Simulación Factorial Exploratorio

Manuel Torres Acosta

Índice

Objetivo

La tarea actual consiste en utilizar simulaciones para evaluar el funcionamiento del análisis factorial en algunas circunstancias de nuestra elección. En ésta práctica analizaré la calidad de la recuperación factorial y los valores que adopta el MSA de la técnica KMO en función del tamaño muestral y el valor de los pesos simulados para los factores. Además estudiaremos si existe efecto de interacción entre éstos elementos.

Código para las simulaciones

Antes de realizar las simulaciones es conveniente crear algunas funciones para hacer más fácil el desarrollo de las mismas.

```
simDF <- function(nRow, nCol, lambda = 0, means = 0, std = 1,
                  includeFactor = FALSE,
                  existingFactor = NA){
  #lambda define si los items estan correlacionados
  #si vale 0, saca valores normales. Si adopta otro valor,
  #es el peso factorial
  #means, std: media y desv tip de las variables
  #includeFactor: si lambda != 0, incluye el factor
  #al final del fichero
  #existingFactor: vector. Crea nuevos items para un
  #factor ya existente en otro fichero
  #asi se pueden crear con pesos diferentes
  df <- as.list(rep(nRow, nCol))
  if(lambda == 0){
      df <- lapply(df, function(x) rnorm(n = x))
  } else {
    if(length(existingFactor) > 1){
      factorComun <- existingFactor
    } else {
      factorComun <- rnorm(n = nRow)
    }
    generaVector <- function(x){
      error <- sqrt(1 - (lambda^2)) * rnorm(n = x)
      x <- (lambda * factorComun) + error
      return(x)
    }
    df <- lapply(df, generaVector)
    if(includeFactor == TRUE) df$Factor = factorComun
  }
  df <- data.frame(df)
  colnames(df)[1:nCol] <- paste("V", 1:nCol, sep = "")
  df <- round((df * std) + means, 2)
  return(df)
}
```

La función `simDF()` puede generar un dataframe de ítems sin correlacionar o bien correlacionados con un peso factorial determinado. Por defecto genera puntuaciones normales con media 0 y desviación típica 1, pero puede cambiarse la escala.

Podemos comprobar que la función genera los conjuntos de datos deseados:

```
#20 items con un peso factorial de 0.5, 10000 sujetos.
df <- simDF(10000, 20, lambda = 0.5, means = 100, std = 15,
            includeFactor = TRUE)
correlaciones <- cor(df)
correlaciones[correlaciones > 0.999] = NA
quantile(correlaciones[1:10], na.rm = TRUE)
```

```
##        0%       25%       50%       75%      100%
## 0.2313034 0.2343724 0.2408533 0.2438289 0.2492706
```

```
#Correlacion media de los items con el factor
mean(correlaciones[,"Factor"], na.rm = TRUE)
```

```
## [1] 0.4948692
```

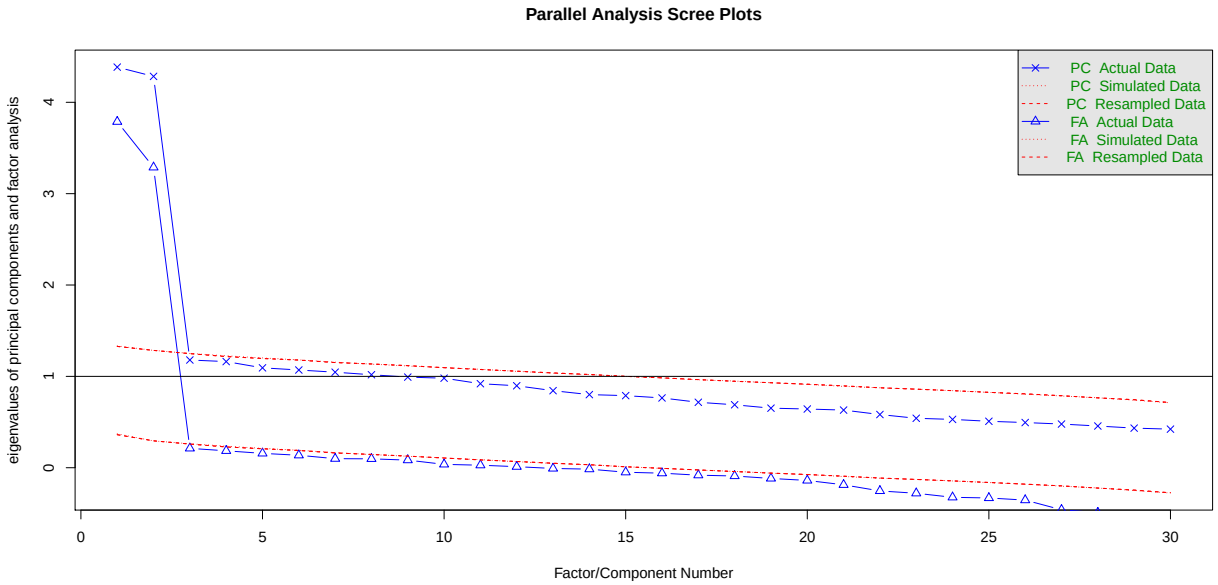
Podemos combinar varios dataframes para generar uno con varios factores y ruido.

```
nSujetos = 1000; nItems = 10
Datos <- list(F1 = simDF(nSujetos, nItems / 2, lambda = 0.5,
              includeFactor = TRUE))

#Items del primer factor con peso factorial mayor
Datos$F1_2 <- simDF(nSujetos, nItems / 2, lambda = 0.7,
                    existingFactor = Datos$F1$Factor)
#Otro factor
Datos$F2 <- simDF(nSujetos, nItems / 2, lambda = 0.5,
                  includeFactor = TRUE)
#Items del segundo factor con peso factorial mayor
Datos$F2_2 <- simDF(nSujetos, nItems / 2, lambda = 0.7,
                    existingFactor = Datos$F2$Factor)
#Ruido
Datos$Ruido <- simDF(nSujetos, nItems, lambda = 0)

df <- do.call(cbind, Datos)

SoloItems <- df[, !grepl("Factor", colnames(df))]
#Realizamos un análisis paralelo para ver si recuperamos el
#numero original de factores
paralelo <- fa.parallel(SoloItems)
```



