Andalucía lab CENTRO DE INNOVACIÓN TURÍSTICA

Junta de Andalucía
Consejería de Turismo, Regeneración, Justicia y Administración Local

UNIÓN EUROPEA
Fondo Europeo de Desarrollo Regional

PROGRAMA DE ATRACCIÓN DE TALENTO TECNOLÓGICO PARA EL SECTOR TURÍSTICO ANDALUZ

2020

Machine Learning con lenguaje R.

FORMACIÓN TURISMO Y TECNOLOGÍA

UTE conformada por:

ideanto comunicación y tecnología

arelance Building Digital Talent

Machine Learning en lenguaje R.

Aprendizaje con reglas de asociación: Apriori.

- Se trata de un mecanismo de aprendizaje en donde se determina la probabilidad de que un evento ocurra en función a otro que se ha observado previamente
- Ejemplos típicos de esto se encuentran en estudios de mercado en donde se intenta predecir la conducta de un sujeto en función a un perfil generado en base a gustos comunes que podrían tener otros sujetos.
- Ejemplo de la cesta de la compra: Se observa que de “Z” compras, un porcentaje determinado compra un producto “A” y “B”. Se determina que hay una asociación entre la compra de los productos “A” y “B”.
- Ejemplo de películas: Se observa que de “Z” usuarios que han visto una película “A” un porcentaje significativo también han visto la película “B”, por lo tanto puede existir una asociación entre “A” y “B” para el perfil de usuarios “Z” puede

Aprendizaje con reglas de asociación: Apriori.

User ID	Películas que le han gustado
46578	Película1, Película2, Película3, Película4
98989	Película1, Película2
71527	Película1, Película2, Película4
78981	Película1, Película2
89192	Película2, Película4
61557	Película1, Película3

Reglas Significativas:

Película1 ⇒ Película2

Película2 ⇒ Película4

Película1 ⇒ Película3

Aprendizaje con reglas de asociación: Apriori.

Transaction ID	Productos comprados
46578	Hamburguesas, Patatas, Verduras
98989	Hamburguesas, Patatas, Ketchup
71527	Verduras, Fruta
78981	Pasta, Fruta, Mantequilla, Verduras
89192	Hamburguesas, Pasta, Patatas
61557	Fruta, Zumo de Naranja, Verduras
87923	Hamburguesas, Patatas, Ketchup, Mayo

Reglas Significativas:

Hamburguesas → Patatas

Verduras → Fruta

Hamburguesas, Patatas → Ketchup

Aprendizaje con reglas de asociación: Apriori.

Definición de “Soporte” para el modelo Apriori.

Recomendación de Películas:

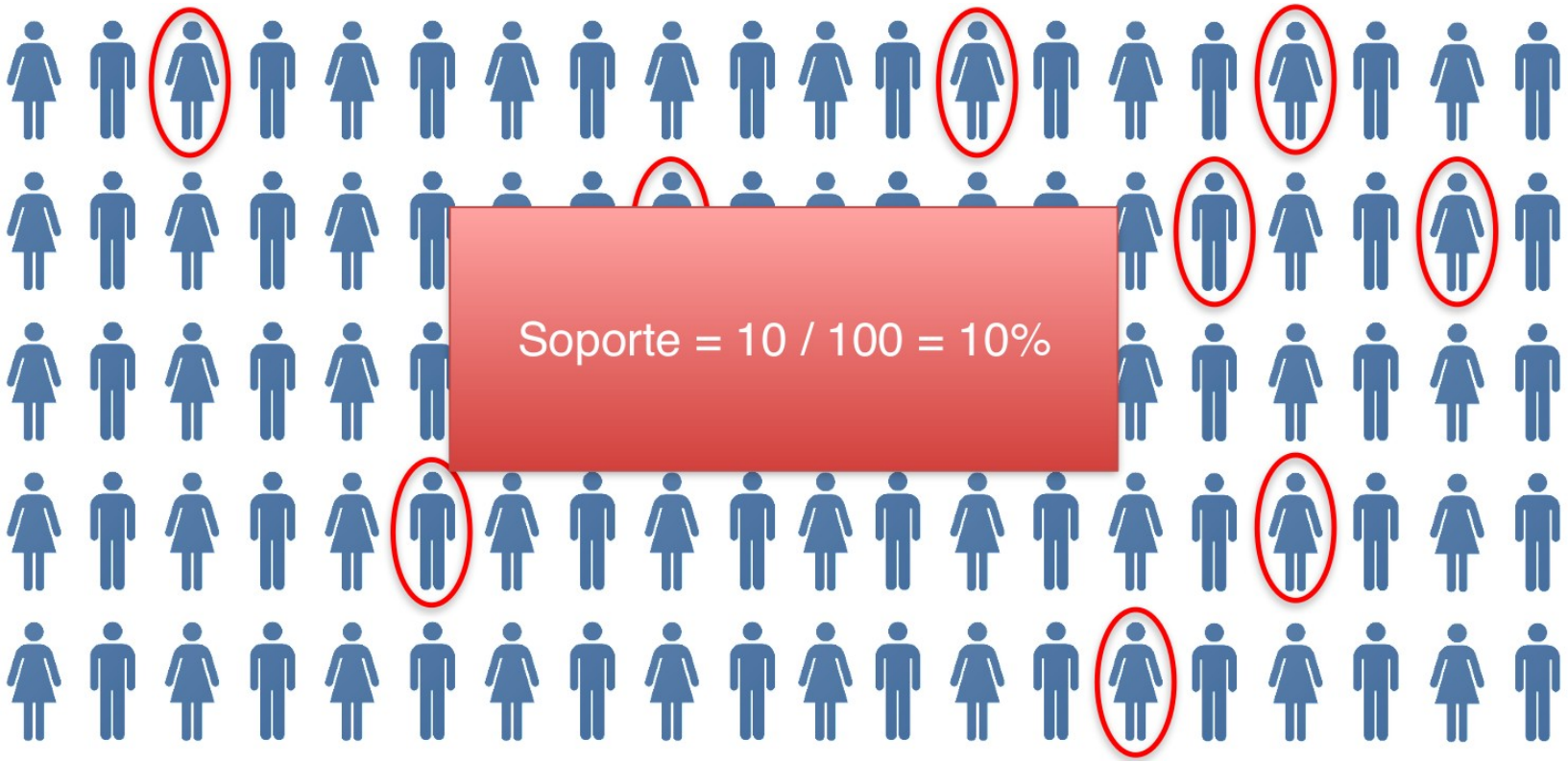
$$sop(\mathbf{M}) = \frac{|\text{usuarios que vieron } \mathbf{M}|}{|\text{usuarios}|}$$

Optimización de la Cesta de la Compra:

$$sop(\mathbf{I}) = \frac{|\text{transacciones que contienen } \mathbf{I}|}{|\text{transacciones}|}$$

Aprendizaje con reglas de asociación: Apriori.

En una muestra de 100 observaciones 10 cumplen con premisa 1→ soporte



Aprendizaje con reglas de asociación: Apriori.

Definición de “Confianza” para el modelo Apriori.

