

Creación de un Servicio RESTful para el Modelo de Crédito con Aprendizaje Automático, en R

Maria Alejandra Quevedo Vera

¹Universidad Distrital Francisco José de Caldas, RITA

maquevedov@correo.udistrital.edu.co

Abstract. *En este documento se ilustrará paso a paso el proceso que se siguió para obtener un servicio RESTful utilizando RStudio. Para esto, se utilizó una base de datos que contenía información de algunos ciudadanos alemanes, relevante para las entidades bancarias en el momento de realizar un estudio de crédito, tras lo cual se crea un árbol de predicción y se accede a esta información a través de la WEB utilizando POSTMAN.*

1. Introducción

La Transferencia de Estado Representacional o REST por sus siglas en inglés (Representational State Transfer) puede entenderse como una interfaz sencilla entre sistemas que usa HTTP¹ para obtener datos o generar operaciones sobre esos datos.

REST se basa en el concepto de "sin estado", en un protocolo de comunicaciones fácil de usar que en la mayoría de los casos es HTTP, y es un tipo de arquitectura que se emplea para diseñar aplicaciones en la red en donde HTTP es el principal protagonista para hacer los llamados entre máquinas. El ejemplo más sencillo es justamente el de www (World Wide Web) que está basado en HTTP y que puede verse como una arquitectura REST. Las aplicaciones RESTful utilizan las ya conocidas solicitudes de HTTP: CREAR, ACTUALIZAR, LEER, ELIMINAR, o como se conoce comúnmente CRUD (Create, Read, Update, Delete). Y son estos conceptos los que utilizaremos en este experimento.

Para el desarrollo de este proyecto se ha utilizado la propuesta de Knowru [?] en donde describen paso a paso el procedimiento para construir un servicio REST con R después de hacer machine learning a una base de datos desde una URL. En la primera sección se describirá la base de datos utilizada, a continuación se explicará el procedimiento junto con la descripción del código, se observarán algunas pruebas y finalmente se analizarán los resultados obtenidos.

2. La Base de Datos

La Base de datos se tomó del Repositorio de la Universidad Irvine de California (UCI). La base de datos se llama *Statlog*² (*German Credit Data*) *Data Set*, y está compuesta por 1.000 instancias, con 7 atributos numéricos y 13 categóricos o cualitativos.

¹Hypertext Transfer Protocol

²Statistical and Logical Learning Algorithm

2.1. Descripción de los Atributos

A continuación se enlistarán los atributos de la base de datos en donde aparecerá el nombre del atributo, su tipo y su función. La descripción detallada de los atributos junto con sus etiquetas, se encuentra en el .

Atributo 1: (cualitativo): Estado de cuenta corriente existente

Atributo 2: (numérico): Duración en meses

Atributo 3: (qualitative): Historial Crediticio

Atributo 4: (cualitativo): Propósito

Atributo 5: (numérico).Monto del Crédito.

Atributo 6: (cuallitativo): Cuenta de ahorros/bonos

Atributo 7: (cualitativo): Empleo actual desde

Atributo 8: (numérico): Tasa de pago en porcentaje del ingreso disponible

Atributo 9: (cualitativo): Estado civil/sexo

Atributo 10: (cualitativo): Otros deudres/garantes

Atributo 11: (numérico): Residencia actual desde:

Atributo 12: (cualitativo): Propiedad

Atributo 13: (numérico): Edad en años

Atributo 14: (cualitativo): Otros planes de pago

Atributo 15: (cualitativo): Vivienda

Atributo 16: (numerica): Número de crditos existentes en este banco

Atributo 17: (cualitativo):Trabajo

Atributo 18: (numérico): Número de personas a cargo

Atributo 19: (cualitativo): Teléfono

Atributo 20: (cualitativo): Trabajador Extranjero

En la siguiente figura se muestran los datos cargados en RStudio.

	Status.of.existing.checking.account	Duration.in.month	Credit.history	Purpose	Credit.amount	Savings.account.bonds	Employe
1	A11	6	A34	A43	1169	A65	A75
2	A12	48	A32	A43	5951	A61	A73
3	A14	12	A34	A46	2096	A61	A74
4	A11	42	A32	A42	7882	A61	A74
5	A11	24	A33	A40	4870	A61	A73
6	A14	36	A32	A46	9055	A65	A73
7	A14	24	A32	A42	2835	A63	A75
8	A12	36	A32	A41	6948	A61	A73
9	A14	12	A32	A43	3059	A64	A74
10	A12	30	A34	A40	5234	A61	A71
11	A12	12	A32	A40	1295	A61	A72
12	A11	48	A32	A49	4308	A61	A72
13	A12	12	A32	A43	1567	A61	A73
14	A11	24	A34	A40	1199	A61	A75
15	A11	12	A32	A40	1169	A61	A73

Showing 1 to 15 of 1,000 entries

Figura 1. Conjunto de Datos

3. Desarrollo

En esta sección se describirá paso a paso el modelo empleado para saber si una persona va a pagar un crédito utilizando la base de datos de crédito alemán.

