99%", "6.08%", "5.32%", "11.99%", "16.02%", "9.93%", "7.35%", "10.42%...
$ total_rec_prncp       <dbl> 247.11, 353.67, 1521.39, 509.13, 463.30, 861.63, 663.14, 129.93, 419.83, 1079.51, 80.78, 59....
$ bc_open_to_buy        <int> 27362, 296, 13467, 47199, 58568, 11974, 8400, 13826, 1813, 2039, 5320, 23136, 15707, 9483, 2...
$ mths_since_recent_inq <int> 17, 14, 1, 5, 12, NA, 2, 3, NA, 11, 2, 1, 5, 1, 4, NA, NA, 18, 2, 6, NA, NA, 8, 8, 13, 1, 16...
$ last_pymnt_d          <chr> "Mar-2018", "Mar-2018", "Mar-2018", "Mar-2018", "Mar-2018", "Mar-2018", "Mar-2018", "Mar-201...
>
```

Clasificación

En este apartado realizaremos un estudio de clasificación “no balanceada” con 3 modelos de predicción: “R-part”, “Random forest” y “Redes Neuronales”.

R-part

Es un algoritmo de partición recursiva para clasificación, árboles de regresión y supervivencia en este algoritmo, se define el parámetro de complejidad “.cp”, obteniendo un árbol óptimo en función de este parámetro.

```
## 1. Crear modelo de prediccion usando rpart
trainIndex <- createDataPartition(data$loan_status, p = .82, list = FALSE, times = 1)
train <- data[ trainIndex, ]
val   <- data[-trainIndex, ]

# Entrenar modelo
rpartCtrl <- trainControl(verboseIter = F, classProbs = TRUE, summaryFunction = twoClassSummary)
rpartParametersGrid <- expand.grid(.cp = c(0.001, 0.01, 0.1, 0.5))
rpartModel <- train(loan_status ~ ., data = train, method = "rpart", metric = "ROC",
                    trControl = rpartCtrl, tuneGrid = rpartParametersGrid)
```

La curva Roc que se ha obtenido en este modelo lo podemos apreciar en la siguiente ilustración:

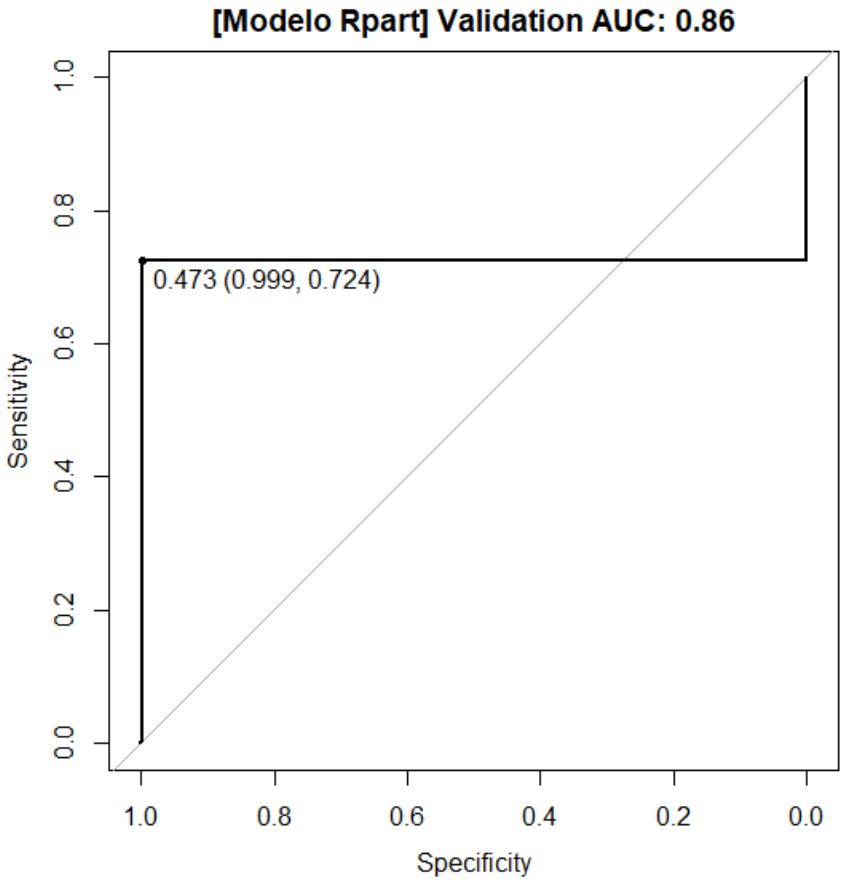


Ilustración 10: Modelo R-part

Las medidas de validación que se obtuvo.