Recomendación de Películas:

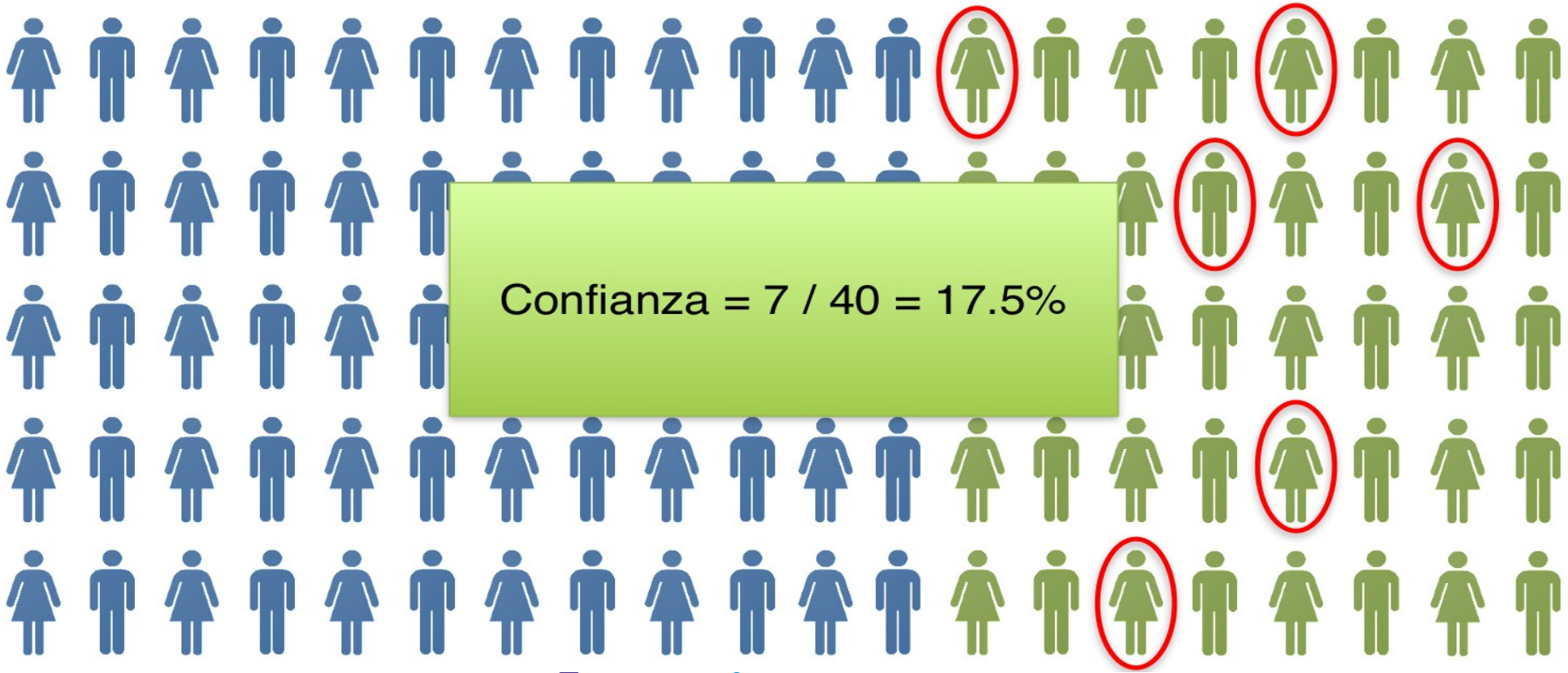
$$conf(\mathbf{M}_1 \Rightarrow \mathbf{M}_2) = \frac{|\text{usuarios que vieron } \mathbf{M1} \text{ y } \mathbf{M2}|}{|\text{usuarios que vieron } \mathbf{M1}|}$$

Optimización de la Cesta de la Compra:

$$conf(\mathbf{I}_1 \Rightarrow \mathbf{I}_2) = \frac{|\text{transacciones que contienen } \mathbf{I1} \text{ y } \mathbf{I2}|}{|\text{transacciones que contienen } \mathbf{I1}|}$$

Aprendizaje con reglas de asociación: Apriori.

En una muestra de 100 observaciones 40 cumplen con premisa 1 y además, 7 cumplen con la premisa 2→ Confianza.



Aprendizaje con reglas de asociación: Apriori.

Definición de “Lift” para el modelo Apriori.

Recomendación de Películas:

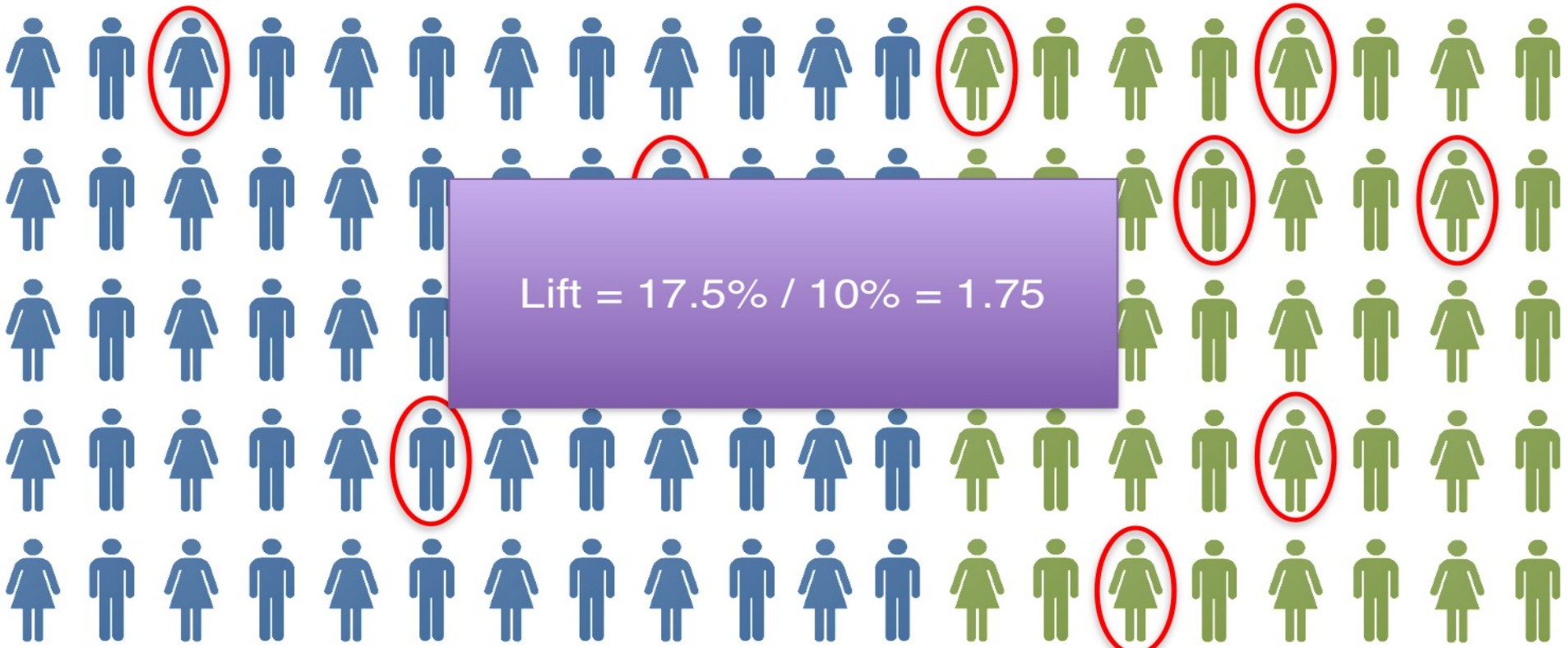
$$lift(\mathbf{M}_1 \Rightarrow \mathbf{M}_2) = \frac{conf(\mathbf{M}_1 \Rightarrow \mathbf{M}_2)}{sup(\mathbf{M}_1)}$$

Optimización de la Cesta de la Compra:

$$lift(\mathbf{I}_1 \Rightarrow \mathbf{I}_2) = \frac{conf(\mathbf{I}_1 \Rightarrow \mathbf{I}_2)}{sup(\mathbf{I}_1)}$$

Aprendizaje con reglas de asociación: Apriori.