3.1. Modelo de Aprendizaje Automático

En la Figura 2 se puede observar la primera parte del código en donde se carga la información desde una url, esta url contiene los datos en formato csv, para este caso separados por espacios, por lo cual, al hacerse el llamado de la url se obtiene la base de datos pura.

```
1 url <- "https://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german/german.data"
2 col.names <- c(
3   'Status of existing checking account', 'Duration in month', 'Credit history'
4   , 'Purpose', 'Credit amount', 'Savings account/bonds'
5   , 'Employment years', 'Installment rate in percentage of disposable income'
6   , 'Personal status and sex', 'Other debtors / guarantors', 'Present residence since'
7   , 'Property', 'Age in years', 'Other installment plans', 'Housing', 'Number of existing credits at this bank'
8   , 'Job', 'Number of people being liable to provide maintenance for', 'Telephone', 'Foreign worker', 'Status'
9 )
10 # Obtención de Datos
11 data <- read.csv(
12   url
13   , header=FALSE
14   , sep=' '
15   , col.names=col.names
16 )
17
```

Figura 2. Acceso a la base de datos desde url

A continuación se crea *col.names* en donde serán almacenados los nombres de las columnas de la base de datos; con este proceso será más sencillo manipular los atributos por separado. Aquí los nombres corresponden a la descripción anteriormente ilustrada en la sección de "Descripción de Atributos."

Después de tener la base de datos almacenada en una variable llamada *url* y de tener los nombres de las columnas almacenados en otra variable llamada *col.names*, se procede a cruzar esta información para poder tener una sola matriz con la información y sus respectivas referencias como se puede ver en las líneas de la 11 a la 16 de la Figura 2, en donde se lee el archivo csv u se asigna como nombres los almacenados en *col.names*.

En la Figura 3 se puede observar el proceso para la predicción utilizando una librería llamada *rpart*³ que se utiliza para clasificar datos o hacer regresiones y los resultados pueden visualizarse en árboles de decisión. En este punto, se cargan los nombres de las columnas que se utilizarán en la regresión. Se hizo de esta manera debido a que en experimentos anteriores se encontró que con los 4 atributos utilizados (Atributo 1, Atributo 2, Atributo 3 y Atributo 6) era más que suficiente para obtener la discriminación deseada.

```
18 library(rpart)
19 # Construcción del árbol de decisión
20 decision.tree <- rpart(
21   Status ~ Status.of.existing.checking.account + Duration.in.month + Credit.history + Savings.account.bonds
22   , method="class"
23   , data=data
24 )
```

Figura 3. Utilización de la librería *rpart* y construcción del árbol de decisión

³Recursive Partitioning for Classification

Después de esto se visualizó la información en el árbol de decisión y para esto fue necesario utilizar una nueva librería *rpartplot* con la que era posible utilizar la información del proceso anterior y presentar los resultados de forma gráfica como se muestra en la Figura 4. Para la construcción de esta gráfica se utilizaron los datos que coincidieran con las siguientes características:

Atributo 1 (Estado de cuenta corriente existente)= A11

Atributo 2 (Duración en meses)= 20

Atributo 3 (Historial Crediticio)= A32

Atributo 6 (Cuenta de ahorros/bonos)= A65

```

25 install.packages("rpart.plot")
26 library(rpart.plot)
27 # Visualize the tree
28 # 1 is good, 2 is bad
29 prp(
30   decision.tree
31   , extra=1
32   , varlen=0
33   , faclen=0
34   , main="Decision Tree for German Credit Data"
35 )
36 new.data <- list(
37   Status.of.existing.checking.account='A11'
38   , Duration.in.month=20
39   , Credit.history='A32'
40   , Savings.account.bonds='A65'
41 )
42 predict(decision.tree, new.data)

```

Figura 4. Paquete *rpart.plot* para presentación gráfica del proeso anterior.

Es necesario tener en cuenta que la instalación de los paquetes para la utilización de las librerías es un proceso que se realiza solo una vez. No es necesario instalarlo cada vez que se corre el código. De hecho, es posible verificar si un paquete ha sido instalado correctamente si aparece marcado en la sección de "paquetes." *Package* que aparece en la ventana de RStudio como se ilustra en la Figura 5.

Files	Plots	Packages	Help	Viewer
<div> <div>Install</div> <div>Update</div> <div> <input type="text" value="rpa"/> </div> </div>				
Name		Description	Version	
<input checked="" type="checkbox"/>	rpart	Recursive Partitioning and Regression Trees	4.1-13	⊗
<input checked="" type="checkbox"/>	rpart.plot	Plot 'rpart' Models: An Enhanced Version of 'plot.rpart'	2.1.2	⊗

Figura 5. Verificación de la instalación de los paquetes que serán utilizados.

El árbol de decición obtenido es el que se muestra en la Figura 6. Para entender esta figura se debe tener en cuenta que una respuesta positiva se va a representar con "1" una respuesta negativa se va a representar con un "2", como se puede ver en las cajas del árbol.

El primer atributo correspondiente al Estado de Cuenta Corriente Existente ubicado en primer lugar en la parte superior del árbol de decisión, de éste se derivan dos posibilidades: *la primera*, que si exista una cuenta, con lo cual se aprueba el crédito inmediatamente, *la segunda*, que no exista una cuenta con lo cual se procede a evaluar el siguiente atributo, y así sucesivamente hasta llegar a la parte baja del árbol de decisión en donde se puede ver que 2 opciones indicarían una negación del crédito con una respuesta de 2 y las otras opciones indicarían un crédito aprobado con una respuesta de 1.

