ACP en grandes volúmenes de datos

Juan David Ceballos Ayala
Fiorella Jessica Vanessa Tapia Pizo

Universidad del Valle
Facultad de Ingeniería, Escuela de Estadística
Santiago de Cali, Colombia
2017

ACP en grandes volúmenes de datos

Juan David Ceballos Ayala
Fiorella Jessica Vanessa Tapia Pizo

Trabajo de grado presentada(o) como requisito parcial para optar al título de:
Profesional en Estadística

Director(a):
Victor Manuel Gonzales Rojas, Ph.D.

Universidad del Valle
Facultad de Ingeniería, Escuela de Estadística
Santiago de Cali, Colombia
2017

A mi familia....

Gracias por el apoyo incondicional que me han dado en cada uno de los pasos para alcanzar mis metas.

Agradezco a mi madre Lorena Ayala....

quién además de ser parte de mi corazón, su dedicación ha sido y sigue siendo hasta hoy una meta digna de alcanzar. Agradezco también a mi compañera Fiorella Tapia, ya que su colaboración ha hecho que gran parte de este proyecto sea posible

Resumen

El análisis a grandes volúmenes de datos o matrices de gran dimensionalidad gana importancia en la actualidad gracias al desarrollo de las nuevas tecnologías de información; el presente trabajo tiene por objetivo abarcar dicha problemática teniendo en cuenta el reciente computo en paralelo y una técnica estadística de tipo descriptivo del análisis multivariado, aplicada a variables cuantitativas continuas, específicamente se hará uso del análisis de componentes principales mediante un algoritmo vía NIPALS. La propuesta ofrece como resultado un algoritmo que sirve de insumo principalmente a las ciencias de la computación con el fin de implementar el análisis de componentes principales integral y procesado en paralelo a futuros proyectos. Se verificarán las propiedades intrinsecas del ACP en cada uno de los pasos y a través de los resultados se presentará la comparación de este método con algunos otros, mostrando que en efecto es un método más eficiente y que a pesar del poder de procesamiento que tienen hoy día la computación, este método puede resultar una mejor manera de análisis a grandes volumenes de datos.

Palabras clave: Grandes volúmenes de datos, ACP, algoritmo NIPALS..

Abstract

The analysis of large volumes of data or high dimensional matrices is gaining importance nowadays thanks to the development of new information technologies; the present work aims to cover this problem taking into account the recent analysis and a descriptive statistical technique of multivariate analysis, applied to continuous quantitative variables, specifically used the principal component analysis through an algorithm via NIPALS. The proposal offers an algorithm that serves mainly for computer science in order to implement the principal component analysis and processed in parallel to future projects. The intrinsic properties of the ACP will be verified in each one of the steps and through the results of this method with others, showing that in fact it is a more efficient method and that at the same time of the power of processing that computers have today, this method can prove a better way of analyzing large volumes of data.

Keywords: High dimensional data, PCA, NIPALS algorithm.

Contenido

Lista de Tablas

Lista de Figuras

Capítulo 1

Introducción

El análisis de grandes volúmenes de datos ha ganado gran relevancia a medida que se desarrollan las tecnologías de información. Debido a esto las organizaciones han tenido que asumir el desafío de descubrir y analizar información con métodos y técnicas que van más allá de los métodos convencionales. Cada vez la frecuencia con que se mide y la precisión de estas mismas medidas hacen que la información disponible para analizar sea mayor. Algunos campos de aplicación se pueden ver en las transacciones bancarias, búsquedas en Internet y diagnóstico de fallos en piezas electrónicas; sin embargo no han logrado crecer y desarrollarse de la misma manera las técnicas que son implementadas para estos tipos de análisis. La minería de datos, el aprendizaje de máquina, entre otras, son técnicas que beben de la estadística y de las ciencias de la computación para dar respuesta al descubrimiento, análisis y posterior obtención de resultados.
El análisis de componentes principales (ACP) ayuda a los investigadores a comprender en una primera instancia, las posibles relaciones que existen al interior de un gran conjunto de datos así como las correlaciones presentes entre una variable de respuesta y una variable observable independiente para así ayudar a generar hipótesis sobre las variables.
Además, una de las propiedades más importantes del ACP es proporcionar una reducción óptima de dimensionalidad proyectando los datos en un subespacio ortogonal de menor dimensión capturando la mayor variación posible (Andrecut, 2009), ayudando a que el análisis de los datos sea más comprensible. Lamentablemente, el ACP al ser aplicado a conjuntos de datos de alta dimensión, ya sea que posean una gran cantidad de individuos, variables o gran cantidad de ambos, el ACP se vuelve bastante costoso de computar (Andrecut, 2009) . Pero debido a la gran utilidad que tiene el ACP, se ha requerido llevar esta técnica del análisis multivariado a los grandes volúmenes de datos y así, tratar de superar su actual alcance.

Este trabajo está enfocado en la implementación de un algoritmo vía NIPALS, puesto que es una manera factible de realizar este tipo de análisis en un conjunto de datos de alta dimensión demostrado en la investigación literaria de este trabajo. Además, se

verificará continuamente que las propiedades derivadas del ACP se cumplan con el algoritmo implementado a una base de datos simulada de gran tamaño.

1.1. Planteamiento del problema

El Análisis de Componentes Principales (ACP), técnica del análisis multivariado de datos con aplicaciones importantes en el álgebra lineal (Lay et al., 2007) es definida como una técnica de descripción estadística para la visualización aproximada de la información contenida en una tabla de datos, describe de forma simultánea la asociación existente entre variables y similitud entre individuos; es conocido además como una técnica de reducción de dimensionalidad de un conjunto de variables continuas (Morineau & Aluja, 1999). El uso extendido de la tecnología le ha permitido a la comunidad científica y a la estadística en particular adentrarse al ACP obteniendo rápidos resultados de fácil interpretación gracias a la implementación de algoritmos ACP en distintos software de tipo estadístico, tales como lo son SAS y R.

El enfoque estándar es utilizar métodos globales que para el ACP calculen las componentes principales de manera simultánea. No obstante cuando se aplica para conjuntos de datos de alta dimensión, donde el número de variables e individuos es alta, el ACP se vuelve costoso de calcular computacionalmente (Andrecut, 2009), por lo tanto para este tipo de situaciones un camino posible es utilizar un algoritmo iterativo que calcule las componentes de manera secuencial. NIPALS-PCA (Non-linear iterative partial least squares)(Hilbert & López, 2011) es el algoritmo iterativo de mayor uso, a menudo considerado como el algoritmo ACP estándar que dada su naturaleza es comúnmente utilizado para calcular únicamente las primeras componentes.

Otros intentos de realizar ACP en grandes volúmenes de datos se han desarrollado en sistemas distribuidos o en procesos paralelos. Los sistemas distribuidos se definen como una colección de computadoras separadas físicamente conectadas entre sí por una red de comunicaciones (Tanenbaum et al., 1996), en cambio, el proceso en paralelo es la ejecución de diferentes procesos en dos o más procesadores al mismo tiempo, donde estos procesos resuelven juntos un problema en su totalidad (dentro de un mismo ordenador) (Aiu.edu, 2016). A fin de implementar los programas estadísticos y de las ciencias de la computación a los sistemas distribuidos, se han desarrollado librerías como Pandas, la cual está orientada al lenguaje de programación Python y por el otro lado Spark enfocado a R (de gran popularidad en la comunidad estadística) mediante SparkR (Venkataraman et al., 2016). Hadoop ha sido desarrollado en forma exclusiva para generar procesos en simultáneo y guardar conexión tanto con Python como con R.