Se mide la probabilidad que se cumpla la premisa 2 en función a aquellas observaciones que han cumplido la premisa 1→ Lift. La probabilidad se mide ahora sobre el conjunto de datos completo. Mejora un 1,75%



Aprendizaje con reglas de asociación: Apriori.

Paso 1: Decidir un soporte y nivel de confianza mínimo

Paso 2: Elegir todos los subconjuntos de transacciones con soporte superior que el mínimo elegido

Paso 3: Elegir todas las reglas de estos subconjuntos con nivel de confianza superior al mínimo elegido

Paso 4: Ordenar todas las reglas anteriores por lift descendiente

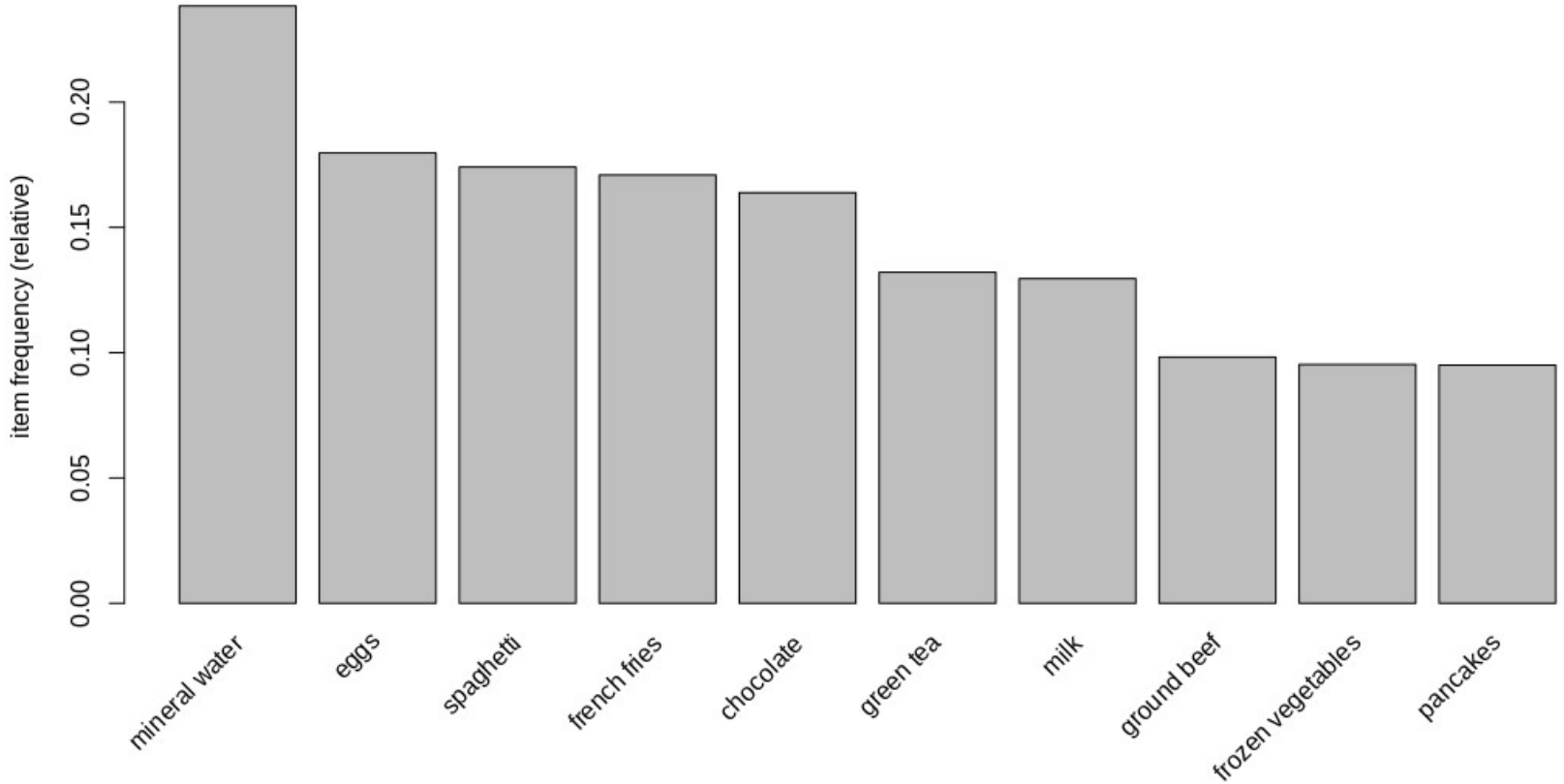
Aprendizaje con reglas de asociación: Apriori.

```
# Apriori

#Paso 1. Preprocesado de Datos e instalación de librería arules
install.packages("arules")
install.packages("arulesViz")
library(arules)
library(arulesViz)

#En este caso el dataset no tiene cabeceras, por lo tanto se debe indicar el parámetro "header" a FALSE.
#Esto se hace para que no utilice el primer registro como nombre de las columnas.
#En este dataset en concreto cada fila representa una cesta de la compra. Clientes que han llegado a un supermercado
#y ha comprado los productos que aparecen en cada fila del dataset (cesta de la compra).
dataset = read.csv("Datasets/Market_Basket_Optimisation.csv", header = FALSE)

#dada la forma que tiene el dataset, en este caso el procesado de datos será distinto.
#Se procede a crear una matriz de dispersión, la cual se encargará de organizar los valores de una forma más natural.
#La matriz se encargará de ubicar los valores del dataset original en las columnas (es decir, los productos de la cesta).
#Y cada fila de dicha matriz de dispersión será cada una de las transacciones (compras).
# El valor de cada celda representará si en cada transacción se ha comprado o no el producto referenciado en la columna.
# En este caso, los productos duplicados se han eliminado con "rm.duplicates".
dataset = read.transactions("Datasets/Market_Basket_Optimisation.csv",
                            sep = ",", rm.duplicates = TRUE)
```

Aprendizaje con reglas de asociación: Apriori.

```
#Paso 2. Análisis de los datos.
#Con la función summary sobre el dataset generado, se podrán observar el número de columnas (productos de las compras).
#También se podrán ver otros detalles como los items más frecuentes en el conjunto de datos, por ejemplo:
#most frequent items:
#mineral water          eggs     spaghetti  french fries     chocolate       (Other)
#1788                   1348          1306          1282          1229         22405
#Significa que agua mineral y huevos son los productos que han aparecido con más frecuencia en las transacciones (compras).
#Por otro lado también se puede apreciar la longitud de la districión, indicando cuántos items se han incluido en cada transacción.
#Por ejemplo:
# element (itemset/transaction) length distribution:
#   sizes
# 1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   18   19   20
# 1754 1358 1044  816  667  493  391  324  259  139  102   67   40   22   17    4    1    2    1
#
# Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
# 1.000   2.000   3.000   3.914   5.000  20.000
#
# Significa que, por ejemplo, 1754 transacciones contenian solamente 1 producto, 1358 contenian 2 productos, etc.
# También indica que la media de los usuarios compra 3,914 items, la mediana 3 y el máximo 20 (campana de Gauss).
summary(dataset)

#Paso 3. Enseñar un plot con los items más frecuentes del dataset.
itemFrequencyPlot(dataset, topN = 10)
```

Aprendizaje con reglas de asociación: Apriori.