La referencia que aparece a la derecha de los nombres de cada atributo corresponde a las claves que resultarían en un resultado negativo. Por ejemplo, *Historial Crediticio* = A32, A33, A34, según las claves descritas en el ANEXO 1, corresponden a crédito reembolsado correctamente hasta ahora, retraso en pagos en el pasado y cuenta crítica o sin cuentas en este banco, respectivamente, lo que ocasionaría una negación del crédito.

Si un cliente no tiene una cantidad marginal de dinero en su cuenta corriente o no tiene una cuenta corriente, es probable que esté al día (1 significa bueno, 2 significa malo). Si un cliente tiene solo una pequeña cantidad de dinero en su cuenta corriente, la duración del préstamo es mayor o igual a 22, y también tiene una cantidad insignificante de fondos en su cuenta de ahorros, es más que probable que incumpla por lo cual tendrá una calificación negativa (2).

En la parte baja del árbol en donde se encuentra el historial crediticio y las cuentas de ahorro, se puede observar que los nodos de la izquierda de cada uno resutó positivo, esto significa que algunos clientes tuvieron retrasos en el pago en el pasado o créditos en otros bancos, pero se clasificarán como buenos en este árbol de decisiones.

Este modelo permitiría a los usuarios que bien podrían ser bancos, realizar una consulta rápida y sencilla sobre la confiabilidad para abrir un crédito con un nuevo cliente. Por facilidad en este ejercicio sólo se han tomado en cuenta 4 parámetros de los 21 disponibles para complementar el modelo.

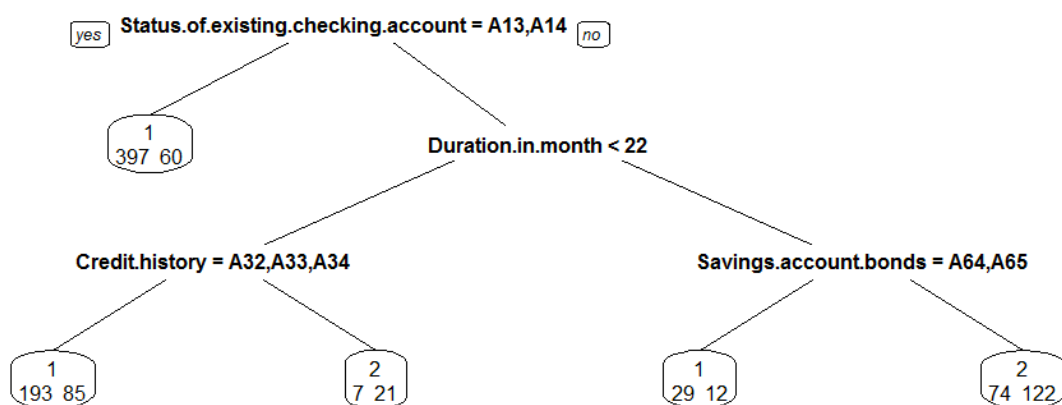


Figura 6. Árbol de decisión.

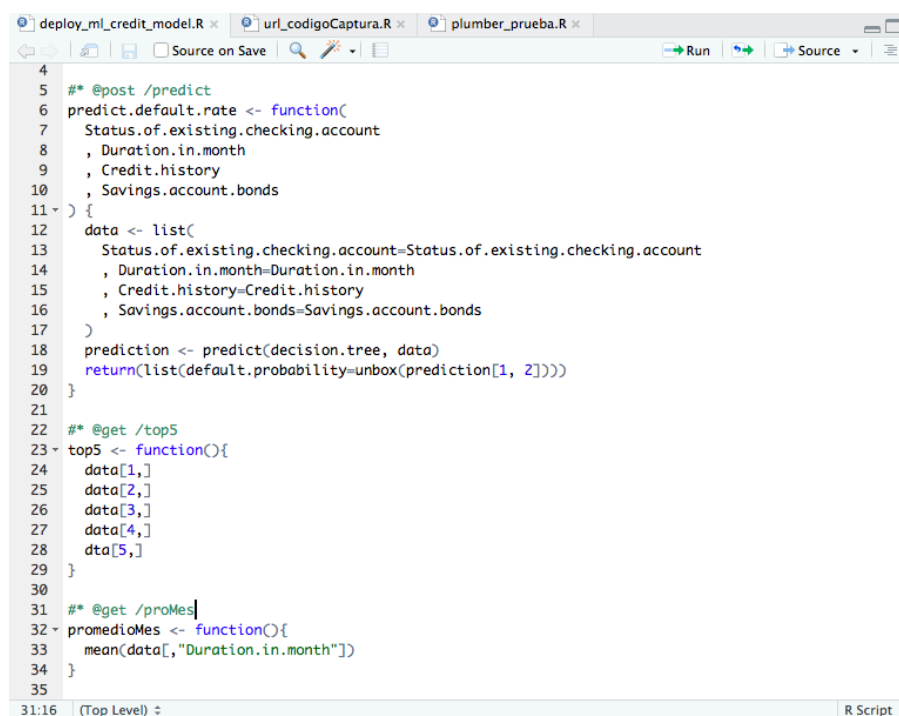
A continuación se creará una API RESTful para el modelo de credito implementado, permitiendo que cualquier entidad bancaria pueda utilizarla. En este caso se utiliza la librería *jsonlite* que permite implementar un mapeo bidireccional entre los datos de R y los de JSON, permitiendo convertir objetos R y JSON sin pérdida de información. Con