De acuerdo a González Rojas (2014) el ACP aplicado a una misma matriz de datos

debe ser integral (un único ACP), por lo tanto al emplear los sistemas de computo distribuidos o paralelos se debe encontrar la manera adecuada de particionar la matriz de datos. Realizar un análisis de componentes principales en cada una de las máquinas o procesadores y consolidar los resultados encontrados sería conceptualmente erróneo, ya que la contribución de las variables a cada componente principal se expresa en valores y vectores propios, por lo tanto , un análisis de varios conjuntos de datos donde no todas las variables como no todos los individuos serán analizados en conjunto darían valores y vectores propios diferentes que en la matriz de datos original. Puesto que el valor propio representa la varianza asociada con el componente principal y decrece a medida que se genera dichos componentes esta información puede estar distorsionada o puede ser muy diferente en relación a la información global (Franco & Hidalgo, 2003).

Este proyecto pretende proponer un algoritmo vía NIPALS que permita la realización de un solo ACP en grandes cantidades de datos utilizando la correcta partición de matrices, donde inicialmente se esperaba adaptar dicho algoritmo a un sistema paralelo de forma que cada uno de los procesadores o cada uno de los computadores en el caso de un sistema distribuido se encargase de cada una de las diferentes particiones de la matriz, haciendo así un método más eficiente que combine la estadística con las ciencias de la computación, sin embargo dada la dificultad técnica de adaptar este algoritmo a un sistema paralelo o distribuido, se trabajará desde una perspectiva principalmente estadística de forma que el algoritmo producido sirva de insumo a las ciencias de la computación. Una primera fase del proyecto es la revisión bibliográfica y conceptual de los intentos por hacer ACP en grandes volúmenes de datos, posteriormente se seleccionará un algoritmo que permita el análisis de componentes principales manteniendo intactas todas sus propiedades, de forma seguida este algoritmo será adaptado de forma tal que sea posible una correcta subdivisión en las tareas o al menos en parte de los procesos, de forma que se pueda implementar a matrices de alta dimensionalidad, cabe resaltar que para el interés de este proyecto se trabajará con matrices donde $n >> p$, es decir el número de individuos es mucho mayor al número de variables, puesto que el problema puede ser abarcado desde esta óptica o desde el punto donde $p >> n$ o los casos donde el número de individuos y variables son ambos similares entre sí en cuestión de tamaño y además ambos de gran dimensión, se simulará para finalizar un caso de estudio donde se aplicará el algoritmo ya desarrollado, de forma que se puedan validar la confiabilidad de los resultados presentados mediante esta nueva metodología.

1.2. Justificación

El análisis de grandes volúmenes de datos se ha convertido en un aspecto de gran relevancia debido al creciente avance tecnológico y a la gran cantidad de información disponible para ser procesada y analizada. De acuerdo a IBM (2017), solo en los últimos dos años se ha generado más del 90 % de la información existente y por otro lado la firma consultora

Gartner (2017) estima que la tasa de crecimiento anual de información en el mundo es del 59 %. En Colombia según el Departamento Nacional de planeación, sus bases de datos del año 2014 a 2015 aumentaron en un 40 %, contando así al día de hoy con cerca de 1.000 terabytes en información disponible (Gaviria, 2016). La reciente tendencia a manipular grandes volúmenes de datos es debido a la necesidad de implementar dicha información en análisis estadísticos y modelos de predicción que pueden ser de gran ayuda en análisis de negocios y publicitario (Molina & García, 2006).

El crecimiento de las tecnologías ocasiona que los métodos de recolectar información sean capaces de captar mayor cantidad de información y con más precisión, ejemplos de ello se encuentran en los dispositivos móviles y teléfonos celulares. La tecnología M2M (Machine to machine) que ha tomado valor con la llegada del Wi-fi y el GPS consiste en compartir datos con dispositivos encargados de medir magnitudes físicas donde éstas se transforman en datos.

La corriente actual de Big Data, hace que el correcto análisis de grandes cantidades de información gane importancia puesto que con la tecnología actual se obtiene información desde diversas fuentes de información y que pueden ser de tipo estructural o no estructural, los emails, las busquedas en internet, las notas de audio, entre otros pueden ser considerados como fuentes de datos no estructural, lo anterior hace realmente difícil su organización mediante bases de datos relacionales; no obstante, el ritmo al que crecen y se desarrollan las tecnologías de información es muy superior a los medios actuales que procesan información dejándolos al día de hoy obsoletos. El nacimiento de las técnicas de mineria de datos y los algoritmos de aprendizaje automático entran entre los nuevos métodos utilizados para extraer y analizar información de grandes volumenes de datos, no obstante muchos de estos métodos son considerados “cajas negras” al no haber claridad sobre la forma en que operan y al no contar con la rigurosidad matemática (propiedad de optimalidad) con que cuentan los métodos más convencionales (Rayón, 2016).

El valor real de los datos reside en cómo es utilizada la información para la toma asertiva de decisiones, en la actualidad la gran cantidad de información de la que se dispone no hace fácil obtener el máximo beneficio; se podría elegir solo una cierta fracción de la información total, pero en algunos casos, grandes cantidades de datos proporcionan información suficiente para derivar conocimiento más que adicional.
El análisis de grandes volúmenes de datos es usualmente una tarea de difícil realización, no solo por el limitado poder de procesamiento en comparación al desarrollado poder de captar información, sino también por la dificultad de observar relaciones cuando el espacio dimensional en el que están es mayor a la segunda dimensión (relaciones de uno a uno).El análisis de componentes principales ha vuelto particularmente "fácilreducir dimensionalidad con el propósito de ver relaciones existentes de la mejor que sabe el ser humano: por medio de un plano. Como técnica estadística el ACP destaca por no necesitar que los datos

cumplan con ningún supuesto distribucional, tampoco requiere la utilización de muestras, esto debido a que trabaja con la información disponible como un total y todas las posibles relaciones que se encuentren serán a partir de esta (Lebart et al., 2000), adicional a ello de acuerdo a González Rojas (2014) el ACP cumple con la condición de ser optimal en el sentido que emplea multiplicadores de Lagrange a fin de maximizar la inercia recogida por las componentes principales.

Implementar el análisis de componentes principales en grandes volúmenes de datos posibilitará por primera vez la descripción simultánea entre individuos y variables junto con un análisis descriptivo en algunos conjuntos de datos que debido a su gran extensión no se llegaban a estudiar. Ante el peso que conlleva la cantidad de datos, se debe enlazar ACP con las herramientas tecnológicas implementadas en la actualidad; utilizar un algoritmo que sea computacionalmente eficiente en cálculos, que utilice sus propiedades y características para facilitar el ACP en cantidades masivas de información y que además a futuro sea posible implementar dentro de un sistema de computación distribuido o en paralelo.

Se ha encontrado que algunos algoritmos que se usan para realizar ACP en grandes volúmenes de datos sufren de problemas de ortogonalidad en los vectores propios como el algoritmo Lanczos (Saad, 1992) o se convierten muy costosos computacionalmente como el algoritmo jacobi (Sameh, 1971), SVD (descomposición de valores singulares), y EVD (descomposición de valores propios)(Andrecut, 2009).

El algoritmo NIPALS (Nonlinear estimation by Iterative Partial Least Square) base de la regresión PLS que es ampliamente utilizada en las ciencias sociales y económicas, ha ganado fuerza gracias a sus beneficios, entre estos son el hecho de ser capaz de realizar un ACP ante la existencia de datos faltantes (González Rojas, 2014) y la capacidad de realizar también ACP aún cuando hay más variables que individuos; se hace sumamente importante la capacidad de utilizar el análisis de componentes principales mediante el algoritmo NIPALS en grandes cantidades de datos, logrando así describir y categorizar grandes poblaciones de individuos en estudios de todo tipo, aún cuando haya una alta proporción de datos faltantes.

Capítulo 2

Objetivos

2.1. Objetivo general

- Proponer un algoritmo que implemente el Análisis de componentes principales en grandes volúmenes de datos

2.2. Objetivos específicos

- Desarrollar un algoritmo vía NIPALS para implementar ACP en grandes volúmenes de datos.
- Establecer un caso de estudio real o simulado para implementar el algoritmo desarrollado.
- Verificar las propiedades derivadas del ACP.