El análisis paralelo indica que dos factores son claramente suficientes para explicar la estructura de los datos. Coincide con el número que hemos introducido, vemos que el hecho de incluir ítems no correlacionados no provoca que la técnica considere que hay motivos para introducir un tercer factor.

No obstante, pese a que el número de factores es el correcto, es posible que no agrupe a cada item en su factor correcto. Para comprobar ésta cuestión, podemos representar los pesos de cada ítem en cada uno de los factores.

```
#Comprobamos que los items pertenecen al factor correcto
Factorial <- fa(SoloItems, nfactors = 2)

Temp <- round(data.frame(unclass(Factorial$Structure)), 2)
Temp$Item <- rownames(Temp)
Temp <- Temp[, c("Item", "MR1", "MR2")]
rownames(Temp) <- NULL

Temp2 <- cbind(
  Temp[grepl("F1", Temp$Item), ],
  Temp[grepl("F2", Temp$Item), ],
  Temp[grepl("Ruido", Temp$Item), ]
)
```

En la tabla siguiente podemos ver que cada ítem está donde le corresponde[1] y con el peso correcto.[2] Podemos ver también que los items aleatorios tienen pesos muy bajos en ambos factores.

```
kable(Temp2)
```

Item	MR1	MR2	Item	MR1	MR2	Item	MR1	MR2
F1.V1	0.51	0.03	F2.V1	0.02	0.49	Ruido.V1	-0.02	-0.01
F1.V2	0.51	0.03	F2.V2	0.00	0.46	Ruido.V2	-0.02	0.00
F1.V3	0.49	0.02	F2.V3	0.02	0.45	Ruido.V3	-0.02	0.01
F1.V4	0.53	0.04	F2.V4	0.06	0.47	Ruido.V4	-0.02	-0.03
F1.V5	0.47	0.02	F2.V5	0.01	0.52	Ruido.V5	-0.01	0.07
F1_2.V1	0.73	0.04	F2_2.V1	0.03	0.68	Ruido.V6	-0.11	-0.04
F1_2.V2	0.70	0.01	F2_2.V2	0.02	0.70	Ruido.V7	-0.01	0.05
F1_2.V3	0.70	0.05	F2_2.V3	0.05	0.71	Ruido.V8	-0.03	0.02
F1_2.V4	0.71	0.07	F2_2.V4	0.05	0.70	Ruido.V9	0.04	0.04
F1_2.V5	0.69	0.00	F2_2.V5	0.03	0.69	Ruido.V10	0.00	-0.02

Por tanto, queda patente que el código de generación de los datos simulados funciona correctamente.

[1] El factor 1 del factorial no tiene porqué ser el primer factor simulado. Lo importante es que todos los items de cada factor simulado tengan pesos altos en el mismo factor

[2] Observamos que los items que pertenecen al primer grupo para cada factor tienen pesos cercanos a 0.5, mientras que los de los segundos grupos están cerca de 0.7

Para evaluar si el método de análisis factorial que hemos realizado es correcto, podemos calcular las diferencias entre los pesos factoriales reales y los pesos recuperados por la técnica. En condiciones reales no disponemos de el valor del peso real, pero en la simulación sí ya que lo establecemos nosotros mismos.

Podemos calcular ésta diferencia para los datos generados en el apartado anterior, encapsulando todo el proceso en una función para poder repetirlo.

```
PesosReales <- list(F1 = 0.5, F1_2 = 0.7, F2 = 0.5,F2_2 = 0.7)
compruebaRecuperacion <- function(items,
                                  nFactores,
                                  pesosReales,
                                  rotacion = "oblimin",
                                  metodFact = "minres"){
  #Esta funcion devuelve un vector de diferencias entre los
  #pesos recuperados y los originales (al cuadrado)
  #items: dataframe que contiene solo los items
  Factorial <- fa(items, nfactors = nFactores,
                  rotate = rotacion, fm = metodFact)
  Temp <- data.frame(unclass(Factorial$Structure))
  Temp$Item <- rownames(Temp)
  Temp$Grupo <- sub("\\..*", "", Temp$Item)
  Temp <- Temp[Temp$Grupo %in% names(pesosReales), ]
  Temp2 <- sapply(Temp$Grupo, function(x) pesosReales[x])
  Temp$PesoReal <- unname(unlist(Temp2))
  Temp$Grupo <- sub("\\_.*", "", Temp$Grupo)
  Temp <- Temp[!Temp$Grupo %in% c("Ruido", "ruido"), ]

  for(i in 1:nFactores){
    nombreVar <- paste("F", i, sep = "")
    Temp[nombreVar] = abs(Temp$PesoReal - Temp[, i])
  }

  #Averiguamos el factor en el que tiene mas peso cada item
  Temp2 <- Temp[, (ncol(Temp) - nFactores + 1):ncol(Temp)]

  seleccinaCol <- function(index){
    return(Temp[index, Temp$Grupo[index]])
  }

  Temp$pesoRecup <- apply(Temp2, MARGIN = 1, min)
  # Temp$pesoRecup <- sapply(1:nrow(Temp), seleccinaCol)
  Temp$errorRecup <- Temp$pesoRecup ^ 2

  #Tenemos que resumir si la recuperacion ocurrio correctamente
```

```
  #en una cifra. Aplicamos una penalizacion por cada item
  #que no ha sido asignado al factor correcto
  Temp2 <- data.frame(Temp2 == Temp$pesoRecup)
  Temp2$Grupo <- Temp$Grupo
  Temp2 <- lapply(unique(Temp2$Grupo),
         function(x) colSums(Temp2[Temp2$Grupo == x,
                                   1:nFactores]))
  Temp2 <- sapply(Temp2,
                  function(x) max(x / sum(x)))
  Penalizacion <- (1 / mean(Temp2)) ^ 2
  Temp$errorRecup <- Temp$errorRecup * Penalizacion

  return(Temp$errorRecup)
}

#Error de recuperacion medio
mean(compruebaRecuperacion(SoloItems, 2, pesosReales = PesosReales))
```

```
## [1] 0.0004539481
```