este paquete los datos pueden fluir sin interrupción dentro y fuera de R utilizando JSON. En el mismo script de código es posible implementar varios métodos, sin embargo es necesario tener en cuenta que cada uno debe tener su encabezado indicando la función y su nombre propio. En la Figura 7 se puede observar el proceso que se utilizó para la creación de la API.

```
43 library(rpart)
44 library(jsonlite)
45 load("decision_Tree_for_german_credit_data.RData")
46
47 ## @post /predict
48 predict.default.rate <- function(
49   Status.of.existing.checking.account
50   , Duration.in.month
51   , Credit.history
52   , Savings.account.bonds
53 ) {
54   data <- list(
55     Status.of.existing.checking.account=Status.of.existing.checking.account
56     , Duration.in.month=Duration.in.month
57     , Credit.history=Credit.history
58     , Savings.account.bonds=Savings.account.bonds
59   )
60   prediction <- predict(decision.tree, data)
61   return(list(default.probability=unbox(prediction[1, 2])))
62 }
```

Figura 7. Creación de API en R.

Aquí es posible observar cómo se implementa el POST que se va a llamar más adelante. Esta función hace uso de 4 parámetros (columnas) para poder utilizar el árbol de decisión y calcular la probabilidad de aprobación del crédito en cuestión. El nombre del POST que se ha creado para este caso es *predict* como se puede observar en la línea de código que aparece en verde. A continuación se crea la función, indicando entre paréntesis las variables de entrada, y en llaves, se coloca lo que se quiere que haga esta función.



```
4
5 ## @post /predict
6 predict.default.rate <- function(
7   Status.of.existing.checking.account
8   , Duration.in.month
9   , Credit.history
10  , Savings.account.bonds
11 ) {
12   data <- list(
13     Status.of.existing.checking.account=Status.of.existing.checking.account
14     , Duration.in.month=Duration.in.month
15     , Credit.history=Credit.history
16     , Savings.account.bonds=Savings.account.bonds
17   )
18   prediction <- predict(decision.tree, data)
19   return(list(default.probability=unbox(prediction[1, 2])))
20 }
21
22 ## @get /top5
23 top5 <- function(){
24   data[1,]
25   data[2,]
26   data[3,]
27   data[4,]
28   data[5,]
29 }
30
31 ## @get /proMes
32 promedioMes <- function(){
33   mean(data[, "Duration.in.month"])
34 }
35
31:16 (Top Level) R Script
```

Figura 8. Creación de GET y POST

La diferencia entre la implementación de un POST y un GET, radica en los parámetros de entrada; mientras que para un GET no es necesario poner ningún parámetro de entrada debido a que es una consulta, para un POST si es necesario utilizar los parámetro de entrada los cuales van a alimentar la función en cuestión. En la Figura 8 Se muestran 3 servicios: **1.** POST: "predict" que de acuerdo al valor de los parámetros de entrada, arrojará la predicción producto de hacer uso del modelo de machine learning. **2.** GET: "top5" que muestra las 5 primeras filas del set de datos, y finalmente **3.** GET: "prom-Mes" que arrojará el promedio de la columna de Duración en meses (Duration.in.month) del set de datos.

Una es creados los servicios, se utilizará la librería **plumber**, que es un paquete que permite crear fácilmente una API RESTful para programas escritos en R. Es necesario que la creación de los métodos con JSON se guarden en un archivo aparte, para poder llamar este archivo con plumber y poder crear la API como se indica en la Figura 9. En la primera línea se carga la librería, la segunda línea crea la API, y la tercera línea abre el puerto por el que se va a acceder a los servicios. Para este caso, el puerto es el 8000 en el local host.

```
74 library(plumber)
75 r <- plumb("deploy_ml_credit_model.R")
76 r$run(port=8000)
77 |
78
```

Figura 9. Utilización de la librería Plumber.

Cuando se corre el código descrito en la figura anterior, aparece en la consola que el servidor ha empezado a escuchar a través del puerto 8000 como se le había indicado. Esto se puede ver en la Figura 10.

```
Console Terminal x
~/
> library(plumber)
> r <- plumb("deploy_ml_credit_model.R")
> r$run(port=8000)
Starting server to listen on port 8000
Running the swagger UI at http://127.0.0.1:8000/___swagger___/
```

Figura 10. Plumber.

Se realiza entonces un experimento para comprobar el funcionamiento de esta API. Se abre la terminal y se hace una solicitud a través de un post utilizando el puerto 8000 con la siguiente información de un cliente:

Estado de Cuenta Corriente: A11

Duración en meses: 24

Historial Crediticio: A32

Cuenta de Ahorros/bonos: A63


```
curl -X POST -d '{"Status.of.existing.checking.account": "A11", "Duration.in.month": 24, "Credit.history": "A32",  
"Savings.account.bonds": "A63"}' -H 'Content-Type: application/json' localhost:8000/predict
```