Capítulo 3

Antecedentes

El análisis de componentes principales es el método más popular para la compactación de conjuntos de datos en alta dimensión a dimensiones inferiores para el análisis, visualización, extracción de relaciones, o comprensión de datos (Jolliffe, 2002).

Lamentablemente, el ACP no funciona de manera eficiente al analizar conjuntos de datos de dimensión muy alta (número de individuos y variables muy grandes). Frente a esto, se han utilizado algoritmos iterativos que calculan las componentes secuencialmente en vez de manera simultánea (Andrecut, 2009).

El algoritmo NIPALS-PCA (Wold et al., 1987) se usa cuando el número de componentes principales es de menor dimensión que el número de variables originales. Es el método más utilizado para el cálculo de las componentes principales y proporcionan numéricamente resultados más precisos en comparación con otros métodos como la técnica de descomposición de covarianza (Statsoft, 2015). Sin embargo, este algoritmo para grandes cantidades de datos o matrices con alto grado de colinealidad entre columnas, sufre de pérdida de ortogonalidad debido a acumular errores en cada paso de iteración(Kramer, 1998). Por lo tanto, el algoritmo Nipals tiende a usarse para estimar las primeras componentes.

Otro método popular para obtener raíces características y vectores propios ha sido el método de potencia. En este procedimiento, los valores y vectores propios se calculan de manera secuencial encontrando primero el valor propio más grande con su respectivo vector asociado, luego la segunda raíz más grande, y así sucesivamente. Aunque con los años este método ha sido reemplazado por procedimientos más eficientes en los paquetes estadísticos informáticos, es más sencillo, fácil de entender y de manejar que muchos métodos nuevos (Jackson, 2005).

En el artículo de Kettaneh et al. (2005) se desarrolla un enfoque multivariado basado en PCA y PLS en algunos ejemplos de problemas observados en RD&P (Investigación,

desarrollo y producción) orientado en la quimiometría (disciplina química que aplica métodos matemáticos o estadísticos sobre datos químicos), puesto que el aumento en el tamaño de estos datos comienza a crear insuficiencias en los métodos multivariables, tanto en su eficiencia como en su interpretación (Kettaneh et al., 2005).
Para el desarrollo del PCA y PLS, los autores utilizaron una técnica denominada quilt-PCA o quilt-PLS usando una idea similar a los modelos jerárquicos basados en variables latentes (componentes principales) y objetos latentes (vectores propios). Esta técnica consiste en colocar en un mismo bloque variables similares para luego realizar un ACP o modelo PLS por cada bloque; las componentes resultantes se usan como variables para la agrupación jerárquica. Esta división de variables en bloques ha demostrado que simplifica la interpretación de modelos multivariados con bastantes variables (Wold et al., 1987).

Kettaneh et al. (2005) encontraron que las complicaciones involucradas en el análisis de muchas observaciones parecen ser mayores que las de analizar muchas variables, por lo tanto, el método propuesto no se pudo realizar de manera exitosa. También, los autores observaron que algunos problemas se vuelven más graves con el tamaño creciente del conjunto de datos, especialmente el de los valores atípicos, agrupamiento y no linealidades.

Un estudio de la universidad de Calgary, Canadá, realizado por Andrecut (2009) formuló un ACP iterativo basado en el algoritmo de ortogonalización Gram-Schmidt, denominado GS-PCA. Para acelerar el procesamiento del algoritmo utilizó una aplicación paralela para GPU (unidades de procesamiento gráfico). Analizó la aplicación paralela de la GPU tanto para los algoritmos NIPALS-PCA como para GS-PCA.
Los resultados generados por este estudio muestran que los algoritmos implementados en GPU de manera paralela entregan resultados con mayor rapidez (hasta 12 veces) que implementados en la CPU. El algoritmo GS-PCA resultó aproximadamente entre el 5 % y 7 % más lento que el algoritmo NIPALS-PCA estándar, tanto en CPU como en GPU.

Halko et al. (2011a) presentan un algoritmo para el análisis de componentes principales basados en un diseño aleatorio del método de Lanczos bloque para grandes conjuntos de datos, donde a una matriz de información se construye la descomposición de valores singulares (SVD) para aproximarse lo más posible a la matriz original de datos.
El método de Lanczos es un algoritmo creado por Cornelius Lanczos, el cual es una adaptación de los métodos iterativos para encontrar los primeros valores propios y vectores propios de un sistema lineal de dimensión cuadrada realizando un número de operaciones menores al tamaño de la matriz (Lanczos, 1950).
Aunque en un principio la eficiencia computacional de Lanczos no era útil debido a su inestabilidad numérica, en 1970, Ojalvo y Newman mostraron cómo estabilizarlo usando un método para la corrección de los vectores propios a cualquier grado de precisión que, cuando no se realiza, provoca una serie errores en los vectores que se magnifican conforme el cálculo de estos procede (Newman & Ojalvo, 1970).

Con el método mejorado, Halko et al. (2011a) paralelizaron los pasos básicos del procedimiento que incluye el algoritmo Lanczos con la ayuda del procesamiento multinúcleo y distribuido. El rendimiento del algoritmo a través de varios ejemplos numéricos muestra que el método necesita solo un par de iteraciones para producir una buena precisión con una probabilidad bastante alta de aproximadamente $1 - 10^{-5}$ (Halko et al., 2011a).

Figueroa Mata et al. (2012) realizaron un estudio utilizando análisis de componentes principales en paralelo donde aplicaban el poder de cálculo de los grandes procesadores GPU'S con apoyo de CUDA (Compute Unified Device Architecture) el cual es un compilador junto con un conjunto de herramientas de desarrollo creadas por nVidia que permiten usar una variación del lenguaje de programación C para codificar algoritmos en las unidades de procesamiento gráficos GPU'S (Wikipedia, 2016).

Los investigadores utilizaron el modelo de programación CUDA para aprovechar el paralelismo que ofrecen los múltiples núcleos de las GPU'S al lanzar un gran número de hilos de forma simultánea, donde un hilo es básicamente una tarea que puede ser ejecutada en paralelo con otra tarea. El problema de determinar los valores propios de una matriz para la realización del ACP lo resolvieron a partir del método Jacobi.

El método Jacobi se planteó en 1846, sin embargo en los años 60 y 80 se dejó de usar debido a la llegada del algoritmo QR que generaba buenos resultados en máquinas secuenciales. Sin embargo, el método de Jacobi adquirió importancia por la aparición de la computación paralela (Sameh, 1971).
A principios de los 70 apareció el primer estudio del método de Jacobi en memoria compartida (Sameh, 1971) y a principios de los 80 para memoria distribuida (Brent & Luk, 1982). El método de Jacobi reducen una matriz simétrica a la forma diagonal, con los valores propios como los elementos diagonales de esta matriz. Esta reducción se realiza sin operaciones previas de tridiagonalización o a otras formas condensadas intermedias (Figueroa Mata et al., 2012).

En el 2014, Gad Abraham y Michael Inouye de la universidad de Melbourne, Australia realizaron un estudio para analizar los datos del polimorfismo de nucleótido único (SNP) en todo el genoma para detectar la estructura de la población y los valores atípicos potenciales. Sin embargo, el tamaño de los datos SNP ha ido aumentado enormemente en los últimos años y realizar un PCA en este conjunto de datos se convierte en un problema clave para el estudio de Abraham & Inouye (2014). Por consiguiente, los autores desarrollaron FLASHPCA para 150000 individuos, siendo este método 125 veces más rápido que las herramientas existentes, como los empleados por el paquete de software Eigensoft (Abraham & Inouye, 2014).

FLASHPCA es un algoritmo aleatorizado como el de Halko et al. (2011a), basado en la construcción de una pequeña matriz que captura los valores propios superiores y vectores propios con SVD. En la mayoría de las aplicaciones genómicas solo interesa algunos vectores propios de los datos, lo cual FLASHPCA hace que el costo computacional sea sustancialmente más pequeño.