```
> table(results$contingency)

   FN    FP    TN    TP
   78    14 14744   205
> f_measure
[1] 0.8167331
```

values	
accuracy	0.993883385413204
error	0.00611661458679609
f_measure	0.816733067729084
FN	78L
FP	14L
n	15041L
precision	0.936073059360731
prediction	Factor w/ 2 levels "Paid","Unpaid": 1 1 1 1 1 1…
sensitivity	0.724381625441696
specificity	0.999051361973167
TN	14744L
TP	205L

Se realizó unas pruebas con el modelo de R-part “Cross-Validation”, obteniendo resultados similares al modelo anterior.

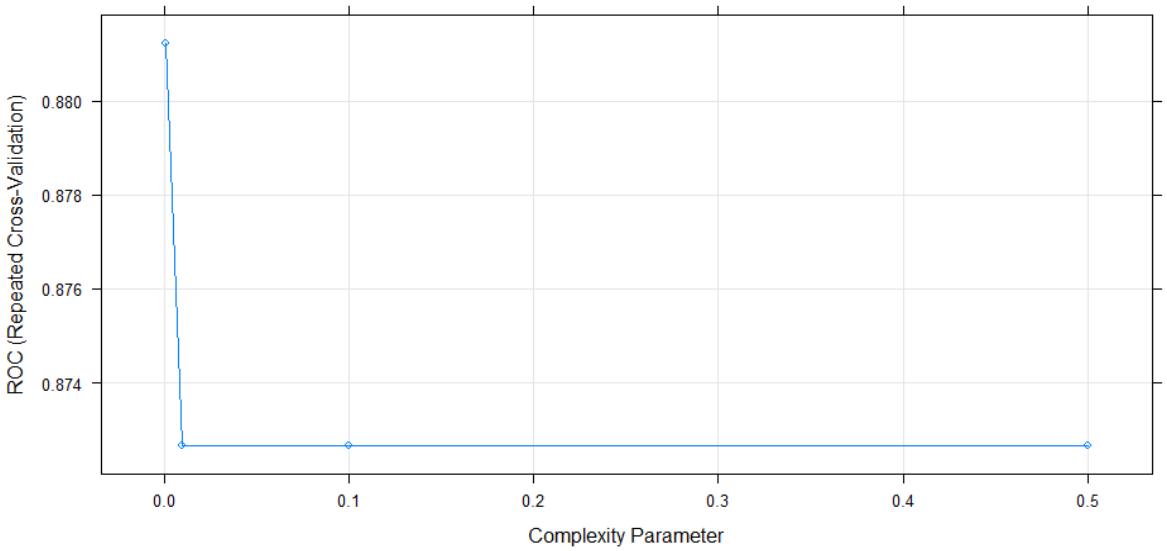


Ilustración 11: Modelo R-part “Cross-Validation”

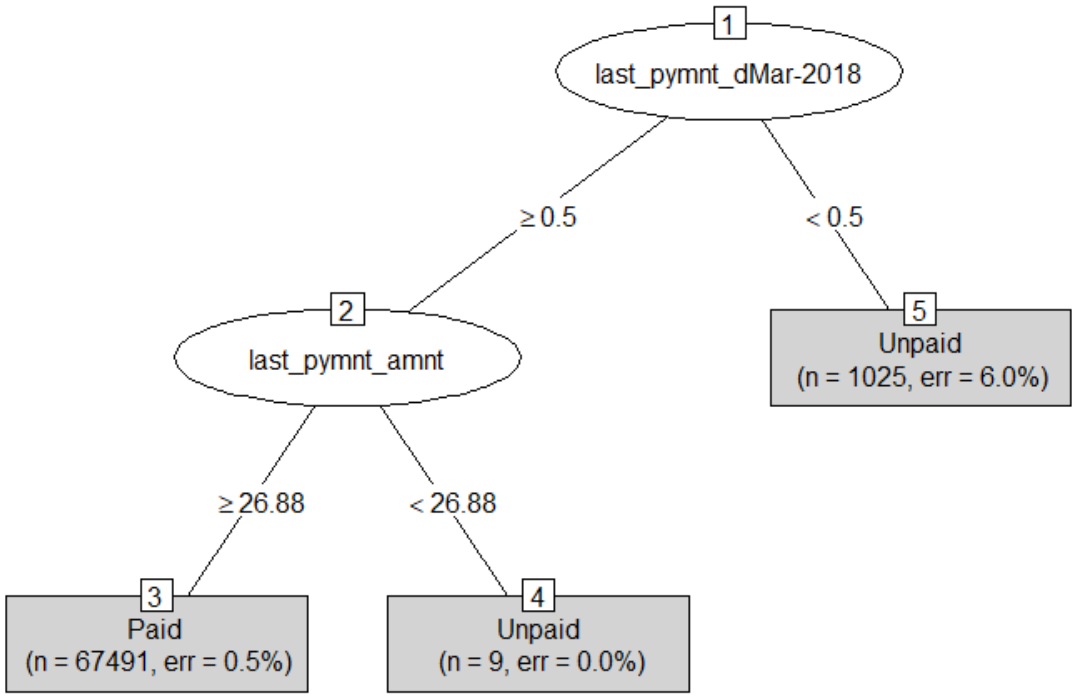


Ilustración 12: Árbol Modelo R-part “Cross-Validation”

Random Forest

Es un modelo clasificación y regresión basadas en un bosque de árboles usando entradas aleatorias.

Utiliza el parámetro ".mtry", este indica el número de variables aleatoriamente muestreados como candidatos en cada división.

```
## Crear modelo de prediccionn usando RF
### Modelo basico, ajuste de manual de hiperparametros (.mtry)
rfCtrl <- trainControl(verboseIter = F, classProbs = TRUE, method = "repeatedcv",
                       number = 10, repeats = 1, summaryFunction = twoClassSummary)
rfParametersGrid <- expand.grid(.mtry = c(sqrt(ncol(train))))
rfModel <- train(loan_status ~ ., data = train, method = "rf", metric = "ROC",
                 trControl = rfCtrl,tuneGrid = rfParametersGrid)
my_roc(val, predict(rfModel, val, type = "prob"), "loan_status", "Unpaid")
```

Ejecutamos el modelo de predicción básico con ajuste de manual de hiperparametros (.mtry), obteniendo el siguiente resultado que se puede apreciar en la siguiente ilustración.