Aprendizaje con reglas de asociación: Apriori.

```
#Paso 4. Entrenar algoritmo Apriori con el dataset
#A continuación se utiliza el método "apriori" que se encuentra disponible en la librería "arules".
#Esta función solo necesita el dataset que en este caso es la matriz de dispersión y el argumento "parameter".
#Dicho argumento recibe una lista, en la cual es necesario indicar el valor mínimo de soporte y confianza.
#Estos dos valores son importantes para definir las reglas de asociación entre productos con unas condiciones mínimas.
#Para decidir qué valores indicar en estos parámetros puede ser útil ver el gráfico anterior sobre la frecuencia relativa de los items.
#Evidentemente, para que las reglas generadas sean más significativas y aporten valor al modelo, se seleccionan
#aquellos items que tienen la mayor frecuencia relativa.
#Por ejemplo, el soporte podría ser aquellos productos que se compran al menos 3 veces cada día.
#Dado que el número de observaciones (transacciones) es de 7500, se procede a hacer el calculo: 3*7/7500 = 0.0028.
#Dicho valor se puede redonderar, por ejemplo a 0.003 y este sería el soporte del modelo.
#Para el nivel de confianza, al ver la documentación de la función apriori, el valor por defecto es 0.8
#El problema de este nivel de confianza es que significa que las reglas creadas deben cumplirse
#en al menos el 80% de las transacciones estudiadas, lo cual puede traducirse en que no se pueden generar reglas con un mínimo tan exigente.

rules = apriori(data = dataset,
                parameter = list(support = 0.003, confidence = 0.4))
```

Aprendizaje con reglas de asociación: Apriori.

```
#Paso 4. Entrenar algoritmo Apriori con el dataset
#A continuación se utiliza el método "apriori" que se encuentra disponible en la librería "arules".
#Esta función solo necesita el dataset que en este caso es la matriz de dispersión y el argumento "parameter".
#Dicho argumento recibe una lista, en la cual es necesario indicar el valor mínimo de soporte y confianza.
#Estos dos valores son importantes para definir las reglas de asociación entre productos con unas condiciones mínimas.
#Para decidir qué valores indicar en estos parámetros puede ser útil ver el gráfico anterior sobre la frecuencia relativa de los items.
#Evidentemente, para que las reglas generadas sean más significativas y aporten valor al modelo, se seleccionan
#aquellos items que tienen la mayor frecuencia relativa.
#Por ejemplo, el soporte podría ser aquellos productos que se compran al menos 3 veces cada día.
#Dado que el número de observaciones (transacciones) es de 7500, se procede a hacer el calculo: 3*7/7500 = 0.0028.
#Dicho valor se puede redonderar, por ejemplo a 0.003 y este sería el soporte del modelo.
#Para el nivel de confianza, al ver la documentación de la función apriori, el valor por defecto es 0.8
#El problema de este nivel de confianza es que significa que las reglas creadas deben cumplirse
#en al menos el 80% de las transacciones estudiadas, lo cual puede traducirse en que no se pueden generar reglas con un mínimo tan exigente.

rules = apriori(data = dataset,
                parameter = list(support = 0.003, confidence = 0.4))
```

Aprendizaje con reglas de asociación: Apriori.

Generación del “top 10” de las reglas más relevantes ordenadas por lift.

```
> inspect(sort(rules, by = 'lift')[1:10])
     lhs                                          rhs                  support     confidence coverage    lift     count
[1]  {mineral water,whole wheat pasta}         => {olive oil}          0.003866151 0.4027778  0.009598720 6.115863 29
[2]  {spaghetti,tomato sauce}                  => {ground beef}        0.003066258 0.4893617  0.006265831 4.980600 23
[3]  {french fries,herb & pepper}              => {ground beef}        0.003199573 0.4615385  0.006932409 4.697422 24
[4]  {cereals,spaghetti}                       => {ground beef}        0.003066258 0.4600000  0.006665778 4.681764 23
[5]  {frozen vegetables,mineral water,soup}    => {milk}               0.003066258 0.6052632  0.005065991 4.670863 23
[6]  {chocolate,herb & pepper}                 => {ground beef}        0.003999467 0.4411765  0.009065458 4.490183 30
[7]  {chocolate,mineral water,shrimp}          => {frozen vegetables}  0.003199573 0.4210526  0.007598987 4.417225 24
[8]  {frozen vegetables,mineral water,olive oil} => {milk}             0.003332889 0.5102041  0.006532462 3.937285 25
[9]  {cereals,ground beef}                     => {spaghetti}          0.003066258 0.6764706  0.004532729 3.885303 23
[10] {frozen vegetables,soup}                  => {milk}               0.003999467 0.5000000  0.007998933 3.858539 30
>
```

¿Las reglas generadas tienen sentido? ¿Qué ocurre con ellas si se aumenta el nivel de confianza exigido al 60%? ¿Qué ocurre con ellas si se aumenta el nivel de confianza exigido al 50%? ¿Siguen siendo significativas?

Aprendizaje con reglas de asociación: Apriori.

```
#Paso 5. Visualización de los resultados.
#Se debe aplicar el método "lift" que basicamente consiste en ordenar las reglas de mayor a menor dependiendo de su relevancia.
#La función "inspect" del paquete "arules" permite precisamente obtener dicho conjunto de reglas.
#inspect(rules[1:10]) #Se obtienen 10 reglas, pero no en orden de significancia.
#La función "sort" permite ordenar las reglas utilizando el método "lift", es decir, las más significativas primero.
inspect(sort(rules, by = 'lift')[1:10])

plot(rules, method = "graph", engine = "htmlwidget")
```

Aprendizaje con reglas de asociación: Eclat.

- El modelo Eclat es similar al modelo “Apriori” pero con la diferencia de que se busca determinar el soporte para un conjunto de items. Es decir, cuál es la probabilidad de un conjunto de items tenga una relación de asociación.
- Una diferencia crucial con respecto al modelo Apriori es que Eclat trabaja a nivel de agrupaciones, mientras que Apriori lo hace sobre items independientes partiendo de la premisa de que no hay relaciones entre cada item y que se deben calcular o generar en función a calculos como el soporte, la confianza y el lift.
- Eclat intenta “empaquetar” las observaciones.

Aprendizaje con reglas de asociación: Eclat.

Paso 1: Configurar un soporte mínimo a utilizar

Paso 2: Tomar todos los subconjuntos de las transacciones con soporte superior al mínimo establecido.