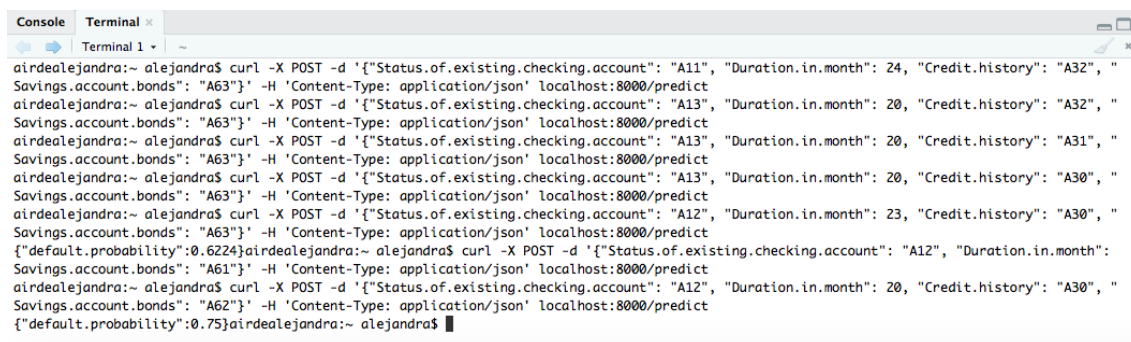
Para esto se abre la terminal y se ejecuta el siguiente código:

Aquí, curl -X POST corresponde a una solicitud ,y -H 'Content-Type: application/json' a una aplicación de tipo JSON. En la Figura , se muestra la respuesta obtenida indicando que el moelo ha respondido efectivamente.

```
host:8000/predicthistory": "A32", "Savings.account.bonds": "A63"}' -H 'Content-Type: application/json' local  
{ "default.probability": 0.6224 }airdealejandra:~ alejandra$
```

Figura 11. Solicitud realizada y obtención de respuesta

Como se puede observar la respuesta obtenida arroja una probabilidad de aprobación del crédito de 62,24 % para el cliente en cuestión. En la Figura 12, se realizaron otros experimentos para comprobar la eficacia del modelo. En donde se obtuvieron porcentajes diferentes dado que se tenían claves diferentes. Para este ejercicio el mayo porcentaje obtenido fue el del último cliente ingresado.



```
airdealejandra:~ alejandra$ curl -X POST -d '{"Status.of.existing.checking.account": "A11", "Duration.in.month": 24, "Credit.history": "A32", "Savings.account.bonds": "A63"}' -H 'Content-Type: application/json' localhost:8000/predict
airdealejandra:~ alejandra$ curl -X POST -d '{"Status.of.existing.checking.account": "A13", "Duration.in.month": 20, "Credit.history": "A32", "Savings.account.bonds": "A63"}' -H 'Content-Type: application/json' localhost:8000/predict
airdealejandra:~ alejandra$ curl -X POST -d '{"Status.of.existing.checking.account": "A13", "Duration.in.month": 20, "Credit.history": "A31", "Savings.account.bonds": "A63"}' -H 'Content-Type: application/json' localhost:8000/predict
airdealejandra:~ alejandra$ curl -X POST -d '{"Status.of.existing.checking.account": "A13", "Duration.in.month": 20, "Credit.history": "A30", "Savings.account.bonds": "A63"}' -H 'Content-Type: application/json' localhost:8000/predict
airdealejandra:~ alejandra$ curl -X POST -d '{"Status.of.existing.checking.account": "A12", "Duration.in.month": 23, "Credit.history": "A30", "Savings.account.bonds": "A63"}' -H 'Content-Type: application/json' localhost:8000/predict
airdealejandra:~ alejandra$ curl -X POST -d '{"Status.of.existing.checking.account": "A12", "Duration.in.month": 20, "Credit.history": "A30", "Savings.account.bonds": "A62"}' -H 'Content-Type: application/json' localhost:8000/predict
airdealejandra:~ alejandra$
```

Figura 12. Solicitudes co diferentes, simulando diferentes clientes

Para comprobar si este modelo respondía cuando se hacía la solicitud desde otro ambiente distinto a R, se utilizó **POSTMAN** que es una herramienta que cuenta con utilidades gratuitas que permiten realizar tareas diferentes dentro del mundo API REST como creación de peticiones internas o a terceros, realizar pruebas y validar el comportamiento de APIs entre otros.

Se utilizarán los datos de la ultima solicitud de experimento en donde se obtuvo una probabilidad del 75 %. La Figura 13, muestra la solicitud hecha desde otro ambiente a través del localhost:8000, y en la parte inferior se obtiene la misma respuesta que se había obtenido desde R. Como se puede observar el ambiente de POSTAMN implica no sólo un lenguaje diferente desde donde se realiza la petición sino que además implica un ambiente diferente. Aquí se puede comprobar que este pequeño experimento es transversal y que puede emplearse en las situaciones actuales reales, en donde cada Banco (en este caso) cuenta con su propio sistema pero todo pueden utilizar este modelo.

Se realiza otra petición simulando el primer cliente ingresado, y se obtienen los mismos resultados, comprobando dentonces que el modelo funciona y responde correctamente ante las diferentes solicitudes que se le hagan, como se muestra en la Figura 14