En el artículo de Yu et al. (2017) proponen un algoritmo aleatorizado de paso único para calcular ACP. Este algoritmo de un solo paso tiene el beneficio de requerir solo una "pasada"sobre los datos, es decir, hacer una sola lectura sobre estos. Además, es particularmente útil para transmitir datos o datos almacenados en una memoria lenta (Halko et al., 2011b). Los autores reconstruyeron el algoritmo bloqueado aleatorio de Martinsson & Voronin (2016), para estabilizar el algoritmo, dando como resultado el PCA de un solo paso.

El algoritmo bloqueado aleatorio en Martinsson & Voronin (2016) está inspirado en el procedimiento Gram-Schimdt para la ortonormalización de randQB (QB aleatorio), que constituye una actualización iterativa del error de aproximación QB. El randQB utiliza proyección al azar para identificar un subespacio que captura las acciones dominantes de una matriz A, produciendo una matriz que bosqueja la matriz original para facilitar el cálculo de descomposición de esta.
El algoritmo resultante se paralelizó y se puso a prueba con un conjunto de datos de alta dimensión con un peso de 150 GB, los resultados mostraron que el algoritmo fue capaz de calcular las primeras 50 componentes principales en 24 minutos con una precisión aceptable llevados a cabo en un servidor linux con dos CPUs de 12 núcleos Intel Xeon E5-2630 (2.30 GHz) y 32 GB de RAM (Martinsson & Voronin, 2016).

Capítulo 4

Marco Teórico

4.1. Grandes volúmenes de datos

El concepto de grandes volúmenes de datos es a menudo confundido con el término Big Data, donde ambos hacen referencia a cantidades masivas de información, sin embargo el primero trata de bases de datos específicamente estructuradas que contienen Terabytes o Petabytes de información, 1.000 y 1.000.000 de Gygabytes respectivamente, y que han sido derivados de investigaciones, empresas y procesos industriales que han trabajado con ellos mediante tecnologías como los Data Warehouse (almacenes de datos) que se encargan de extraer, transformar y analizar información. El segundo, trabaja con información de tipo estructurada, semi-estructurada o no estructurada y obtiene información a partir de cualquier fuente de información, como puede ser el vuelo de un avión, una nota de voz, un documento de texto y demás. Big Data no se restringe al análisis de datos, es un concepto que engloba infraestructuras, tecnologías y servicios que han sido creados para dar solución al procesamiento de enormes conjuntos de datos. (López, 2014)

Los grandes volúmenes de datos por su parte traen consigo problemas de tipo técnico-teórico, puesto que conforme aumentan los registros de individuos (filas) y variables (columnas) en una base de datos o matriz de información, la dimensionalidad de dicha matriz aumenta también. Frente a estas altas dimensiones, las técnicas estadísticas convencionales muestran una gran variedad de problemas en uso y fiabilidad sobre sus resultados, a este conjunto de problemas se les conoce "maldición de la dimensión" (Silva Mata et al., 2017), ejemplos de estos se encuentran en errores de clasificación basados en las distancias euclidianas, problemas en la recientemente desarrollada inferencia de datos funcionales (Fauzi, 2011), en la validez de la prueba t Student cuando $p >> exp(n^{1/2})$ y demás.

Con el fin de mitigar la problemática debido a las grandes dimensiones, existen un conjunto de métodos de reducción de dimensionalidad, entre los más conocidos está el análisis de componentes principales, donde existen las variantes de análisis de componentes

principales probabilístico, con funciones kernel (Hernández, 2016) y análisis de componentes principales funcional (Fauzi, 2011). A partir de estos métodos se buscan algoritmos matemáticos que mejoren tiempos de respuesta pero que, a su vez, la información intrínseca se pueda recuperar.

4.2. Valores y vectores propios

Los conceptos de valor y vector propio, también conocidos como eigenvalores y eigenvectores ó valores y vectores característicos, son conceptos fundamentales dentro del álgebra lineal considerados como una excelente herramienta para dar solución a múltiples problemas de matemáticas, física, estadística, ciencias de la computación, entre otros. En la estadística y el análisis de componentes principales (ACP) es fundamental encontrar estos.

De acuerdo a Arce et al., 2004, sea A una matriz cuadrada de tamaño n, un número real λ es un valor propio de A si existe un vector x tal que $Ax = \lambda x$, donde x es el vector propio de A que está asociado a λ y este es por lo tanto distinto de cero.

Cuando la matriz A a tratar no cumple con la condición de ser diagonalizable existe una serie de descomposiciones tal que sea posible determinar sus valores y vectores propios; incluso es posible determinarlos aun cuando la matriz A no es cuadrada. El algoritmo de análisis de componentes principales hace uso por excelencia de la descomposición singular y espectral para el cálculo de valores y vectores propios.
Dichos valores y vectores propios pueden ser calculados mediante dos formas:

1. **A través del polinomio característico**

Es demostrable, partiendo de la ecuación 4.1 que los valores propios de una matriz no son más que las raíces del polinomio característico ($p(\lambda)$) de esta, dado que un polinomio puede tener a lo sumo m raíces entonces una matriz puede tener como máximo m valores propios donde m es el grado del polinomio o rango de la matriz.

$$\begin{aligned} Ax - \lambda x &= 0 \\ (A - \lambda I)x &= 0 \end{aligned} \tag{4.1}$$

Dado que $x \neq 0$. Es equivalente tener:

$$\begin{aligned} det(A - \lambda I) &= 0 \\ p(\lambda) &= det(A - \lambda I) \end{aligned} \tag{4.2}$$

Una vez encontrados los valores propios λ es posible encontrar los respectivos vectores propios x mediante la solución del siguiente sistema de ecuaciones homogéneo:

$$Ax = \lambda x \tag{4.3}$$

2. **Por diagonalización de matrices**

El cálculo de valores y vectores propios por medio del polinomio característico suele ser difícil de hacer, por lo tanto en la práctica se utilizan métodos numéricos como el método de las potencias, el de Raleigh y el QR de Francis (Martınez & Pérez, 1990) o métodos de descomposición para matrices diagonalizables como lo son la descomposición espectral y la descomposición en valores singulares. Es demostrable que dada una matriz invertible C, los valores propios de A son iguales a los valores propios de $C^{-1}AC$, esto gracias a que ambas matrices guardan el mismo polinomio característico :

$$det(C^{-1}AC - \lambda I) = det(C^{-1}(A - \lambda I)C) = det(A - \lambda I) \tag{4.4}$$

A su vez, es posible encontrar una matriz C tal que $C^{-1}AC = D$, donde D es una matriz diagonal y por lo tanto los valores de su diagonal serán sus mismos valores propios, en este caso también los valores propios de A y dado que C es una matriz invertible, los vectores columna cumplen con ser linealmente independientes y ser los vectores propios de A. Para el caso donde la matriz A no es diagonalizable, se pueden trabajar con ciertas propiedades del álgebra lineal para encontrar sus respectivos valores y vectores propios, tal y como lo hace el ACP.

4.3. Descomposición en valores singulares

La descomposición en valores singulares o SVD (singular value decomposition) es una factorización empleada en matrices reales mediante la cual cualquier matriz puede ser expresada en términos de tres matrices, así:

$$A_{nxp} = U_{mxn}\Sigma_{nxp}V^T_{pxp} \tag{4.5}$$

De la ecuación 4.5, U es una matriz donde sus vectores columna son los vectores propios ortonormales de AA^t, Σ es una matriz diagonal que contiene en forma decreciente los valores singulares (raíz de los valores propios) de la matriz A y V es una matriz que contiene los vectores propios de la matriz A^tA.

4.4. Rango de una matriz

El rango de una matriz se refiere al número máximo de columnas (filas) que son linealmente independientes, es decir, cuando ninguna de las columnas (filas) son combinación lineal de los restantes.
El rango fila y rango columna son iguales, por lo tanto comúnmente solo se le denomina rango de A, expresada como $\rho(A)$, siendo A una matriz de tamaño n filas y p columnas. El número de columnas y filas independientes de A es igual a la dimensión del espacio columna y espacio fila de A, respectivamente. Por lo tanto, el rango debe ser un número positivo, menor o igual al mínimo entre n y p.