Paso 3: Ordenar esos subconjuntos por support descendente

Aprendizaje con reglas de asociación: Eclat.

```
# Eclat

#Paso 1. Preprocesado de Datos
#install.packages("arules")
library(arules)
dataset = read.csv("Datasets/Market_Basket_Optimisation.csv", header = FALSE)
dataset = read.transactions("Market_Basket_Optimisation.csv",
                            sep = ",", rm.duplicates = TRUE)
summary(dataset)
itemFrequencyPlot(dataset, topN = 10)

#Paso 2. Entrenar algoritmo Eclat con el dataset
#En este caso el algoritmo de eclat se encargará de realizar los calculos en función a grupos, en lugar de hacerlo sobre items
#Apriori requiere de un nivel de soporte y confianza mínimo, en el casl de eclat, solo es necesario el nivel de soporte
#mientras que el nivel de confianza no se especifica. No obstante, es importante indicar de cuántos elementos estará compuesto el grupo.
#El parametro "minlen" permite establecer el número de elementos que debe contener cada regla de asociación
#generada por el algoritmo "eclat", es decir, que los grupos de items asociados deben estar compuestos por un valor mínimo X.
rules = eclat(data = dataset,
              parameter = list(support = 0.003, minlen = 2))

# Paso 3. Visualización de los resultados
inspect(sort(rules, by = 'support')[1:10])
```

Técnicas para selección de modelos.

- Una vez estudiados los principales modelos de Machine Learning es el momento de abordar algunas cuestiones:

¿Cómo gestionar la diferencia de datos entre evaluar en la fase de entrenamiento y de testing para medir su eficacia?

¿Cómo elegir los valores óptimos de los hiperparámetros del algoritmo?

¿Cómo elergir el modelo de Machine Learning más adecuado en función al problema?

Técnicas para selección de modelos.

Para contestar a las cuestiones anteriores existen técnicas de selección de modelos, en donde las más interesantes y potentes son las siguientes:

k-Fold Cross Validation

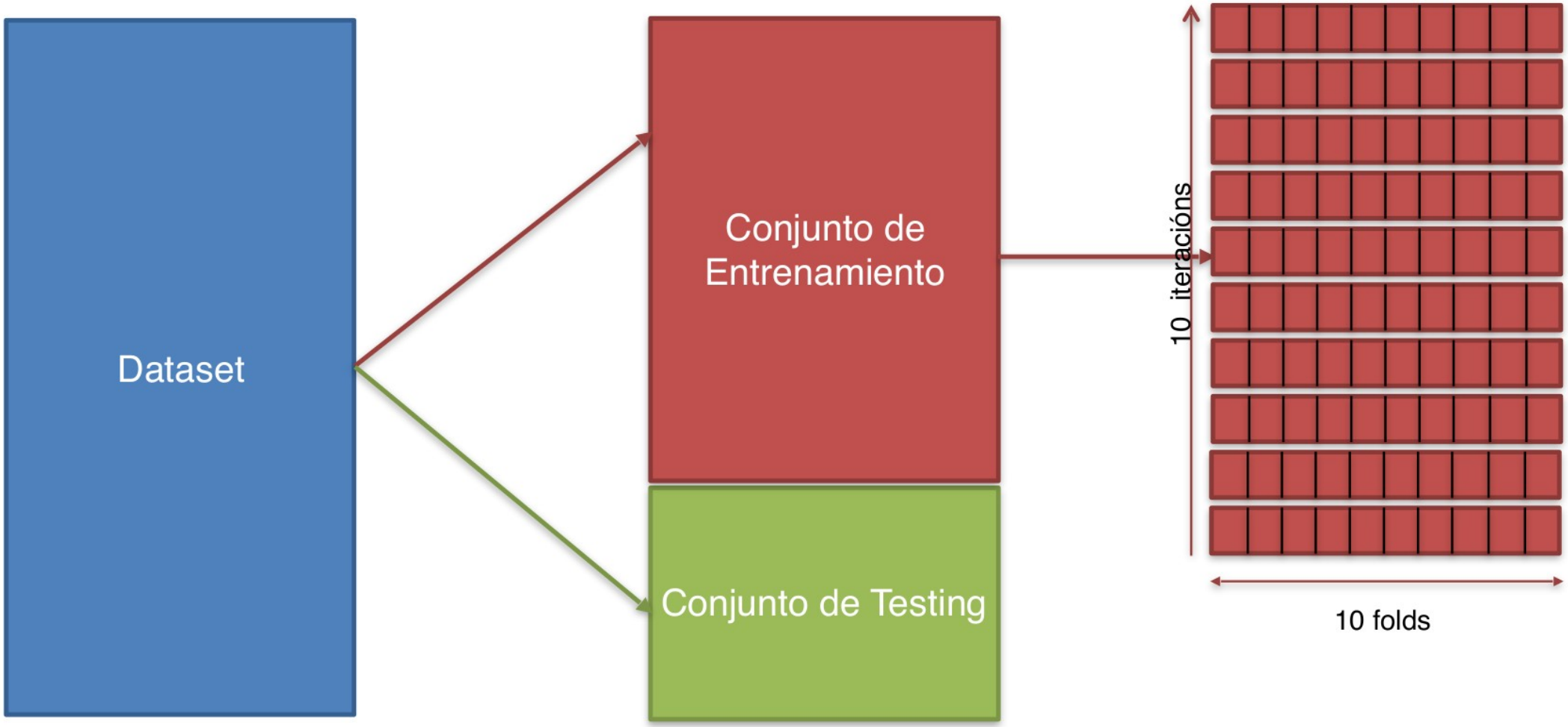
Grid Search

XGBoost.