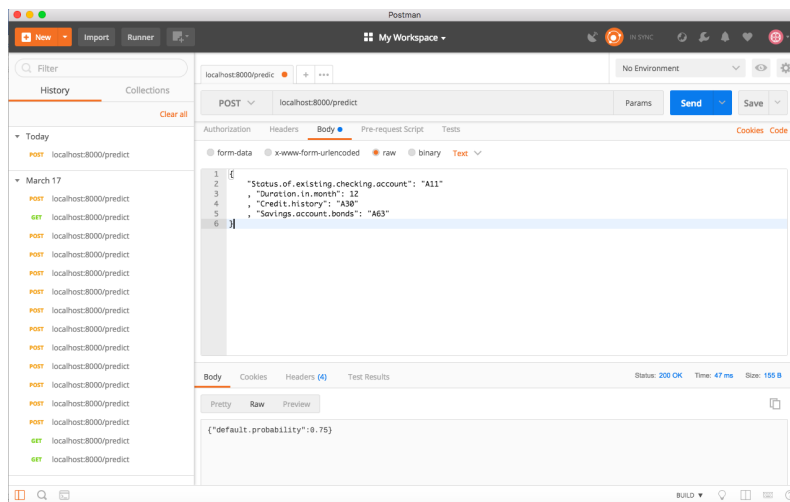


Figura 13. Solicitud desde POSTMAN

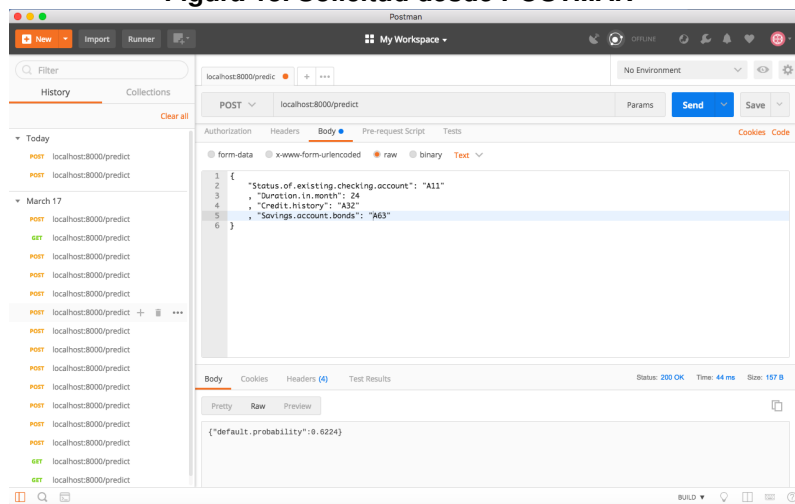


Figura 14. Solicitud desde POSTMAN

Figura 15. Caption place holder

4. Servicio en R Studio en los Servidores de RITA

Para lograr montar el servicio en los servidores de RITA⁴ fue necesario solicitar una IP pública por la cual fuera posible acceder al servicio, además se habilitaron los permisos necesarios para poder activar el puerto por el que se accedería a la API creada con *plumber* para poder utilizar los servicios. En la Figura 16 se puede observar la *url* en donde se encuentra el R studio que está ejecutando el programa.

Una gran diferencia con el servicio que se estaba creando de manera local, es la implementación de la API con *plumber*, ya que debido a las características propias de la RITA, por seguridad no era posible obtener todos los permisos para habilitar y deshabilitar los puertos. Sin embargo, esto sí era posible mediante un host que se asignó para poder llevar a cabo este experimento. En la Figura 17 se puede observar que para poder hacer uso de la API, se corre nuevamente abriendo el puerto 8000 pero con el host que se ha

⁴Red de Investigaciones de Tecnología Avanzada de la Universidad Distrital Francisco José de Caldas

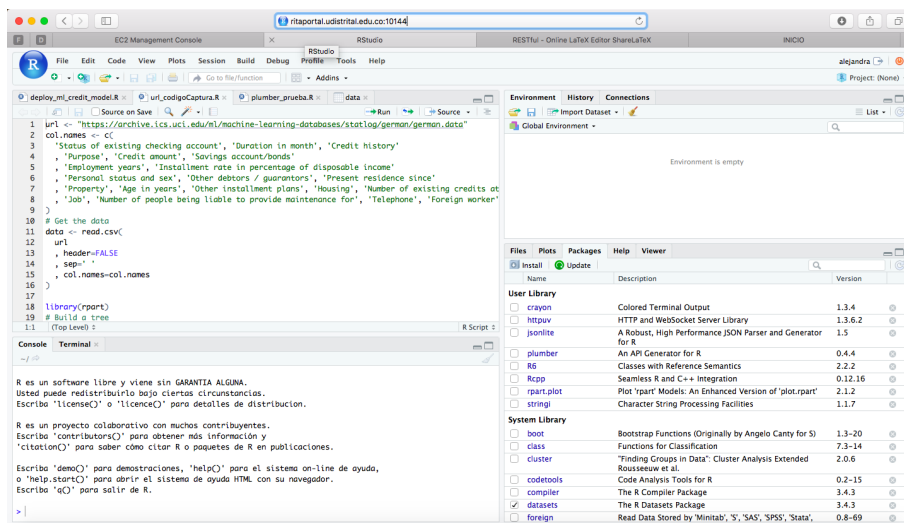


Figura 16. R Studio en los servidores de RITA

asignado.

Para comprobar que el servicio está en funcionamiento, se realizaron peticiones desde 2 ambientes diferentes: desde la misma terminal de R Studio y desde POSTMAN. Se utilizarán los mismos valores para los parámetros de entrada, con lo cual será posible evidenciar si hay diferencias en las respuestas o si se han obtenido las mismas respuestas para los mismos valores de entrada.