Vemos que en nuestro caso el error de recuperacion es muy pequeño. Una vez que tenemos las funciones necesarias creadas, podemos empezar a realizar las simulaciones.

Calidad de la recuperación

En función de lambda

Podemos evaluar la calidad de la recuperación en función del valor de los pesos simulados. La hipótesis es que a medida que disminuye el peso simulado, la recuperación tendría que ser peor. Simulamos una estructura con 5 factores.

```
simulacionLambda <- function(index,
                             returnMean = TRUE){
  #returnMean: si vale TRUE, la funcion retorna la recuperacion
  #media para cada valor de lambda. Si vale FALSE, retorna el
  #vector de resultados sin promediar
  Resultados <- c()
  for(j in 1:length(nMuestra)){
    set.seed(semillas[[index]][j])
    Datos <- as.list(1:nFactores)
    Datos <- lapply(Datos,
                    function(x) simDF(nMuestra[j], 10,
                                      lambda = pesos[index]))
    lambdas <- as.list(rep(pesos[index], nFactores))
    names(Datos) <- paste("F", 1:nFactores,
                          sep = "") -> names(lambdas)

    Datos <- do.call(cbind, Datos)
    Temp <- compruebaRecuperacion(Datos, nFactores = nFactores,
                                  pesosReales = lambdas)
    Resultados <- c(Resultados, mean(Temp))
  }

  if(returnMean){
    return(mean(Resultados))
  } else {
    return(Resultados)
  }
}

nFactores <- 5
pesos <- seq(0.1, 0.9, 0.1)
nMuestra <- rep(1000, 50) #Tamaños muestrales. En éste caso lo
#mantenemos constante para repetir el procedimiento 10 veces
#Generamos las semillas de antemano
set.seed(111)
semillas <- as.list(1:length(pesos))
```

```
semillas <- lapply(semillas, function(x) sample.int(100000000,
                                                    length(nMuestra)))
rm(.Random.seed)
library(parallel)
Ncores <- round(detectCores()/2, 0)
if(Ncores < 1) Ncores = 1
#Creamos un cluster de procesamiento
clust <- makeCluster(Ncores, type = "FORK")

#Aplicamos la funcion en paralelo
Resultados <- parSapply(clust,
                        1:length(pesos),
                        simulacionLambda)
stopCluster(clust) #Detenemos el cluster
```

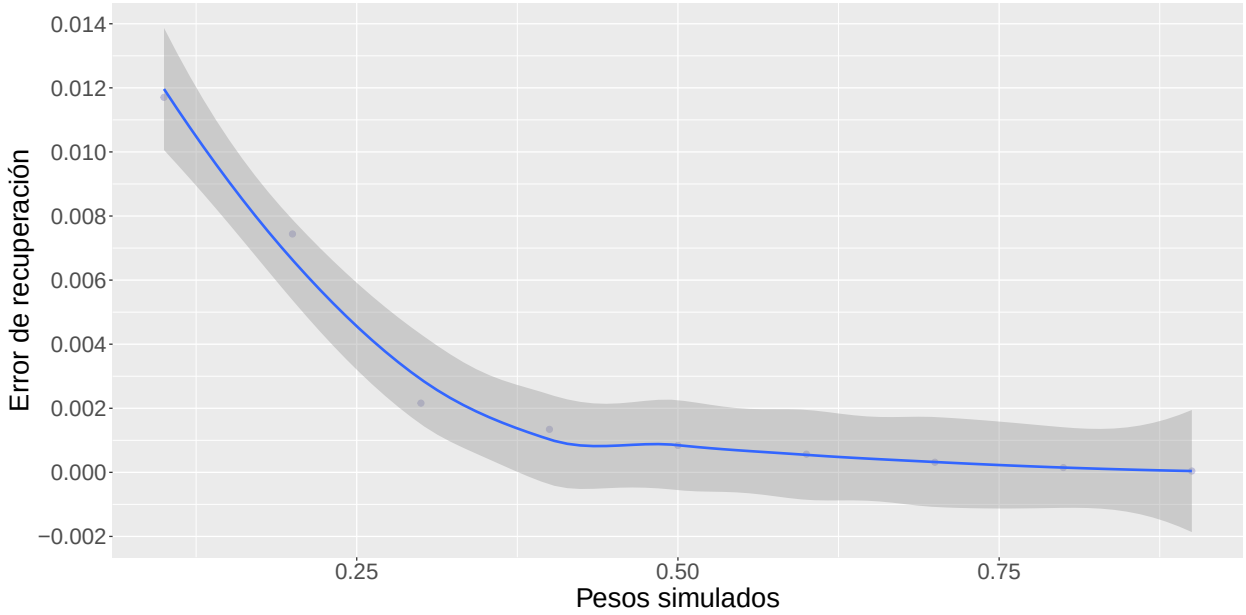
La simulación anterior consiste en ir generando conjuntos de datos de n factores. La simulación recorre un vector de pesos (λ), que serán los pesos factoriales de los ítems en el factor correspondiente. Para cada valor de λ, la función permite ejecutar el codigo varias veces variando el tamaño muestral.