La curva Roc que se ha obtenido en este modelo lo podemos apreciar en la siguiente ilustración:

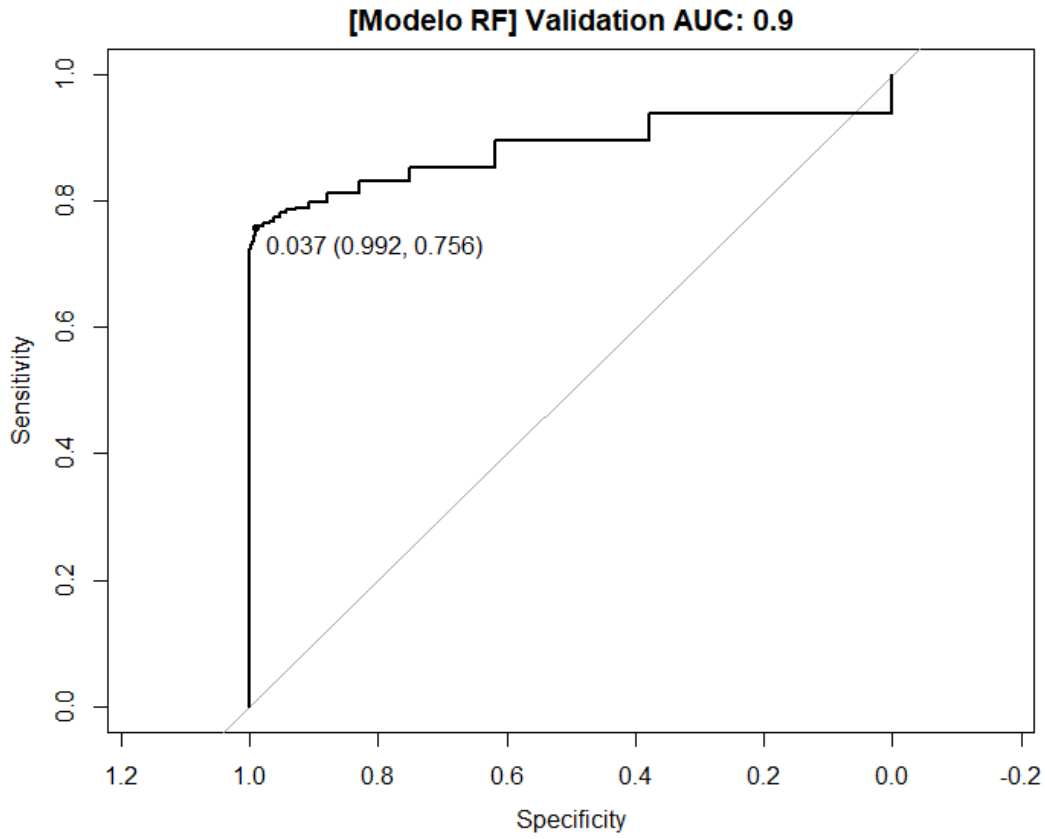


Ilustración 13: Modelo Random Forest

Las medidas de validación que se obtuvo.

```
> table(results$contingency)

   FN    FP    TN    TP
   79    14 14744   204
> f_measure
[1] 0.8143713
```

values	
accuracy	0.993816900472043
error	0.00618309952795692
f_measure	0.81437125748503
FN	79L
FP	14L
n	15041L
precision	0.935779816513762
prediction	Factor w/ 2 levels "Paid","Unpaid": 1 1 1 1 1 1 1 1 1 1 .
sensitivity	0.720848056537102
specificity	0.999051361973167
TN	14744L
TP	204L

Redes Neuronales

Para el realizar este modelo de predicción utilizamos el método de "nnet".

Lo cual me permite modificar los siguientes parámetros y así obtener un mejor resultado.

- .decay : es el parámetro de regularización para evitar el ajuste excesivo.
- .size : Es el número de unidades en la capa oculta. (20, 2000, 2)
- maxit: Es el número de iteraciones máximas. (50000)

Se ha ajustado este modelo hasta conseguir un resultado promedio.

He decidido quedarme con estos valores, ya que se realizó varias pruebas y con estos obtuve un mejor resultado.