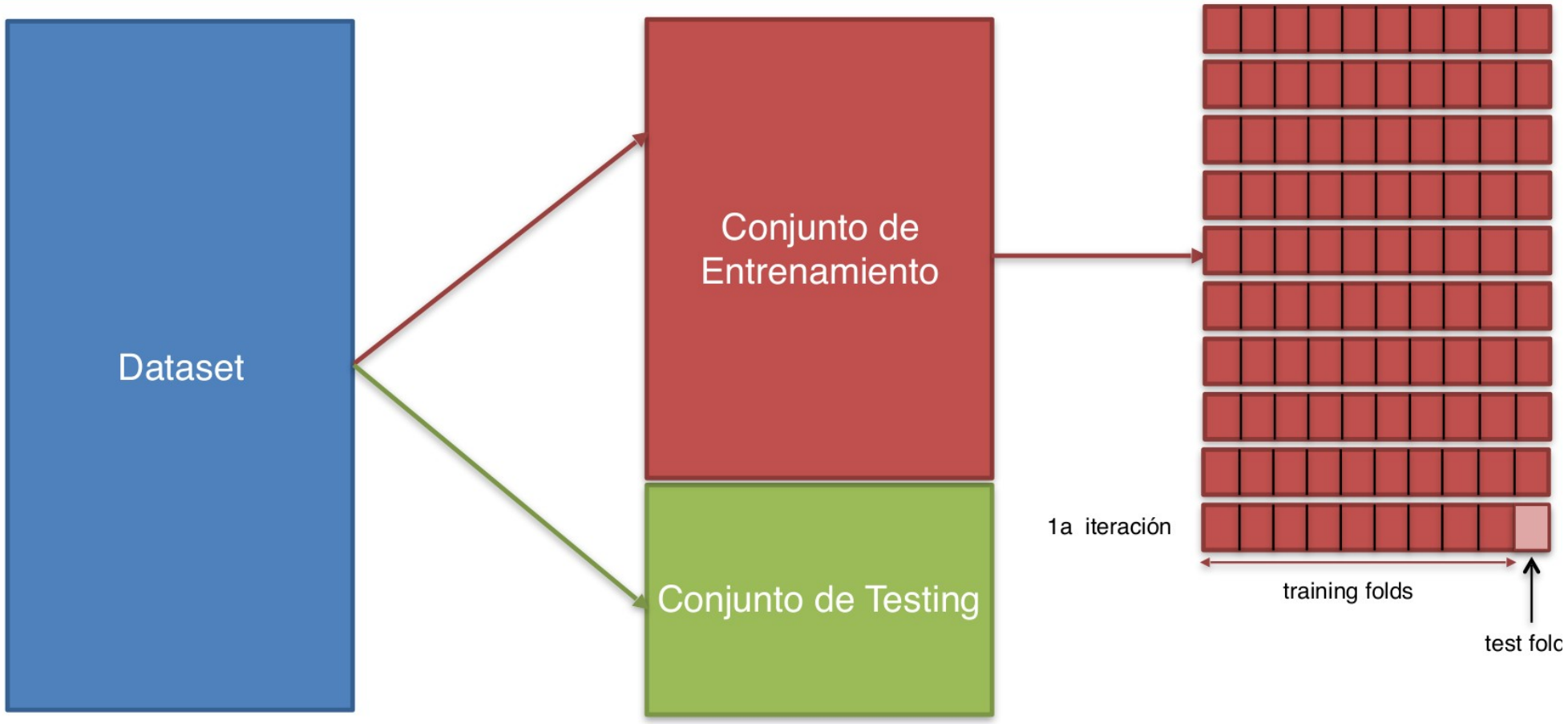
Selección de modelos – F-Cross Valitation

- Los modelos que se han utilizado hasta ahora han utilizado un conjunto de entrenamiento y testing, los cuales se han generado de forma aleatoria tomando un porcentaje del dataset original. No obstante, el problema de este enfoque es que al cambiar el conjunto de pruebas pueden haber cambios en la varianza y efectivamente, al cambiar los datos de prueba pueden aparecer porcentajes de efectividad distintos para el mismo modelo y mismo conjunto de datos.
- Esto significa que juzgar si el modelo es valido o no en función a un único conjunto de prueba puede no ser el mejor enfoque.
- El modelo F-Cross Validation, se encarga de dividir el conjunto de entrenamiento 10 veces. Por lo tanto, cada una de las muestras que se utilice para validar el entrenamiento será diferente.

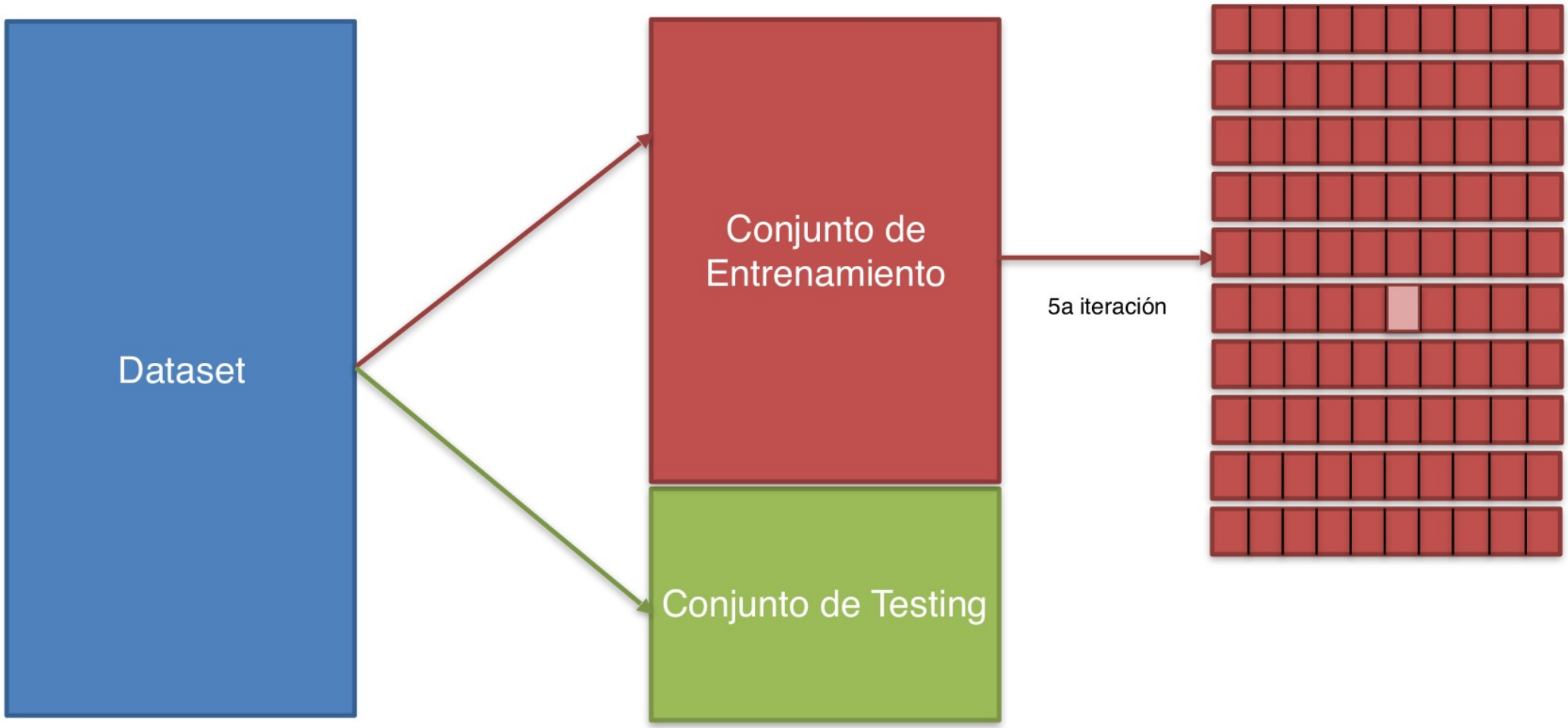
Selección de modelos – F-Cross Valitation