En éste caso, mantenemos el tamaño muestral constante para ver únicamente el efecto del peso factorial sobre la recuperación. Para cada valor de λ, la simulación se ejecuta 10 veces, así podemos obtener la recuperación media.

En el siguiente gráfico podemos ver que efectivamente **los errores de recuperación aumentan considerablemente a medida que se reducen los pesos simulados**, especialmente para valores de λ inferiores a 0.3. En cualquier caso, para nuestra simulación, la recuperación es excelente.

```
DatosGrafico <- data.frame(
  Peso = pesos,
  ErrorRecuperacion = Resultados
)

ggplot(DatosGrafico, aes(x = Peso, y = Resultados)) +
  geom_point(alpha = 0.2,
             size = 2,
             color = "darkblue") +
  geom_smooth(method = 'loess', formula = "y ~ x") +
  xlab("Pesos simulados") +
  ylab("Error de recuperación") +
  scale_y_continuous(breaks = scales::pretty_breaks(n = 10)) +
  theme(text = element_text(size = PlotFont))
```



Éste resultado tiene sentido, ya que si la estructura correlacional es más débil la técnica debería tener más dificultades para identificarla con precisión.

En función del tamaño muestral

A continuación, podemos utilizar el código del apartado anterior para realizar la simulación modificando tanto el valor de lambda como el tamaño muestral, así podremos comprobar el efecto de ambos elementos sobre la calidad de la recuperación. La hipótesis inicial con respecto al tamaño muestral es que si cae por debajo de un valor determinado la recuperación sufrirá bastante.

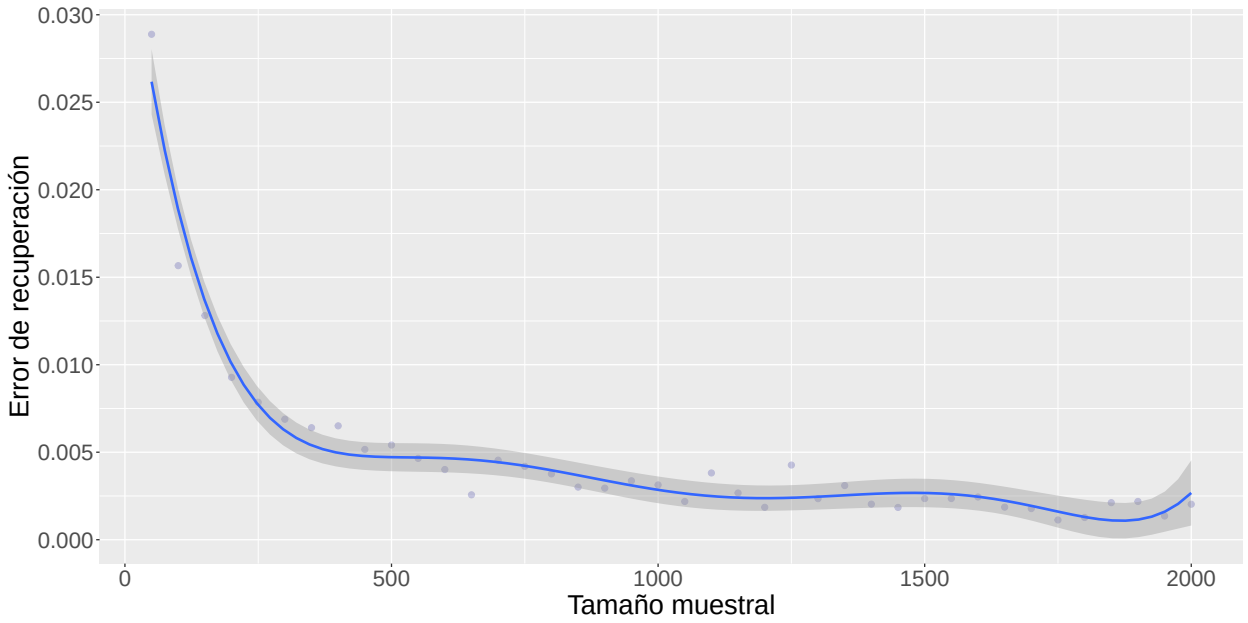
```
nFactores <- 5
pesos <- seq(0.1, 0.9, 0.1)
nMuestra <- seq(50, 2000, 50)
set.seed(111)
semillas <- as.list(1:length(pesos))
semillas <- lapply(semillas, function(x) sample.int(100000000,
                                                    length(nMuestra)))
rm(.Random.seed)

clust <- makeCluster(Ncores, type = "FORK")

#Aplicamos la funcion en paralelo
Resultados <- parSapply(clust,
                        1:length(pesos),
                        simulacionLambda,
                        returnMean = FALSE)
```

```r
stopCluster(clust) #Detenemos el cluster

DatosGrafico <- data.frame(
  Muestra = 1:nrow(Resultados) * 50,
  ErrorRecuperacion = rowMeans(Resultados)
)
```



Observamos que **a medida que aumenta el tamaño muestral el error disminuye notablemente**. La disminución del error es especialmente pronunciada al principio del gráfico, mientras que a partir de unos 500 sujetos el error comienza a decrecer más lentamente.

En la simulación, incluso en las peores circunstancias, mantenemos unas recuperaciones excelentes. No obstante, en condiciones reales la recuperación será mucho peor.