Algunas propiedades del rango son:

- $\rho(A)$=$\rho(A^T)$
- $\rho(A)$=$\rho(AA^T)$=$\rho(AA^T)$

Es importante destacar que el rango de una matriz es igual al número de valores propios no nulos.

4.5. Ortogonalidad

- Se dice que dos vectores u,v $\in R^n$ son ortogonales si el producto interno o producto punto entre ellos es igual a cero ($u \cdot v = u^t v = 0$)
- Se dice que un conjunto de vectores $v_1, ..., v_m$ de R^n es un conjunto ortogonal si cada uno de los vectores v_k es ortogonal a todos los demás,

$$v_k \cdot v_j = 0, \forall j \neq k$$

- Se define como bases ortogonales a las bases formadas por conjuntos ortogonales. La principal ventaja, de tener una base ortogonal de un subespacio, es que el cálculo de las coordenadas de un vector respecto de dicha base es particularmente sencillo (Kolman et al., 1981).

4.6. El proceso de Gram-Schmidt

En álgebra lineal, el proceso de Gram-Schmidt es un método que permite transformar, de manera gradual, una base cualquiera S=$\{v_1, v_1, ..., v_k\}$ para k $\leq$ n de un subespacio W de R^n a una base ortogonal del mismo citep2006algebra.

El proceso se basa en un resultado de la geometría euclidiana, el cual establece que la diferencia entre un vector v y su proyección sobre otro vector u, es perpendicular al vector u (Sullivan, 1997). Dicho resultado ayuda a construir, a partir de un conjunto de vectores no paralelos, otro conjunto, conformado por vectores perpendiculares. Este algoritmo recibe su nombre de los matemáticos Jorgen Pedersen Gram y Erhard Schmidt.

Se define al operador de proyección como:

$$proj_u(v) = \frac{< u, v >}{< u, u >} u \tag{4.6}$$

Donde $< u, v >= u^T v$ es el producto interno de los vectores u y v. Este operador proyecta el vector v ortogonalmente a la línea que abarca el vector u. Dado lo anterior, el proceso Gram-Schmidt funciona de la siguiente manera:

$$\begin{aligned}
u_1 &= v_1 & e_1 &= \frac{u_1}{\|u_1\|} \\
u_2 &= v_2 - proj_{u_1}(v_2) & e_2 &= \frac{u_2}{\|u_2\|} \\
u_3 &= v_3 - proj_{u_1}(v_3) - proj_{u_2}(v_3) & e_3 &= \frac{u_3}{\|u_3\|} \\
u_4 &= v_4 - proj_{u_1}(v_4) - proj_{u_2}(v_4) - proj_{u_3}(v_4) & e_4 &= \frac{u_4}{\|u_4\|} \\
&\vdots & &\vdots \\
u_k &= v_k - \textstyle\sum_{j=1}^{k-1} proj_{u_j}(v_k) & e_k &= \frac{u_k}{\|u_k\|}
\end{aligned}$$

Los vectores $u_1, ..., u_k$ y los vectores normalizados $e_1, ..., e_k$ conforman un conjunto de vectores ortogonales y vectores ortonormales, respectivamente.

Geométricamente, para calcular u_i el método Gram-Schmidt proyeca el vector v_i ortogonalmente en el subespacio u generado por los vectores $u_1, ..., u_{i-1}$ que es igual al subespacio generado por $v_1, ..., v_{i-1}$.

4.7. Análisis de componentes principales

El análisis de componentes principales (ACP) es una técnica de descripción estadística para la visualización aproximada de la información contenida (de tipo cuantitativo) en una tabla de datos (Aluja et al., 1999). Se puede ver también como una técnica de reducción de dimensionalidad, puesto que busca por medio de una transformación ortogonal encontrar un subespacio capaz de representar las variables originales posiblemente correlacionadas en un nuevo conjunto de variables no correlacionadas denominadas componentes principales

(Villardón, 2002).

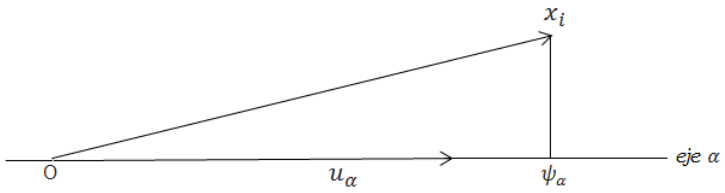
Sea la matriz de datos brutos (con las observaciones originales) B_{nxp} con n puntos-fila (nube de puntos donde cada fila representa un punto en el espacio) individuos y p puntos-columna (nube de puntos donde cada columna representa un punto en el espacio) de variables; b_{ij} corresponde a la observación del i-ésimo individuo en la j-ésima variable, así para comparar semejanzas entre individuos se miden las distancias euclidianas entre parejas de individuos variable a variable por medio de $d^2(i,i') = \sum_{j=1}^{p} m_j(b_{ij} - bi'j)^2$, tal que dos individuos están cercanos si toman valores semejantes en cada variable (González Rojas, 2014). El problema yace en observar relaciones en bases de gran dimensionalidad, ya que para este caso los individuos se encuentran representados en una nube de puntos-fila en R^n; con el objetivo de solucionar este problema el ACP busca un primer plano factorial de dimensión R^2 donde se proyecte la nube de individuos original y se conserven al máximo las características de dicha nube.

Encontrar el plano de proyección que mejor represente a la nube original de puntos-fila equivale a encontrar aquel que mayor información aporte, en términos estadísticos es igual a encontrar el plano que maximice la dispersión de los puntos, dicha dispersión se puede ver como la suma de todas las posibles distancias cuadradas entre parejas de individuos y viene dada por $\sum_i \sum_{i'} d^2(i,i')$. Es demostrable según Lebart et al. (2000) que $\sum_i \sum_{i'} d^2(i,i') =$ 2n $\sum_i d^2(i,g)$ donde g es el centro de gravedad o punto medio de la nube de puntos y sus coordenadas vienen dadas por $g = (\bar{b}_1, \bar{b}_2, ..., \bar{b}_p)$; de lo anterior se hace necesario que para el ACP los datos estén al menos centrados, donde X_{nxp} es la matriz de datos centrados y $x_{i,j} = b_{ij} - \bar{b}_j$, haciendo que las nuevas coordenadas para el centro de gravedad se encuentren en el origen $g = (0, 0, ..., 0)$. Puesto que regularmente cada una de las variables tiene sus propias unidades de medida y en el ACP todas estas deben ser representadas en un mismo plano factorial, se suele trabajar con la matriz de datos estandarizada Z, de forma que dichas unidades no afecten en el análisis final (González Rojas, 2014).

Las coordenadas de la proyección ortogonal de cada uno de los individuos sobre el vector director u_α que forma parte del nuevo subespacio se denomina componente principal del eje α y se denota ψ_α; ya que el vector u_α es aquel que verifica la transformación ortogonal de la nube de puntos-fila a un nuevo subespacio, estos vectores no son más que los vectores propios de X_{nxp}; desde el punto de vista geométrico es posible ver en la figura **4.1** un triangulo rectángulo donde de acuerdo a Aluja et al. (1999) y partiendo del teorema de Pitágoras se tiene que $\sum_i d^2(i,g) = \sum_i d^2_H(i,g) + \sum_i d^2_{\bar{H}}(i,g)$ donde $\sum_i d^2(i,g)$ hace referencia al cuadrado de la hipotenusa del triangulo y representa la suma de las distancias cuadradas entre cada individuo y el centro de gravedad, $\sum_i d^2_H(i,g)$ hace referencia al cuadrado del cateto ubicado de manera horizontal y representa la suma de las distancias cuadradas entre cada individuo y el centro de gravedad en el subespacio H donde se

proyectan cada uno de los individuos y $\sum_i d^2_{\bar{H}}(i,g)$ es el cuadrado del cateto ubicado de manera vertical y representa las distancias cuadradas entre cada uno de los individuos y el centro de gravedad en el subespacio de la proyección ortogonal $\bar{H}$; se busca que las distancias en el plano $\sum_i d^2_H(i,g)$ sean lo más cercanas a las distancias reales $\sum_i d^2(i,g)$, así se tiene que $Max \sum_i d^2_H(i,g) \Leftrightarrow Min \sum_i d^2_{\bar{H}}(i,g)$.