Selección de modelos – F-Cross Valitation

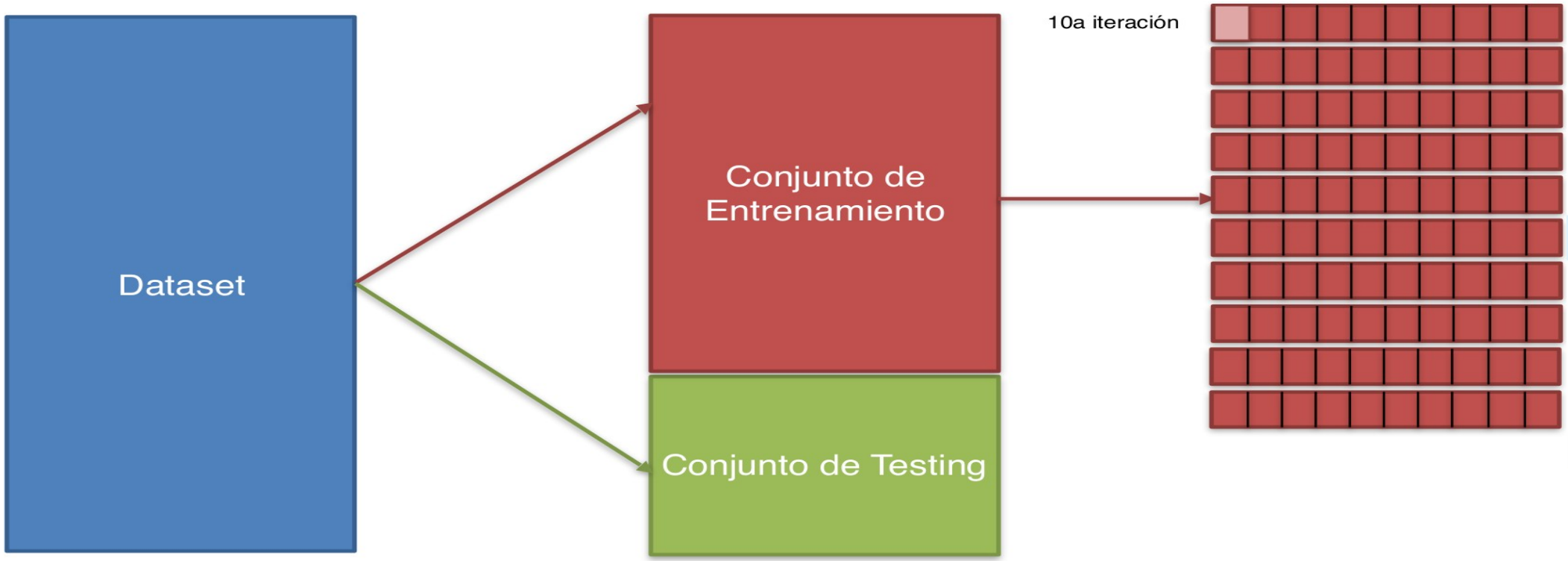


Selección de modelos – F-Cross Valitation



Selección de modelos – F-Cross Valitation

- En cada iteración, se utilizan 9 bloques para entrenar y 1 validar. De esta manera el resultado de la etapa de entrenamiento podrá ser más fiable y se podrá medir cuantitativamente el sesgo y la varianza. Se toma la media y la desviación de cada iteración para saber la varianza.



Selección de modelos – F-Cross Valitation

Sesgo bajo: cuando el modelo elabora predicciones cercanas a los datos reales.

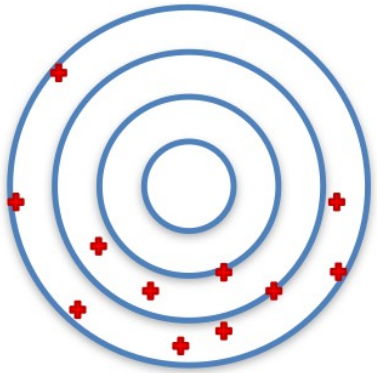
Sesgo alto: cuando el modelo elabora predicciones alejadas de los datos reales.

Varianza baja: cuando ejecutamos el modelo varias veces y las predicciones no varian demasiado.

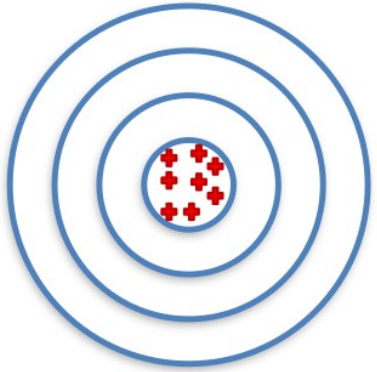
Varianza elevada: cuando ejecutamos el modelo varias veces y las predicciones varian demasiado.

Selección de modelos – F-Cross Valitation

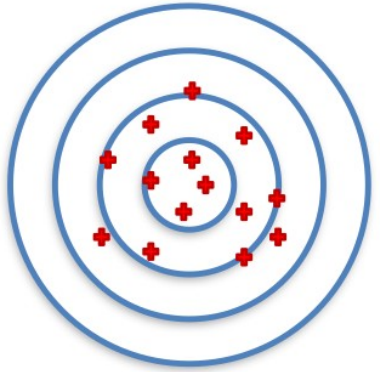
Sesgo alto Varianza baja



Sesgo alto Varianza alta



Sesgo bajo Varianza baja



Sesgo bajo Varianza alta

Selección de modelos – F-Cross Valitation

```r
# k-fold cross validation

#Paso 1. Importar el dataset. #Se aplicará el mecanismo F-Cross validation sobre el modelo Kernel SVM. El código se basa en el mismo ejemplo visto antes para SVM.
dataset = read.csv('Datasets/Social_Network_Ads.csv'); dataset = dataset[, 3:5]

#Paso 2. Dividir los datos en conjunto de entrenamiento y conjunto de test
# install.packages("caTools")
library(caTools); set.seed(123); split = sample.split(dataset$Purchased, SplitRatio = 0.75)
training_set = subset(dataset, split == TRUE); testing_set = subset(dataset, split == FALSE)

#Paso 3. Escalado de valores
training_set[,1:2] = scale(training_set[,1:2]); testing_set[,1:2] = scale(testing_set[,1:2])

#Paso 4. Ajustar el clasificador con el conjunto de entrenamiento.
#install.packages("e1071")
library(e1071); classifier = svm(formula = Purchased ~ ., data = training_set, type = "C-classification", kernel = "radial")

#Paso 5. Predicción de los resultados con el conjunto de testing
y_pred = predict(classifier, newdata = testing_set[,-3])

#Paso 6. Crear la matriz de confusión
cm = table(testing_set[, 3], y_pred)
```

Selección de modelos – F-Cross Valitation

```r
#Paso 7. Aplicar algoritmo de k-fold cross validation.
#Se debe dividir el conjunto de entrenamiento en 10 bloques, dicho valor es arbitrario y se puede cambiar en el parámetro "k".
#A continuación se utiliza la función "lapply" la cual recibe los bloques creados anteriormente con la librería caret y una función anónima o lambda.
# La función lapply ejecutará la función anónima tantas veces como bloques se le envie al primer argumento (flods, que en este caso son 10).
# Esto signfica que que dicha función se ejecutará lo que se le indique 10 veces, que en este caso será el modelo SVM y así se aplica la técnica F-Cross.
# Dentro de la función anónima, se obtiene el conjunto de entrenamiento completo excluyendo el bloque, tal como se ve en la imagen de la diapositiva
# y el conjunto de prueba será simplemente el bloque entero. A continuación se ejecuta el SVM, la predicción y la tabla CM como se ha visto anteriormente.
# No obtante, para cada ejecución del bloque se debe calcular la presición, para ello se utiliza la matriz de confusión de la siguiente forma:
# accuracy = Suma Predicciones Correctas/Total de Predicciones.
# Dicho calculo es precisamente lo que debe devolver la función.
install.packages("caret")
library(caret)
folds = createFolds(training_set$Purchased, k = 10)
cv = lapply(folds, function(x) {
  training_fold = training_set[-x, ]
  test_fold = training_set[x, ]
  classifier = svm(formula = Purchased ~ .,
                   data = training_fold,
                   type = "C-classification",
                   kernel = "radial")
  y_pred = predict(classifier, newdata = test_fold[,-3])
  cm = table(test_fold[, 3], y_pred)
  accuracy = (cm[1,1]+cm[2,2])/(cm[1,1]+cm[1,2]+cm[2,1]+cm[2,2])
  return(accuracy)
})
```