```
## 3.Modelo de Prediccion RN
##  Crear modelo de prediccion usando RN
nnCtrl <- trainControl(verboseIter = F, classProbs = TRUE, method = "repeatedcv",
                       number = 10, repeats = 1, summaryFunction = twoClassSummary)
nnParametersGrid <- expand.grid(.decay = c(0.5, 0.1), .size = c(20, 2000, 2))
nnModel <- train(loan_status ~ ., data = train, method = "nnet", metric = "ROC",
                 tuneGrid = nnParametersGrid, trControl = nnCtrl, trace = FALSE, maxit = 50000)
my_roc(val, predict(nnModel, val, type = "prob"), "loan_status", "Unpaid")
```

La curva Roc que se ha obtenido en este modelo lo podemos apreciar en la siguiente ilustración:

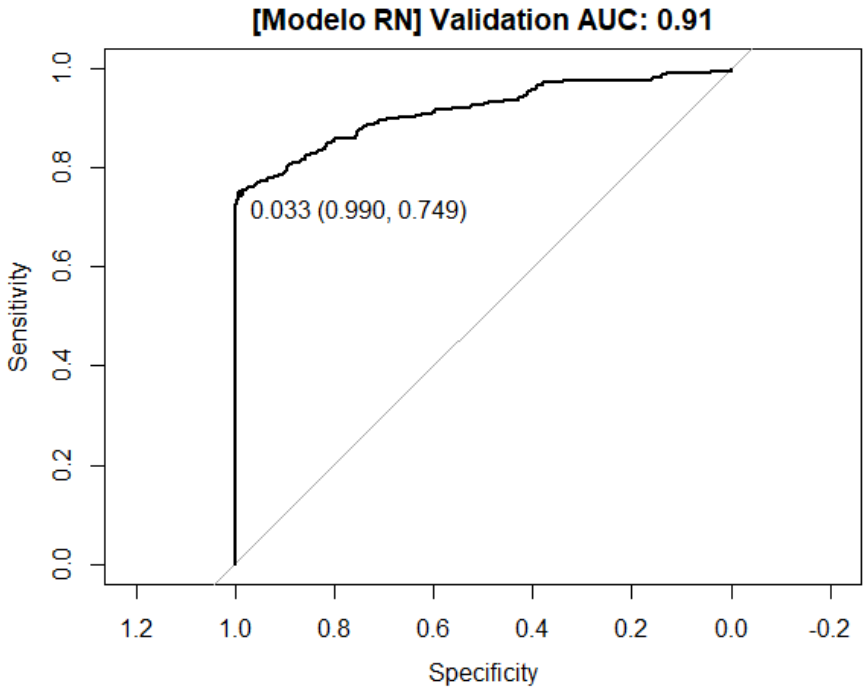


Ilustración 14: Modelo Redes Neuronales

Las medidas de validación que se obtuvo.

```
> table(results$contingency)

   FN    FP    TN    TP
   80    13 14745   203
> f_measure
[1] 0.8136273
```

values	
accuracy	0.993816900472043
error	0.00618309952795692
f_measure	0.813627254509018
FN	80L
FP	13L
n	15041L
precision	0.939814814814815
prediction	Factor w/ 2 levels "Paid","Unpaid": 1 1 1 1 1 1 1
sensitivity	0.717314487632509
specificity	0.999119121832227
TN	14745L
TP	203L

Discusión de resultados

Procederemos a realizar comparativas respecto a los análisis que se han realizado con los diferentes modelos de clasificación.

Obteniendo los siguientes detalles de los resultados de curva Roc, f_measure y contingencias.