Figura 4.1: Proyección ortogonal del individuo i sobre u_α



La maximización de las distancias al cuadrado entre cada uno de los puntos y el centro de gravedad es equivalente a encontrar u tal que maximice $u'Z'NZu$ bajo $u'u = 1$. Resolviendo el Lagrangiano de dicha función a maximizar se obtiene que se debe resolver la ecuación 4.7 que equivale a resolver el sistema de valores y vectores propios; por lo que el ACP se reduce a encontrar los respectivos valores y vectores propios (λ y u) de la matriz Z^TNZ o matriz de correlaciones R y la solución a dicho sistema es igual a diagonalizar esta matriz de correlaciones a través de la descomposición espectral, obteniéndose que $UDU^T = R$, donde U es una matriz ortogonal que contiene los vectores propios u_α y D es la matriz diagonal con los valores propios λ_α de forma que la matriz R es semidefinida positiva.

$$Z^TNZu = \lambda u \tag{4.7}$$

De 4.7 se tiene después de premultiplicar por u^T que $u^TZ^TNZu = \lambda$ es la inercia a maximizar y por lo tanto debe ser el valor propio más grande de la matriz de correlaciones R. De Lebart et al. (2000) el valor propio λ_α es un índice de dispersión de la nube de individuos en la dirección definida por el eje α y se denomina inercia recogida por el eje. El análisis de componentes principales busca entonces el mejor subespacio de dimensión $q \leq p$ (reducción de dimensionalidad) que recoja de forma gradual y decreciente las proporciones de inercia proyectadas (González Rojas, 2014).

Teniendo que la proyección sobre el nuevo eje es igual a la matriz Z por el vector u de la transformación, esto equivale a tener que $\psi_\alpha = Zu_\alpha$ posmultiplicando por u'_α y sumando sobre los ejes se tiene que $Z\sum_\alpha u_\alpha u'_\alpha = Z = \sum_{\alpha=1}^{p} \psi_\alpha u'_\alpha$, por lo tanto es posible reconstruir la matriz de datos original a partir de los vectores propios u_α y las componentes principales ψ_α, que como se verá más adelante es el principio fundamental sobre el que funciona el algoritmo NIPALS.

4.8. Algoritmo NIPALS

Propuesto por Wold en 1.975, el algoritmo de mínimos cuadrados parciales iterativos no lineales (NIPALS) es un método secuencial para el cálculo de las componentes principales. Realiza la descomposición singular de una matriz de datos, mediante secuencias iterativas convergentes de proyecciones ortogonales (González Rojas, 2014).

Dada la matriz de datos X_{nxp} con rango a menor o igual al número de columnas las cuales están centradas o estandarizadas, la matriz X puede ser reconstituida mediante la suma de la multiplicación de las componentes principales con los respectivos vectores propios en el eje h de la siguiente manera:

$$X = \sum_{h=1}^{a} t_h P'_h = t_1 P'1 + ... + t_a P'_a$$

Por lo tanto, la columna X_j=$\sum_{h=1}^{a} P_{hj} t_h$ y la fila X_i=$\sum_{h=1}^{a} t_{hi} P_h$ para j=1,2,...,p y i=1,2,...,n

Cuando $h = 1$, en el espacio columna, cada columna de X, llamadas x_1,..., x_p son multiplicadas por t_h , la cual resultaría en un vector $P_{hj} = X'_j t_h$ de tamaño px1, el cual es como el coeficiente de regresión de X_j sobre t_h. Esto se debe a que la columna j se expresa como $X_j = P_{hj} t_j$, lo que se estaría realizando algo como una regresión de mínimos cuadrados ordinarios ($y = \beta x$), exceptuando que la variable x es la columna t_h y la variable y es la columna en particular de X.

En el espacio fila, t_{hi} es el coeficiente de la regresión sin constante del individuo x_i sobre P_h. Cuando $h > 1$, P_{hj} es el coeficiente de regresión de t_h en la regresión simple del vector deflactado $X_j - \sum_{l=1}^{h-1} P_{lj} t_l$ sobre t_h para el espacio columna. En el espacio fila, de manera similar t_{hi} es el coeficiente de de P_h en la regresión de $x_i - \sum_{l=1}^{h-1} t_{li} P_l$ sobre P_h.

El pseudocódigo con que funciona el algoritmo NIPALS se puede describir a partir del flujograma de la figura **4.2**, tal que en la primera etapa para el cálculo de la primera componente se toma una columna arbitraria de la matriz de datos X con la única restricción de no ser una columna de ceros, generalmente se escoge la primera columna de esta o $\bar{x}$, en la segunda etapa se obtiene P_h como el coeficiente de la regresión lineal simple y estandarizada (que no incluye el intercepto) de X sobre t_h; de la regresión por mínimos cuadrados se tiene que $\hat{\beta} = \frac{x'y}{x'x}$, análogamente para nuestro caso $P_h = \frac{X' t_h}{t'_h t_h}$, acto seguido se norma el vector P_h para eliminar sus unidades de medida y para finalizar con esta etapa, se tiene del ACP que $t_h = X P_h$; el proceso es repetido de acuerdo a un error de precisión ε que define el investigador hasta lograr la convergencia en el cálculo de P_h.

En la tercera etapa se obtendrá una nueva X_h deflactada que garantiza la ortogonalidad de

las restantes componentes y vectores propios, lo anterior basándonos en la reconstitución de la base de datos del ACP a partir de las componentes y vectores propios.

Terminadas las tres etapas del algoritmo, el proceso se repetirá completamente partiendo de la matriz deflactada, hasta lograr el cálculo de cada una de las a componentes, donde $a = rango(X)$.El procedimiento matricial que permite el cálculo de los P_h y t_h en la etapa 2 viene dado a partir de las figuras **4.3** y **4.4** respectivamente.

Figura 4.2: Flujograma del algoritmo NIPALS

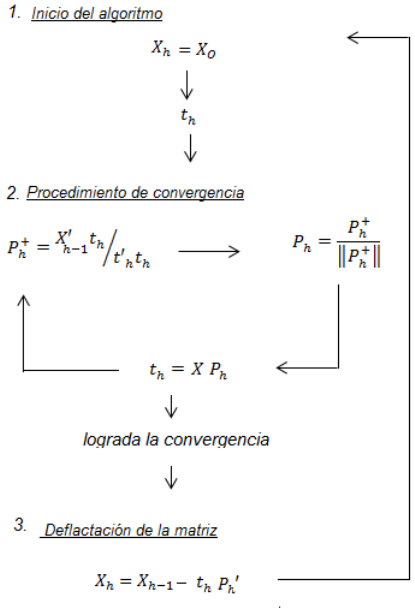


Figura 4.3: Obtención de vectores propios a partir de las componentes

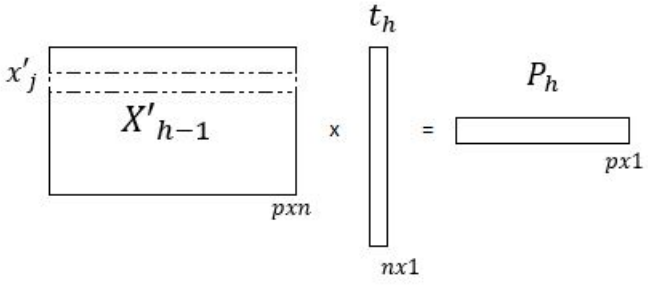
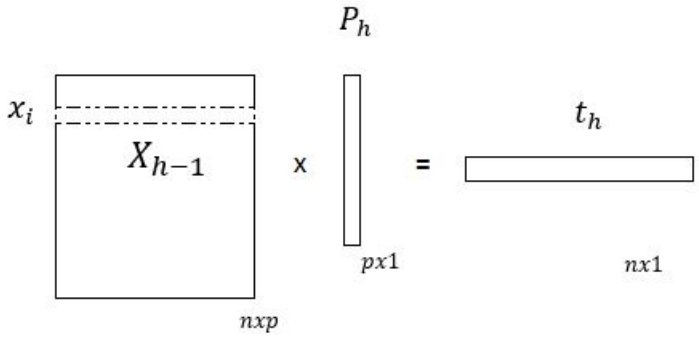