Selección de modelos – F-Cross Valitation

```
#Paso 8. Calculo de la media de las predicciones utilizando el método F-Cross.
#Se debe castear la lista como valores números y aplicar la media sobre dicha lista.
#El valor resultante de dicha media es el nivel de fiabilidad del modelo aplicado y aporta un rendimiento que en este caso es bastante elevado.
#También resulta conveniente calcular el nivel de varianza (desviasión estándar) para saber si entre cada bloque hay cambios grandes.
accuracy = mean(as.numeric(cv))
accuracy_sd = sd(as.numeric(cv))
# > accuracy
# [1] 0.9133333
# > accuracy_sd
# [1] 0.06324555
```

Selección de modelos – Grid Search

- Se trata de un mecanismo que permite probar de forma automática, los hiperparámetros de un modelo concreto.

- En el caso de R existe un proyecto llamado “caret” que cuenta con la función “train” la cual realiza el proceso de afinar el modelo indicado por parámetro y enseñar una serie de estadísticas que pueden ser útiles para efectos de análisis.

- Su implementación puede aplicarse justo después de llevar a cabo el modelo de F-Cross validation con el fin de tener aplicar ambas técnicas juntas y generar la mejor optimización posible del modelo seleccionado.

Selección de modelos – Grid Search

```r
# Grid Search

#Paso 1. Importar el dataset
dataset = read.csv('Datasets/Social_Network_Ads.csv')
dataset = dataset[, 3:5]
dataset$Purchased = factor(dataset$Purchased)

#Paso 2. Dividir los datos en conjunto de entrenamiento y conjunto de test
# install.packages("caTools")
library(caTools)
set.seed(123)
split = sample.split(dataset$Purchased, SplitRatio = 0.75)
training_set = subset(dataset, split == TRUE)
testing_set = subset(dataset, split == FALSE)

#Paso 3. Escalado de valores
training_set[,1:2] = scale(training_set[,1:2])
testing_set[,1:2] = scale(testing_set[,1:2])

#Paso 4. Ajustar el clasificador con el conjunto de entrenamiento.
#install.packages("e1071")
library(e1071)
classifier = svm(formula = Purchased ~ .,
                 data = training_set,
                 type = "C-classification",
                 kernel = "radial")
# Predicción de los resultados con el conjunto de testing
y_pred = predict(classifier, newdata = testing_set[,-3])
# Crear la matriz de confusión
cm = table(testing_set[, 3], y_pred)
```

Selección de modelos – Grid Search

```
#Paso 5. Aplicar algoritmo de k-fold cross validation
# install.packages("caret")
library(caret)
folds = createFolds(training_set$Purchased, k = 10)
cv = lapply(folds, function(x) {
  training_fold = training_set[-x, ]
  test_fold = training_set[x, ]
  classifier = svm(formula = Purchased ~ .,
                   data = training_fold,
                   type = "C-classification",
                   kernel = "radial")
  y_pred = predict(classifier, newdata = test_fold[,-3])
  cm = table(test_fold[, 3], y_pred)
  accuracy = (cm[1,1]+cm[2,2])/(cm[1,1]+cm[1,2]+cm[2,1]+cm[2,2])
  return(accuracy)
})
accuracy = mean(as.numeric(cv))
accuracy_sd = sd(as.numeric(cv))
```

Selección de modelos – Grid Search

```
#Paso 6. Aplicar Grid Search para encontrar los parámetros óptimos.
#Se utiliza la función train para aplicar la técnica de Grid Search. Como ocurre con otros modelos vistos, es necesario indicar
#la relación de variables independientes y dependientes, el dataset de entrenamiento y finalmente el modelo utilizado.
#Probablemente el parámetro más importante en este punto es precisamente el método, ya que representa una cadena con el modelo que utilizará caret.
#Los métodos disponibles se encuentran documentados en el github oficial de la librería caret.
install.packages("caret")
install.packages("kernlab")
library(caret)
classifier = train(form = Purchased ~ .,
                   data = training_set, method = 'svmRadial')
classifier
classifier$bestTune

##Una vez ejecutada la función train con el modelo "SVM con kernel Radial" saldrán unos resultados similares a los siguientes:
# > classifier
# Support Vector Machines with Radial Basis Function Kernel

# Tuning parameter 'sigma' was held constant at a value of 1.18122
# ...........
# Accuracy was used to select the optimal model using the largest value.
# The final values used for the model were sigma = 1.18122 and C = 1.
# > classifier$bestTune
# sigma C
# 3 1.18122 1

# Significa que los hiperparametros "sigma" y "C" dan mejores resultados con valores indicados.
# classifier = svm(formula = Purchased ~ .,
#                  data = training_fold,
#                  type = "C-classification",
#                  kernel = "radial", C=1, sigma=1.18122)
```

Selección de modelos – Ejemplo con XGBoost

```
# XGBoost
#Paso 1.  Importar el dataset. Se intenta determinar si dadas un conjunto de variables un cliente deja o no el banco.
dataset = read.csv('Churn_Modelling.csv')
dataset = dataset[, 4:14]
#Paso 2. Codificar los factores.
dataset$Geography = as.numeric(factor(dataset$Geography,
                                      levels = c("France", "Spain", "Germany"),
                                      labels = c(1, 2, 3)))
dataset$Gender = as.numeric(factor(dataset$Gender,
                                   levels = c("Female", "Male"),
                                   labels = c(1,2)))
#Paso 3. Dividir los datos en conjunto de entrenamiento y conjunto de test
# install.packages("caTools")
library(caTools)
set.seed(123)
split = sample.split(dataset$Exited, SplitRatio = 0.8)
training_set = subset(dataset, split == TRUE)
testing_set = subset(dataset, split == FALSE)

# Paso 4. Ajustar XGBoost al Conjunto de Entrenamiento
#install.packages("xgboost")
library(xgboost)
classifier = xgboost(data = as.matrix(training_set[, -11]),
                     label = training_set$Exited,
                     nrounds = 10)
```

Selección de modelos – Ejemplo con XGBoost

```
#Paso 5. Aplicar algoritmo de k-fold cross validation
# install.packages("caret")
library(caret)
folds = createFolds(training_set$Exited, k = 10)
cv = lapply(folds, function(x) {
  training_fold = training_set[-x, ]
  test_fold = training_set[x, ]
  classifier = xgboost(data = as.matrix(training_set[, -11]),
                       label = training_set$Exited,
                       nrounds = 10)
  y_pred = predict(classifier, newdata = as.matrix(test_fold[,-11]))
  y_pred = (y_pred >= 0.5)
  cm = table(test_fold[, 11], y_pred)
  accuracy = (cm[1,1]+cm[2,2])/(cm[1,1]+cm[1,2]+cm[2,1]+cm[2,2])
  return(accuracy)
})
accuracy = mean(as.numeric(cv))
accuracy_sd = sd(as.numeric(cv))
```