Con 100 sujetos, el error es 2.89 veces mayor que con 500 sujetos. Dado que en la simulación las recuperaciones son casi perfectas, al multiplicar por 2.89 el error cometido con 500 sujetos seguimos obteniendo un valor muy bueno, pero en aplicaciones reales con 500 sujetos podríamos cometer un error del 3 % por ejemplo. En éste caso, con 100 cometeremos un error del 8.67 %, que es ya intolerable.

Interacción

Para comprobar si el efecto de interacción entre lambda y el tamaño muestral es significativo, podemos ajustar un modelo de regresión que intente predecir el error de recuperación a partir de ambas variables y su interacción.

```
res <- data.frame(Resultados)
colnames(res) <- paste("lambda", pesos, sep = "")
res$n <- nMuestra
#Pasamos los datos a formato long
resLong <- reshape2::melt(res, id.vars = "n")
resLong$variable <- as.numeric(gsub("lambda", "",
                                    resLong$variable))
colnames(resLong) <- c("n", "lambda", "errorRecuperacion")

Modelo <- lm(errorRecuperacion ~ log(n) * lambda, data = resLong)
Modelo$coefficients #Coeficientes no estandarizados
```

```
##   (Intercept)        log(n)        lambda log(n):lambda
##    0.21723667   -0.03699935   -0.17881737    0.02712147
```

Encontramos que todos los efectos son significativos, incluido el de interacción, pero éste último modifica poco los pronósticos.

Si ajustamos el modelo sin el efecto de interacción,[3] podemos interpretar los coeficientes estandarizados para evaluar la importancia relativa de cada una de las dos variables.

Para `log(n)`, el coeficiente estandarizado vale -0.740 (-0.619 si no tomamos el logaritmo), mientras que para `lambda` vale -0.346. Ambos son negativos, por lo que valores altos en ambas variables llevan a valores más pequeños para el error de recuperación, teniendo el tamaño muestral un efecto mayor.

El R^2 ajustado del modelo es 0.506. Quizás no es lo suficientemente alto como para utilizarlo para hacer pronósticos.

[3] Utilizamos la función `lm.beta()` del paquete QuantPsyc. No permite utilizar modelos con efectos de interacción.

KMO (factorizabilidad de la matriz de correlaciones)

En función de lambda

KMO es una técnica que a través del análisis de las correlaciones parciales nos permite estudiar si la matriz de correlaciones reune algunos de los requisitos necesarios para poder aplicar análisis factorial sobre ella.

Podemos estudiar que sucede con los valores del indicador a medida que disminuye el peso simulado de los ítems en el factor. La hipótesis inicial al respecto es que el MSA mejorará a medida que aumentan los pesos simulados.

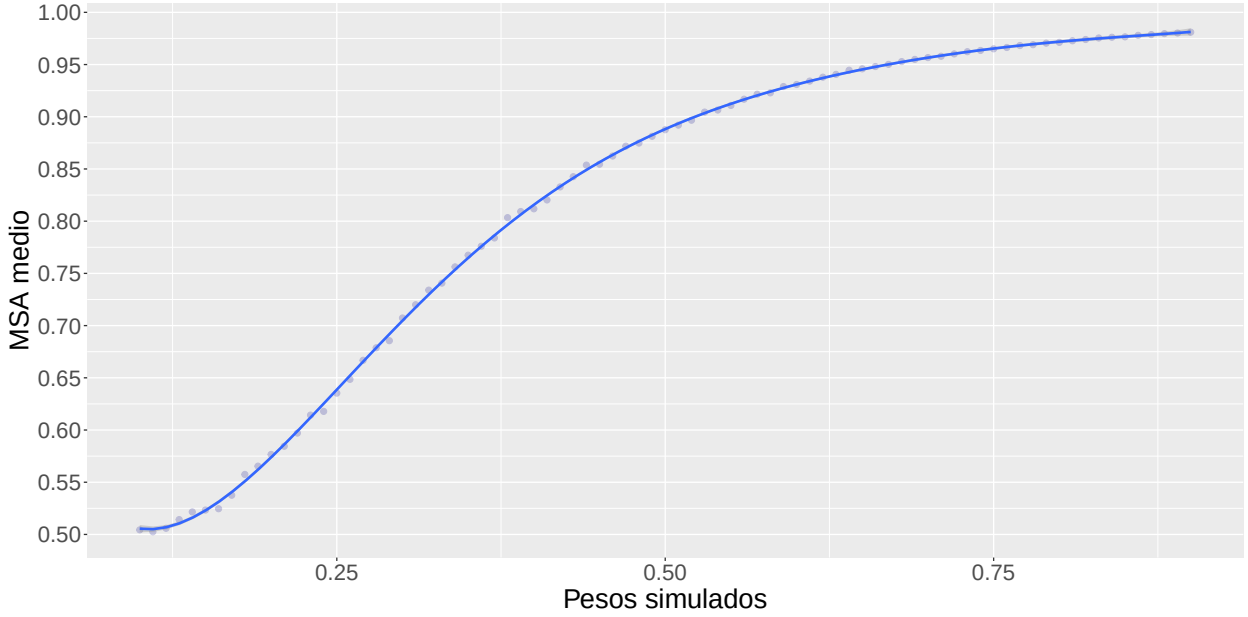
```
simulacionKMO <- function(index){
  #Generamos varios conjuntos de datos con el peso dado y
  #devolvemos el valor de MSA medio obtenido en las iteraciones
  Resultados <- 1:length(nMuestra)
  for(i in 1:length(nMuestra)){
    set.seed(semillas[[index]][i])
    Datos <- simDF(nMuestra[i], 10, lambda = pesos[index])
    Resultados[i] = KMO(Datos)$MSA
  }
  return(Resultados)
}

pesos <- seq(0.1, 0.9, 0.01)
nMuestra <- rep(1000, 20) #Mantenemos el n constante
set.seed(111)
semillas <- as.list(1:length(pesos))
semillas <- lapply(semillas, function(x) sample.int(100000000,
                                                    length(nMuestra)))
rm(.Random.seed)

clust <- makeCluster(Ncores, type = "FORK")
Resultados <- parSapply(clust,
                        1:length(pesos),
                        simulacionKMO)
stopCluster(clust)
```

```r
DatosGrafico <- data.frame(
  PesosSimulados = pesos,
  KMOmedio = colMeans(Resultados)
)

ggplot(DatosGrafico, aes(x = PesosSimulados, y = KMOmedio)) +
  geom_point(alpha = 0.2,
             size = 2,
             color = "darkblue") +
  geom_smooth(method = 'lm', formula = "y ~ poly(x, 6)") +
  xlab("Pesos simulados") +
  ylab("MSA medio") +
  scale_y_continuous(breaks = scales::pretty_breaks(n = 10)) +
  theme(text = element_text(size = PlotFont))
```