Figura 4.4: Obtención de componentes principales a partir de los vectores



El algoritmo NIPALS además de ser una forma alterna de realizar el análisis de componentes principales, cuenta con una serie de beneficios sobre realizar ACP a través de la matriz de correlaciones $Z^T NZ$ (forma en que trabaja dudi.pca); de acuerdo a (Russolillo et al., 2012) permite la realización del ACP ante la presencia de datos faltantes y aun cuando el número de variables es mayor al número de individuos $p > n$, otra ventaja de este es que al cálcular las componentes de manera secuencial una a una, es posible utilizar el algoritmo solo para las primeras componentes, que por propiedades del ACP son las más importantes al recoger la mayor cantidad de inercia, esto reduce por lo tanto drásticamente el tiempo computacional empleado en el cálculo. Esta ventaja es de gran utilidad en las ciencias ómicas como genómica y metabolómica donde generalmente solo es necesario calcular las primeras componentes; otra de las ventajas para NIPALS vistas por (Andrecut, 2009) es la capacidad de paralelizar este algoritmo.

4.9. Ade4

El paquete ade4 (Data Analysis functions to analyse Ecological and Environmental data in the framework of Euclidean Exploratory methods) fue desarrollado para el software estadístico R en el año 2002 por el laboratorio de biometría y biología evolutiva de la universidad de Lyon. Se caracteriza por la implementación de funciones gráficas y estadísticas, la disponibilidad de datos numéricos, la redacción de documentación técnica y temática y la inclusión de referencias bibliográficas; es una versión totalmente reescrita de su antigua versión ADE-4 desarrollada en 1997 (Chessel et al., 2004).

Ade4 propone múltiples métodos multivariantes cuya implementación sigue la tradición de la escuela francesa de “análisis de datos"que se basa en el uso de una herramienta matemática unificadora llamada diagrama de dualidad (Dray et al., 2007), que hace uso de la tripleta de matrices X, Q y D, donde en términos del ACP, X es una matriz de tamaño nxp que contiene variables cuantitativas centradas o estandarizadas, Q es una matriz diagonal de

dimensión pxp que contiene los pesos de las p variables de X y D es una matriz también diagonal de tamaño nxn llamada métrica en el ACP y contiene el peso de los distintos individuos de X. Para cada uno de los diferentes métodos básicos multivariados las matrices X, Q y D toman diferentes características en particular, incluso los métodos más complejos pueden ser también representados a partir de diagramas de dualidad (Chessel et al., 2004).

Tabla 4.1: Tabla para funciones dudi de la librería ade4

Función	**Análisis**
dudi.pca	Componentes principales
dudi.coa	Correspondencias
dudi.acm	Correspondencia múltiple
dudi.fca	Correspondencias difuso
dudi.mix	Mezcla numérica y factorial
dudi.nsc	Correspondencias no simétrico
dudi.mix	Correspondencias descentralizado

Cada uno de los diferentes métodos multivariados que son casos particulares del diagrama de dualidad pueden ser llamados con la función "dudi.*"; la tabla **4.1** muestra las funciones para cada uno de los diferentes análisis. Para el análisis de componentes principales la función correspondiente sería dudi.pca y la descomposición en valores singulares de la tripleta se encarga de devolver una lista de la clase dudi que contiene las componentes principales, valores propios, vectores propios, rango, los pesos de las columnas y de las filas, matriz de distancias, entre otros; no obstante ésta lista se encuentra altamente limitada en grandes volúmenes de datos, puesto que está condicionada por la memoria virtual de R que dependiendo de su configuración y el sistema operativo que utilice puede variar de 2 a 4 Gb; para mayores dimensiones el sistema es incapaz de procesar esta información y arroja un mensaje de error. Cabe resaltar que debido a que este proyecto centra su importancia en encontrar las componentes principales, valores y vectores propios, las funciones de la librería Ade4 en R que entregan dichos valores son li, eig y $c1$ respectivamente a partir del objeto dudi.pca que haya sido creado.

4.10. Cópulas

En teoría de probabilidad las cópulas son funciones de distribución multivariantes con funciones de distribución uniformes $[0 - 1]$ en las marginales; en la actualidad son ampliamente utilizadas para establecer relaciones de dependencia entre variables aleatorias. Es conocido que una variable aleatoria viene determinada por su función de densidad y función de distribución, para el caso de dos o más variables (bivariante y multivariante), el comportamiento de estas vendrá determinado por la función de distribución conjunta (Ayyad

et al., 2008), y a partir de esta es posible determinar las funciones de distribución marginales, sin embargo no siempre es posible determinar cuál será la función de distribución conjunta a partir de sus marginales, el caso más trivial es cuando las variables son independientes, en cuyo caso la función de distribución conjunta será el producto de sus marginales, no obstante cuando las variables guardan cierto nivel de dependencia, lo anterior no se cumple; Sklar (1959) trabajó en dicha problematica mediante un teorema donde una función, posteriormente llamada "cópula" establece el nivel de dependencia entre variables, y mediante el uso de esta es posible obtener la función de distribución conjunta de un conjunto de variables aleatorias a partir de sus marginales, así dicha función se denota por:

$$F_{xy}(x, y) = C_{xy}(F_x(x), F_y(y))$$

$$C_{xy} = F_{xy}(x, y)(F_x^{-1}(x), F_y^{-1}(y)) \tag{4.8}$$

Las cópulas emplean la transformación de funciones marginales de tipo uniforme con una determinada estructura de correlación a variables aleatorias tomando su función de distribución acumulada. La estructura de dependencia vendrá entonces determinada por las relaciones establecidas entre variables uniformes (Chavarro, 2012). La dependencia entre variables puede venir de forma lineal o no lineal, la forma más conocida es el coeficiente de correlación de Pearson que mide relaciones estrictamente lineales, entre otras "medidas de asociación" están el Tau de Kendall y el coeficiente de correlación de Spearman que no miden relaciones necesariamente lineales; ambas medidas se pueden obtener a través de una cópula, de esta manera:

Tau de Kendall

Para el caso bivariante. Sean X y Y variables aleatorias continuas con cópula C, entonces:

$$\tau_{xy} = 4 \int \int C(u, v) dC(u, v) - 1 = 4E[C(U, V) - 1]$$

Coeficiente de correlación de Spearman

Al igual que en el Tau de Kendall, la relación entre una función cópula y el coeficiente de Spearman se puede expresar para el caso bivariado mediante:

$$\tau_{xy} = 12 \int_0^1 \int_0^1 C(u,v) du dv - 3$$

Existen diversos tipos de funciones cópulas que suelen ser empleadas de acuerdo al tipo de relación que se busca, entre estas están las elípticas (simétricas), de las que destacan la normal multivariada y la basada en la distribución t student, están también las funciones cópulas para valores extremos (dependencia en las distribuciones marginales de las colas) y las llamadas arquimedianas que representan otro tipo de dependencia, entre las que cabe mencionar la cópula gumbel, Frank, Clayton, entre otras. En particular, la cópula normal o gaussiana emplea la distribución normal multivariante con una matriz Σ de correlaciones y es principalmente útil cuando el grado de dependencia que se busca es de tipo simétrico (bajos niveles de dependencia en las colas). Si el objetivo yace en simular variables con niveles de dependencia cópula se debe entonces escoger cuál será el nivel de dependencia deseado y de qué forma estará dada dicha dependencia (por medio de qué función), posterior a ello se simulan variables aleatorias con distribución uniforme que tengan el nivel de dependencia deseado y para finalizar a través de estas variables uniformes se obtienen las funciones de distribución marginales.