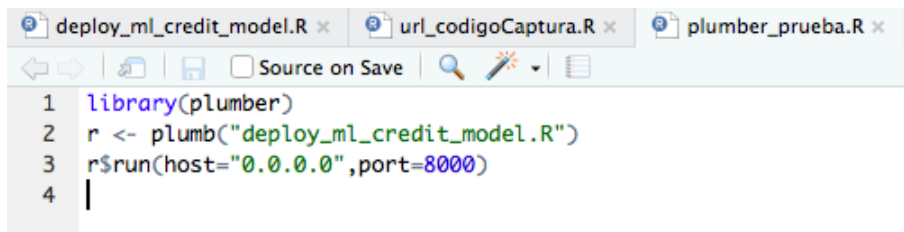


Figura 17. Implementación *plumber* en R Studio de los servidores de RITA

En la Figura 18 se muestran las tres peticiones desde la terminal de R Server. La petición del POST debe retornar el valor de la probabilidad según los datos ingresados que para este caso debía ser de 0.6224. La petición del GET *proMes* debe retornar el valor del promedi de la duración en meses del set de datos completo, que para este caso debía ser de 20.903, y finalmente la última petición que también era un GET, debía retornar la informació de los 5 primeros sujetos del set de datos.

Ahora observemos las peticiones realizadas desde POSTMAN para comprobar que se obtienen los mismos resultados que con el procedimiento anterior. En la Figura 19 se puede observar la realización de las peticiones así como las respuestas obtenidas, y efectivamente concuerdan con lo obtenido anteriormente. Para lograr esto desde el servicio en la Nube de RITA fue necesario habilitar un host y una IP publica mediante la cual se accedía a los servicios del modelo.

```

1 library(plumber)
2 r <- plumb("deploy_ml_credit_model.R")
3 r$run(host="0.0.0.0",port=8000)
4

```

```

[alejandra@localhost ~]$ curl -X POST -d '{"Status.of.existing.checking.account": "A11", "Duration.in.m
onth": 24, "Credit.history": "A32", "Savings.account.bonds": "A63"}' -H 'Content-Type: application/json
' localhost:8000/predict
{"default.probability":0.6224}[alejandra@localhost ~]$
[alejandra@localhost ~]$ curl -X GET -H 'Content-Type: application/json' ritaportal.udistrital.edu.co:1
0143/promes
[20.903][alejandra@localh
[alejandra@localhost ~]$ curl -X GET -H 'Content-Type: application/json' ritaportal.udistrital.edu.co:1
0143/top5
[{"Status.of.existing.checking.account": "A11", "Duration.in.month": 24, "Credit.history": "A33", "Purpose": "
A40", "Credit.amount": 4870, "Savings.account.bonds": "A61", "Employment.years": "A73", "Installment.rate.in.p
ercentage.of.disposable.income": 3, "Personal.status.and.sex": "A93", "Other.debtors...guarantors": "A101", "
Present.residence.since": 4, "Property": "A124", "Age.in.years": 53, "Other.installment.plans": "A143", "Housin
g": "A153", "Number.of.existing.credits.at.this.bank": 2, "Job": "A173", "Number.of.people.being.liable.to.pr
ovide.maintenance.for": 2, "Telephone": "A191", "Foreign.worker": "A201", "Status": 2}][alejandra@localhost ~]$

```

Figura 18. Peticiones POST y GET desde la terminal de R alojado en la Nube de RITA

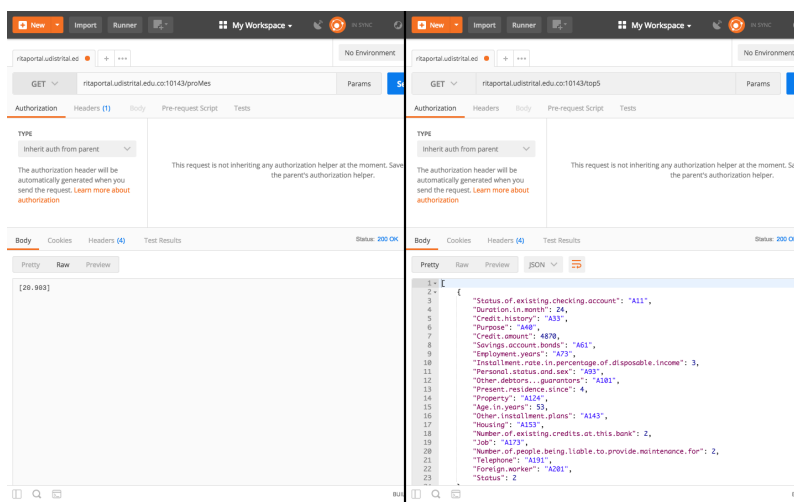


Figura 19. Peticiones GET con POSTMAN

5. Resultados y Conclusiones

- EL acceso a una base de datos a través de una url, no podía realizarse directamente desde *Sharelatex*, ya debía estar la base de datos de manera peexistente en el directorio de trabajo para que pudiera cargarse desde un *chunk* de código.
- Este trabajo permitió comprobar el concepto e servicio RESTful utilizando un programa sencillo desde R y con el cual fue posible no sólo encontrar una aplicación útil, sino que también fue posible evidenciar la interoperabilidad entre ambientes de trabajo y lenguajes de programación.