La simulación genera varios conjuntos de datos con cada uno de los pesos y a continuación calcula el MSA global sobre cada uno de los ficheros generados. Entonces almacena el resultado en un vector, que es recogido en el objeto Resultados. Despues hacemos la media de cada uno de éstos vectores para obtener el MSA medio obtenido en las distintas iteraciones de cada peso.

Observamos que **a medida que aumenta el valor de los pesos simulados mejora el valor del MSA**. En un primer momento ejecuté la simulación con pesos que iban de 0.1 a 0.9 en incrementos de 0.1 y obtuve una curva muy semejante a la que aparece representada. Vi que para el peso 0.1 el MSA medio oscilaba en torno a 0.55.

Probé a empezar con valores más pequeños para los pesos, esperando obtener valores para el MSA próximos a 0, pero encontré que, tal y como se puede observar en el gráfico, el valor no baja de 0.5. Entonces realicé unas cuantas iteraciones manualmente con un peso

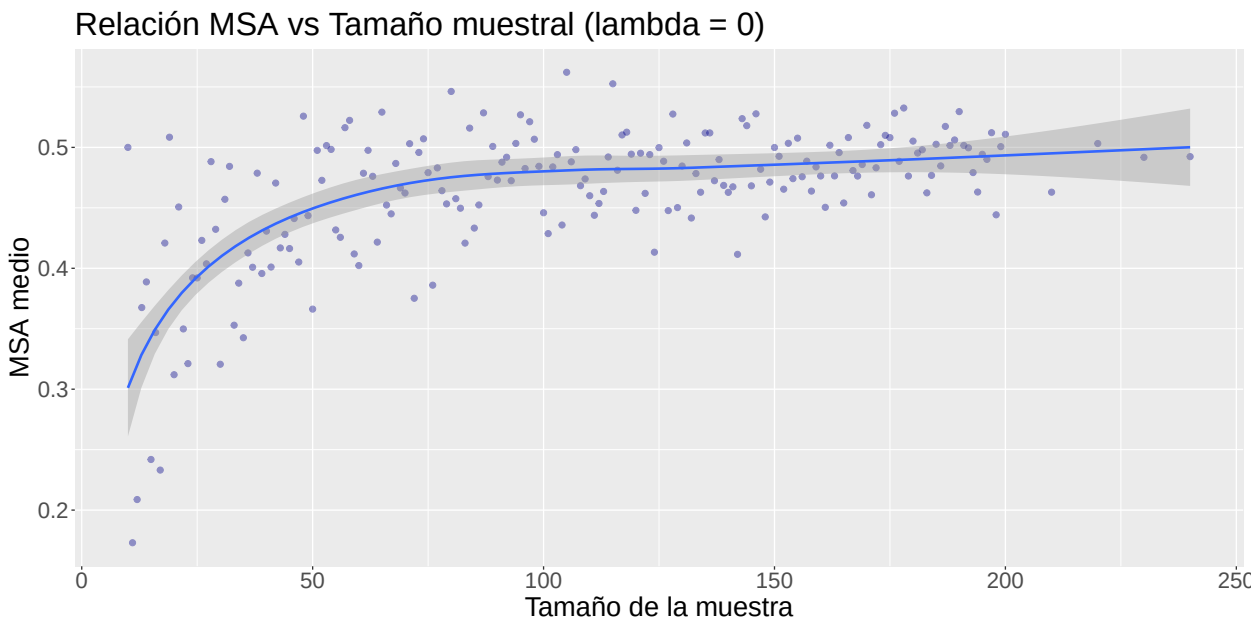
de 0 (recordemos que en éste caso la función genera datos aleatorios sin correlacionar), y obtuve que efectivamente el valor no baja de 0.5.