- Roc

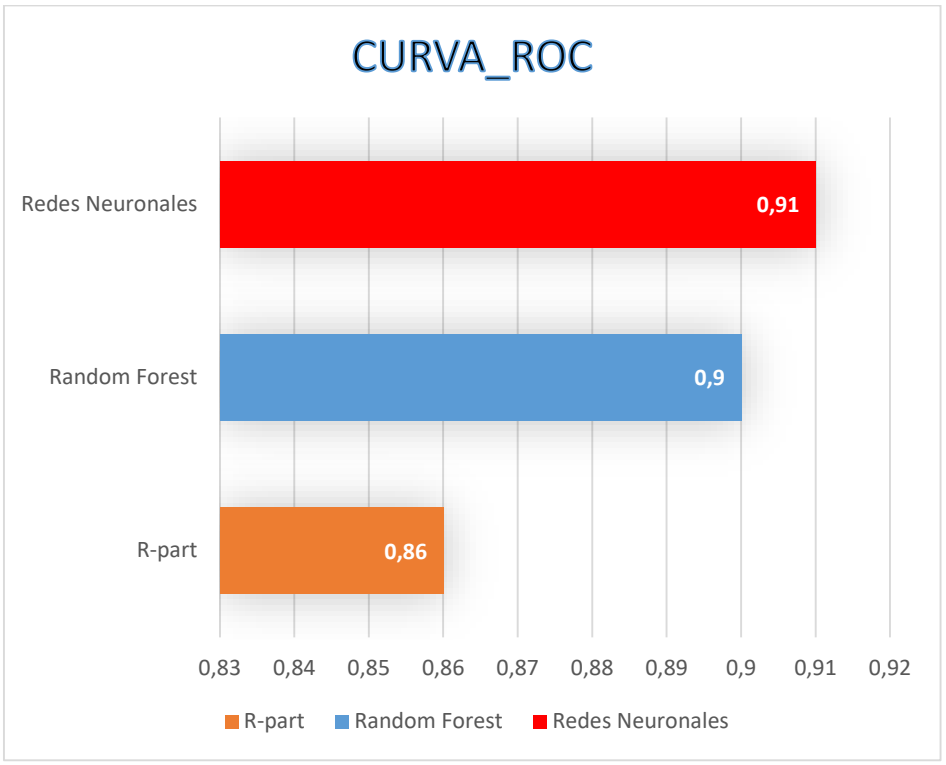


Ilustración 15: Curva Roc

El modelo de “Redes Neuronales” tiene el mejor porcentaje.

- F_measure

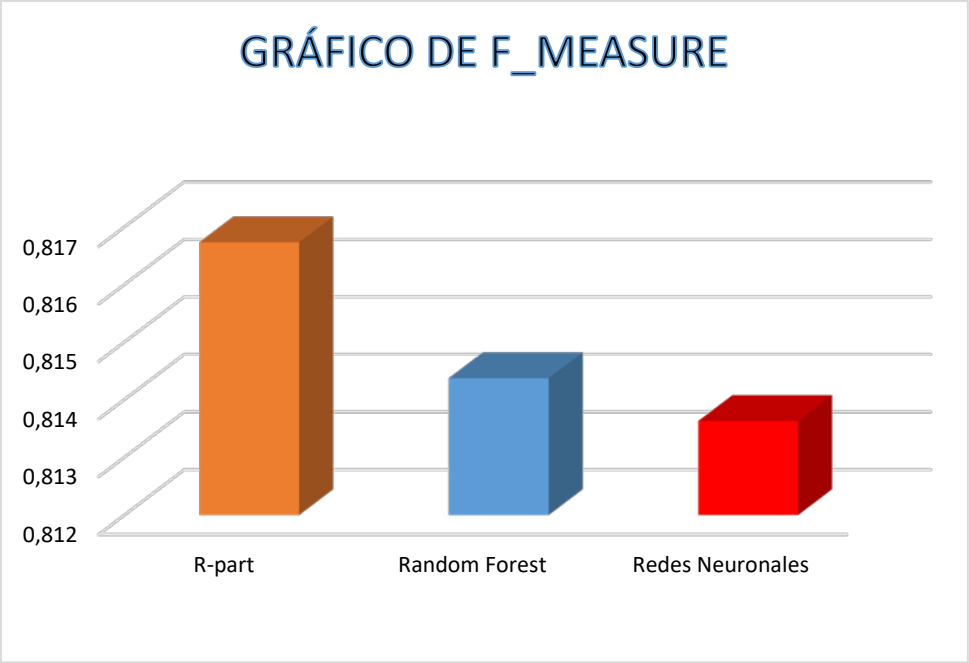


Ilustración 16: f_measure

Procedemos a medir la calidad de predicción de los métodos y en este caso el de mejor calidad fue el que nos dio R-part

- Contingencias

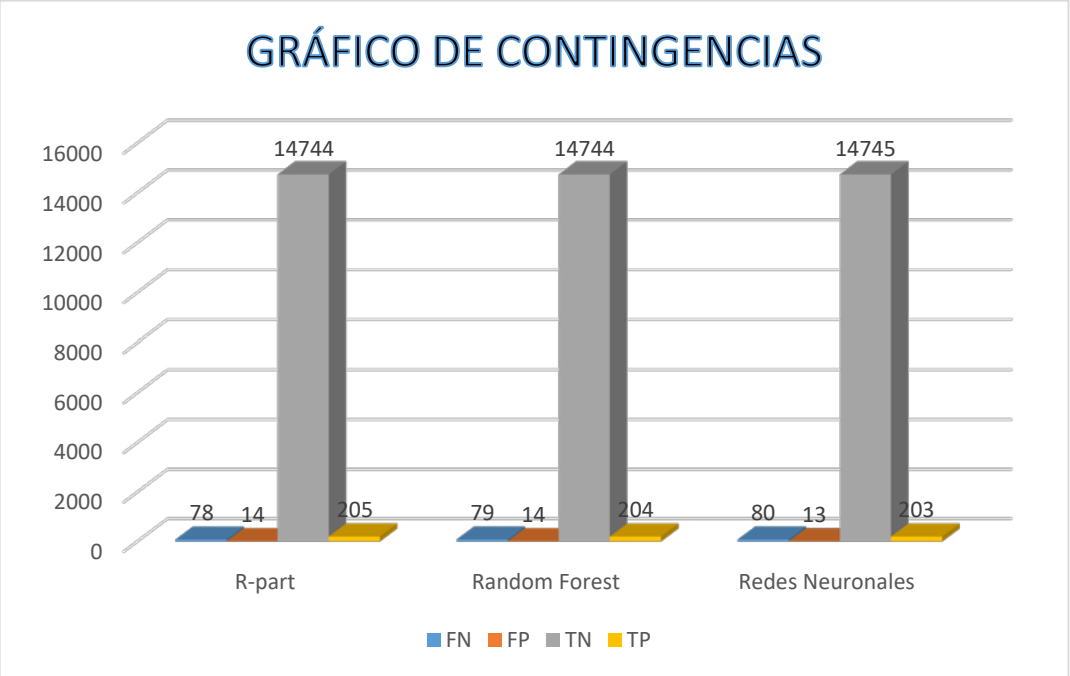


Ilustración 17: Contingencias entre modelos

Podemos observar que los modelos tienen resultados muy parecidos entre sí, pero el que menor falsos negativos tiene es el de Redes Neuronales.

Conclusiones

Se ha procedido a realizar varios estudios con diferentes modelos de predicción con el objetivo de conseguir el mejor modelo para este caso y según los resultados el modelo que está en la media prediciendo con más fiabilidad es el “Random Forest” con una curva Roc de 0.9 una calidad de predicción de 0,8143713.

Se hubiese podido conseguir mejores resultados y hubiésemos tenido un equilibro en las clases.

Este desequilibrio nos genera un problema al momento de realizar el modelo de predicción. Por ende, deberíamos realizar un procesamiento adicional de balanceo.

Bibliografía

- https://www.rdocumentation.org/
- https://rpubs.com/medusa/128016
- http://apuntes-r.blogspot.com.es/
- https://stackoverflow.com/
- https://github.com/jgromero/sige2018