- Para este caso no se tomaron en cuenta algunos aspectos como mensajes de alerta y manejo de excepciones, lo que podría significar trastornos y confusiones a la hora de utilizarlo.
- POSTMAN es una herramienta útil para la verificación de APIs, sin embargo también es posible utilizar Shiny para este fin.
- Con json es posible crear diferentes y múltiples servicios que puedan ser accedidos desde cualquier lugar y desde cualquier ambiente.
- En este caso, no fue posible utilizar chunks de código desde R o desde Sharelatex, específicamente para el caso en que se creaba la API en R, ya que el servidor no lograba establecer la conexión de esta manera.
- Cada vez que se realizaba un cambio en el archivo de la API, era necesario correrlo y después realizar la creación de la API con *plumber* para que los cambios se actualizaran.
- La diferencia más grande entre un GET y un POST radica en los parámetros que reciben y la información que se obtiene. Mientras que para realizar una petición GET la herramienta no me exige ningún tipo de información para entregar una respuesta, con una petición POST sí era necesario ingresar valores que luego serían utilizados para devolver una respuesta.
- Debido a las políticas de seguridad de RITA, al comienzo no era posible abrir el puerto 8000 para poder acceder a la API, sin embargo se realizó la solicitud y concedieron permisos pero con un host específico.
- El R Studio de RITA abierto a la comunidad tiene una configuración con limitaciones: no posee terminal ni tiene permisos para hacer cambios dentro de la red, por lo cual, era imposible comprobar el funcionamiento de la API creada.
- Para solucionar esto, se montó un nuevo R Studio en una url específica para este experimento, y éste sí tenía la terminal para poder comprobar el funcionamiento de la API.
- Se utilizó una herramienta web (Hurl.it) para realizar peticiones al modelo creado, y se comprobó que funcionaba correctamente, sin embargo para poderlo utilizar era necesario ingresar la información de los encabezados y los valores que debían tener, lo cual lo hizo un poco tedioso y por eso no se incluyó dentro de este documento.
- Al tener una IP pública para acceder a la API, se garantiza que pueda accederse al servicio solicitado ya no exclusivamente de manera local, sino que puede accederse desde cualquier lugar de la WEB.

Referencias

- Terry M. Therneau Elizabeth J. Atkinson Mayo Foundation. An Introduction to Recursive Partitioning Using the RPART Routines. February 23, 2018
- A Ohri. R for Cloud Computing. Delhi, India
- <https://www.knowru.com>
- <https://www.getpostman.com>
- <https://archive.ics.uci.edu/ml/datasets/Statlog>
- <http://rest.elkstein.org/2008/02/rest-as-lightweight-web-services.html>

Índice alfabético

ANEXO, 2

DESCRIPCIÓN DE LOS ATRIBUTOS

Atributo 1: (cualitativo): Estado de cuenta corriente existente

A11 : menor a 0 DM

A12 : 0 - 200 DM

A13 : mayor a 200 DM /asignaciones de salario por al menos un año

A14 : no posee cuenta corriente

Atributo 2: (numérico)

Duración en meses

Atributo 3: (cualitativo)

Historial Crediticio

A30 : no posee créditos/todos los créditos fueron pagados correctamente

A31 : todos los créditos en este banco se pagaron correctamente

A32 : créditos existentes devueltos debidamente hasta ahora

A33 : retrasos en los pagos

A34 : cuenta corriente/posee créditos en otras entidades bancarias

Atributo 4: (cualitativo)

Propósito

A40 : carro (nuevo)

A41 : carro (usado)

A42 : muebles/equipos

A43 : radio/televisión

A44 : electrodomésticos

A45 : reparaciones

A46 : educación

A47 : vacaciones

A48 : reentrenamiento

A49 : negocios

A410 : otros

Atributo 5: (numérico).

Monto del Crédito.

Atributo 6: (cualitativo)

Cuenta de ahorros/bonos

A61 : menor a 100 DM

A62 : 100 - 500 DM

A63 : 500 - 1000 DM

A64 : mayor a 1000 DM

A65 : desconocido/ no posee cuenta de ahorros

Atributo 7: (cualitativo)

Empleo actual desde

A71 : desempleado

A72 : menos de 1 año

A73 : 1 - 4 años

A74 : 4 - 7 años

A75 : más de 7 años

Atributo 8: (numérico)

Tasa de pago en porcentaje del ingreso disponible

Atributo 9: (cualitativo)

Estado civil/sexo

A91 : masculino : divorciado/separado

A92 : femenino : divorciada/separada/casada

A93 : masculino : soltero

A94 : masculino : casado/viudo

A95 : femenino : soltera

Atributo 10: (cualitativo)

Otros deudres/garantes

A101 : ninguno

A102 : co-deudor

A103 : garante

Atributo 11: (numérico)

Residencia actual desde

Atributo 12: (cualitativo)

Propiedad

A121 : bienes raíces

A122 : si no

A121 : acuerdo de ahorros de la sociedad constructora/ seguro de vida

A123 : si no

A121/A122 : carros y otros diferentes al atributo 6

A124 : desconocido/sin propiedades

Atributo 13: (numérico)

Edad en años

Atributo 14: (cualitativo)

Otros planes de pago

A141 : banco
A142 : ahorros
A143 : ninguno

Atributo 15: (cualitativo)

Vivienda

A151 : alquilada
A152 : propia
A153 : préstamo a uso gratuito

Atributo 16: (numérica)

Número de créditos existentes en este banco

Atributo 17: (cualitativo)

Trabajo

A171 : desempleado
A172 : residente/pasante
A173 : empleado/oficial
A174 : autoempleado/administrativo/ejecutivo

Atributo 18: (numérico)

Número de personas a cargo

Atributo 19: (cualitativo)

Teléfono

A191 : ninguno
A192 : sí, registrado con el nombre del cliente

Atributo 20: (cualitativo)

Trabajador Extranjero

A201 : sí
A202 : no