En función del tamaño muestral

Podemos adaptar el código de la simulación anterior para que varíe también el tamaño muestral. Así podremos estudiar la frontera inferior del MSA para un determinado tamaño muestral observando cómo evoluciona el índicador ante un conjunto de ítems no correlacionados. La hipótesis es que a medida que se va reduciendo el tamaño muestral KMO irá acercándose más a cero.

```
pesos <- seq(0, 0.9, 0.1)
nMuestra <- c(10:200, seq(210, 1000, 10))
set.seed(111)
semillas <- as.list(1:length(pesos))
semillas <- lapply(semillas, function(x) sample.int(100000000,
                                                    length(nMuestra)))
rm(.Random.seed)
clust <- makeCluster(Ncores, type = "FORK")
Resultados <- parSapply(clust,
                        1:length(pesos),
                        simulacionKMO)
stopCluster(clust)

DatosGrafico <- data.frame(
  n = nMuestra,
  MSA = Resultados[, 1]
)
DatosGrafico <- DatosGrafico[DatosGrafico$n < 250, ]
ggplot(DatosGrafico, aes(x = n, y = MSA)) +
  geom_point(alpha = 0.4,
             size = 2,
             color = "darkblue") +
  geom_smooth(method = "loess", formula = "y ~ log(x)") +
  xlab("Tamaño de la muestra") +
  ylab("MSA medio") +
  ggtitle("Relación MSA vs Tamaño muestral (lambda = 0)") +
  theme(text = element_text(size = PlotFont))
```



Si bien es cierto que **para tamaños muestrales pequeños, el MSA adopta valores más bajos**, encontramos que la relación es mucho más brusca de lo que esperaba inicialmente. Parece que la relación entre el MSA y el tamaño de la muestra sigue un patrón logarítmico.

Al principio, el MSA aumenta de forma drástica cuando aumentamos el tamaño muestral. En cambio, pasados los 100 participantes a penas hay efecto del tamaño muestral (el MSA se estabiliza en torno al valor 0.5). Quizás por éste motivo en una de las lecturas obligatorias Fabrigar et al. (1999) se menciona que según algunos autores el tamaño muestral nunca debe ser inferior a 100.

Interacción

Al representar los datos de todos los valores de lambda obtenemos una serie de curvas logarítmicas semejantes a la anterior. En el siguiente gráfico podemos ver el MSA máximo alcanzado en las iteraciones para cada valor de λ, que es aproximadamente el valor entorno al cual se estabiliza cada una de las curvas.

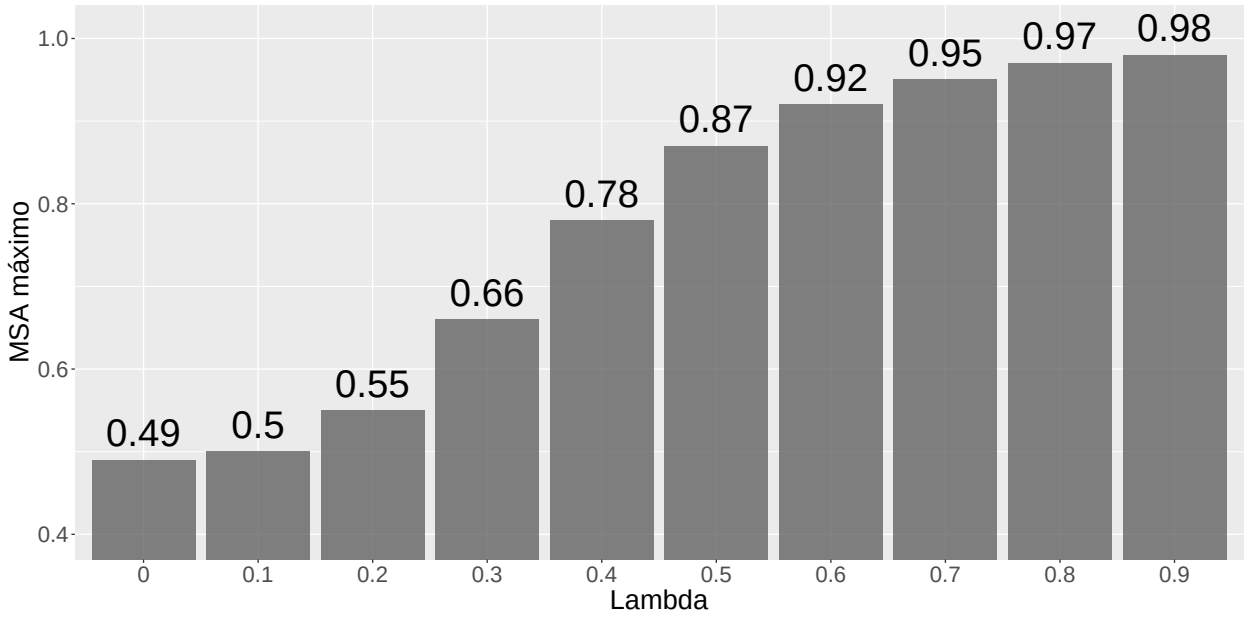
Observamos que en el medio del gráfico la diferencia entre los distintos niveles de λ es mayor que en los extremos. A penas hay diferencia entre λ 0 y 0.1, como tampoco la hay entre 0.8 y 0.9.

```
res <- data.frame(Resultados)
colnames(res) <- paste("lambda", pesos, sep = "")
Limite <- tail(res, 100)
res$n <- nMuestra

DatosGrafico <- data.frame(round(colMeans(Limite), 2))
DatosGrafico$labels <- gsub("lambda", "", row.names(DatosGrafico))
```

```r
colnames(DatosGrafico) <- c("MSA", "lambda")

ggplot(DatosGrafico, aes(y = MSA, x = lambda)) +
  geom_bar(position = "dodge", stat = "identity",
           alpha = 0.75) +
  geom_text(aes(label = MSA, x = lambda),
            stat = 'identity', position = position_dodge(0.9),
            vjust = -0.5, size = PlotFont / 2) +
  coord_cartesian(ylim = c(0.4, 1.01)) +
  xlab("Lambda") +
  ylab("MSA máximo") +
  theme(text = element_text(size = PlotFont))
```



Para comprobar si existe efecto de interacción entre el tamaño muestral y el valor de lambda, podemos ajustar un modelo de regresión. Dicho modelo intentará predecir el valor del MSA a partir del tamaño muestral y del lambda simulado contemplando la posibilidad de interacción.

```r
#Pasamos los datos a formato long
resLong <- reshape2::melt(res, id.vars = "n")
resLong <- resLong[resLong$n <= 200, ]
resLong$variable <- as.numeric(gsub("lambda", "",
                                    resLong$variable))
colnames(resLong) <- c("n", "lambda", "MSA")
z <- as.data.frame(scale(resLong))
Modelo <- lm(MSA ~ log(n) * lambda, data = resLong)
Modelo$coefficients #Coeficientes no estandarizados
```

```
##   (Intercept)         log(n)         lambda log(n):lambda
##    0.10257388    0.06872288    0.46138782    0.04047127
```

Obtenemos que **todos los efectos son significativos**, incluido el de interacción. El efecto de interacción produce un incremento significativo en el poder predictivo comparado con un modelo en el que sólo consideramos los efectos simples. Ademas, recordemos que la relación entre el tamaño muestral y el valor del MSA sigue un patrón logarítmico. El incorporar éste elemento también contribuye de forma significativa al poder predictivo.

El modelo final tiene un R^2 ajustado de 0.886, por lo que con éstas dos variables podemos explicar casi el 90 % de la variabilidad del MSA.

Podemos utilizar éste modelo para hacer pronósticos sobre el valor que adoptará el MSA dado un tamaño muestral determinado y los pesos factoriales que hipotéticamente estarán tras nuestros datos. Por ejemplo, para un tamaño muestral de 500 y un peso factorial de 0.3 estima un MSA de 0.744.

Si ajustamos el modelo sin el efecto de interacción podemos extraer los coeficientes estandarizados para evaluar la importancia de cada variable. Para la variable `log(n)`, el valor de dicho coeficiente es 0.301 (0.254 si no tomamos el logaritmo). Para lambda, el coeficiente es 0.89. Ambos son positivos, por lo que valores altos para el tamaño muestral y lambda llevan a valores mayores para el MSA, teniendo lambda un efecto mayor.

Conclusiones

Las simulaciones realizadas permiten evaluar el efecto de los pesos factoriales simulados (λ) y del tamaño muestral sobre la calidad de la recuperación de los pesos factoriales y sobre el valor del MSA al aplicar KMO.

Para la calidad de la recuperación encontramos que ambas variables tienen un efecto significativo y de signo negativo. Valores altos en ambas variables tienden a reducir el error de recuperación.

El modelo de regresión lineal muestra que existe un efecto de interacción significativo y de signo positivo, aunque bajo en magnitud. Valores altos en ambas variables generan un efecto mayor que la suma de los mismos. El R^2 del modelo resultante no es lo bastante alto como para poder utilizarlo para realizar estimaciones.

Para el valor de MSA, igualmente ambas variables tienen un efecto significativo, en éste caso de signo positivo. Por tanto, valores altos en ambas tienden a generar valores más altos para el MSA.

Encontramos también que al ajustar un modelo de regresión que contempla el efecto de interacción entre ambas variables éste es significativo y de signo positivo (de magnitud algo superior que el del anterior modelo pero moderada).

El R^2 ajustado de éste modelo es excelente, de modo que podemos usar la siguiente ecuación de regresión para hacer pronósticos sobre el MSA:

$$\hat{MSA} = 0{,}1025 + 0{,}0687 \cdot log\ N + 0{,}4613 \cdot \lambda + 0{,}0404 \cdot log\ N \cdot \lambda$$

Donde:

- $\hat{MSA}$: estimación del MSA.
- $log\ N$: logaritmo del tamaño muestral que esperamos ser capaces de reunir.
- λ: pesos hipotéticos de nuestros ítems en los factores subyacentes. Podemos basarnos en teorías y resultados existentes para saber qué valores podemos esperar.

Limitaciones

Las simulaciones realizadas tienen algunas limitaciones. En primer lugar, la función utilizada para evaluar la recuperación de los pesos factoriales tiene un inconveniente importante: para poder representar los resultados era conveniente que resumiese la recuperación en una única cifra (o en un vector de valores).

Sin embargo, la calidad de la recuperación tiene muchos componentes que no son puramente cuantitativos y que por tanto no han podido ser capturados. Por ejemplo, en casos reales es imprescindible preguntarse si las agrupaciones generadas tienen sentido a nivel teórico.

Sería conveniente realizar las simulaciones evaluando también los resultados de aplicarlas sobre conjuntos de ítems con factores subyacentes correlacionados, para evaluar el efecto que esto tiene en la recuperacion, pero ésto no se ha llevado a cabo por cuestiones de espacio.

La simulación relacionada con el MSA arrojó unos resultados inesperados. Al aplicar KMO sobre una muestra de ítems al azar esperábamos encontrar valores próximos a cero, pero encontramos un valor de 0.5, que según muchos autores es un valor que empieza a ser aceptable (Dziuban and Shirkey 1974).

Se comprobó que el código de generación de los datos funcionaba correctamente (las correlaciones entre todas las variables eran próximas a cero), y aún así se obtuvo éste resultado.

A priori no tiene sentido que la técnica nos indique que podría ser útil el análisis factorial para unos datos con éstas características al no haber a penas correlaciones en los mismos. Quizás entiende que los ítems al azar se pueden agrupar en torno a un único factor.

Sería conveniente explorar qué factores pueden hacer que el indicador descienda más todavía.

Referencias

Dziuban, C, and E Shirkey. 1974. “When Is a Correlation Matrix Appropriate for Factor Analysis? Some Decision Rules.”

Fabrigar, L., R. MacCallum, D. Wegener, and E Strahan. 1999. “Evaluating the Use of Exploratory Factor Analysis in Psychological Research.”