Capítulo 5

Metodología

En este capítulo se desarrolla la propuesta metodológica para el presente proyecto, manteniendo siempre una constante revisión bibliográfica en la que se plantean el conjunto de procedimientos necesarios a fin de llevar a cabo el correcto cumplimiento de los objetivos ya especificados. Se plantearán las condiciones bajo las que se realizará la simulación del caso de estudio, tales como: características de la máquina empleada, tipo de correlaciones a utilizar, número de variables e individuos, función de distribución que seguirán las variables, entre otras; se especificará también cuál será el algoritmo a utilizar, las razones por las cuales se eligió y los pasos a seguir para la construcción de éste a fin de que cumpla con ser un algoritmo que presente la posibilidad de implementarse a futuros estudios en un ambiente de computación distribuido o paralelo. Para finalizar se especificará la forma en que el algoritmo planteado se utilizará en un caso de estudio final con 1.000.000 de individuos y 20 variables, con el que para concluir se verifiquen todas las propiedades que debe cumplir un análisis de componentes principales.

5.1. Características computacionales

Debido a que las características internas del computador (velocidad de procesador, memoria de trabajo, sistema operativo, entre otros) influyen directamente en la cantidad de información con que se puede trabajar, se detallarán las características del computador con que se obtendrán los resultados, teniendo en cuenta que este cumple con las capacidades estándar de una computadora promedio actual.

5.1.1. Hardware

Se eligió una computadora con características de Hardware similares a la mostrada en la tabla **5.1**

Tabla 5.1: Especificaciones del Hardware

Intel CPU	i3-5005U CPU
Nº de Cores	4
RAM	6.00 GB
Rapidez de reloj	2 GHZ
Tamaño del disco	931.51 GB
L2 Caché	3 MB
RPM	5400
Bytes/Sector	512
Tipo de sistema	PC basado en x64

5.1.2. Software

Así como las características de hardware del computador influyen sobre la capacidad de procesamiento, es también importante especificar el software o conjunto de programas instalados mediante los que se obtendrán los determinados resultados.

Para la realización y ejecución del algoritmo y los cálculos en su totalidad se utilizará el programa estadístico R project versión 3.4.1. mediante el entorno de desarrollo Rstudio version 1.1.383, el cual es un entorno de software libre para análisis y gráficos estadísticos, que es capaz de ejecutarse en diferentes plataformas (UNIX, Windows y MacOS), para nuestro caso se ejecutará en una plataforma bajo el sistema operativo Windows 10 de 64 bits donde el límite en memoria de trabajo para R sería de 3GB (R Core Team, 2017). Se trabajará además con las librerías cópula (para la generación de cópulas de dependencia) (Hofert et al., 2017) y ade4 (para ACP) (Dray & Dufour, 2007).

5.2. Constitución de la base de datos

Con el fin de cumplir con los objetivos propuestos en este trabajo de grado se requiere del uso de un conjunto de datos de alta dimensionalidad que cumpla con las condiciones necesarias para realizar un análisis de componentes principales; entiéndase por alta dimensionalidad una matriz con $n >> p$, es decir, número de individuos será mucho mayor al número de variables.

Autores como Aluja et al. (1999), sugieren que la aplicabilidad del ACP está sujeta a condiciones ideales, las cuales son la normalidad de los datos y la linealidad de las relaciones entre variables. Por otro lado, Jolliffe (2005) enfatiza que los supuestos de normalidad multivariada son importantes en las propiedades geométricas del ACP. Se tiene también de acuerdo a Lebart et al. (2000) que si los datos son poco estructurados (las variables no están fuertemente correlacionadas entre ellas), la nube tiene una forma

regular. En este caso los valores propios decrecen regularmente y el análisis factorial no dará resultados interesantes.

Se simularán conjuntos de datos con diferente número de variables e individuos, de forma que, el número de individuos estará entre 100 y 1.000.000, mientras que el número de variables será de a lo sumo 35. Todas las variables serán de tipo cuantitativo y gracias a que la relación entre variables simétricas puede ser solamente de tipo lineal y ese es el tipo de relación que más interesa para el ACP, se utilizará una distribución simétrica, para este caso específico serán variables normales con parámetros (media y varianza) también variables; la media tomará valores aleatorios entre 0 y 50 y las varianzas estarán entre 0.01 y 25. Cabe resaltar que según teoría del ACP la matriz a diagonalizar es la matriz de correlaciones $R = Z^T N Z$ que cumple con ser semidefinida positiva, por lo que se debe encontrar el conjunto de parámetros que hagan cumplir a la matriz R con dicha condición.

Cuando las variables no están correlacionadas entre sí, todas estas recogerán aproximadamente la misma proporción de inercia y al saber que la suma de los valores propios debe ser igual al rango de la matriz, los valores propios en el caso de no correlación estarán cercanos a uno en todos los ejes, por lo anterior, se utilizará una función cópula elíptica de tipo normal con dependencia simétrica que simule distintos niveles de dependencia entre cada una de las variables, así para la matriz que se utilizará de prueba con 100.000 individuos y 20 variables, se crearán dos grupos de 8 variables, uno con dependencia fuerte (entre 0.7 y 1), otro con dependencias moderadas (entre 0.5 y 0.69) y un tercer grupo de 4 variables con dependencias moderadamente débiles (entre 0.3 y 0.49). Las correlaciones entre variables de diferentes grupos serán bajas. La simulación para las dependencias tipo cópula se obtendrán a partir de la librería cópula de R project mediante la función normalCopula (Hofert et al., 2017).

Una vez se constituya la matriz de prueba con las respectivas dependencias cópula, se debe verificar que en efecto la constitución de la base de datos es idónea para la realización de un análisis de componentes principales, por lo tanto se evaluarán factores como tiempo de ejecución del algoritmo, iteraciones necesarias para lograr la convergencia en cada uno de los componentes, ortogonalidad entre componentes y vectores propios, inercia recogida por cada uno de los ejes a través de los valores propios, matriz de correlaciones para comprobar la existencia de relaciones importantes entre variables y demás. Al ya contar con una matriz que cumpla con las condiciones suficientes para la realización de un ACP, se obtendrá la matriz final para el caso de estudio con 1.000.000 de individuos y 20 variables que contendrá las mismas características y parámetros con que fue creada la matriz de prueba.

5.3. Alcance en librería dudi.pca

Programas estadísticos como R que contienen funciones como dudi.pca (Dray & Dufour, 2007), realizan un análisis de componentes principales mediante diagramas de dualidad a través de la matriz $Z^T NZ$. No obstante ante grandes volúmenes de datos se mostrará que dicha librería presenta fallos en la presentación de resultados, no siendo capaz el sistema de almacenar cantidades tan grandes de información y por lo tanto, no genera resultados al realizar un ACP.

Por lo anterior, se probará la función dudi.pca de la librería ade4 con matrices de datos que varíen en tamaño a fin de determinar el límite computacional que tiene dicha función tanto en cantidad de información para procesar (máximo número de individuos y variables con que se arrojan resultados) como en tiempo que toma para entregar dichos resultados. Con estos resultados se realizará un análisis para identificar cómo afecta el aumento de las variables y los individuos y cuál de estos dos afecta más al tiempo de ejecución y fallo de esta función. Se presentará así una versión del algoritmo NIPALS que permita la solución de esta problemática.

5.4. Construcción del algoritmo

Para proponer un algoritmo que implemente ACP en grandes cantidades de datos, el algoritmo debe ser capaz de obtener tres elementos muy importantes: los valores propios, los vectores propios y las componentes principales. El elemento principal son los vectores propios, puesto que con estos se obtienen las componentes principales y con ello los valores propios. Recordemos que este conjunto de elementos son esenciales para el ACP.

Existen varios algoritmos para ACP en la literatura, entre estos los que utilizan directamente la matriz de datos como el SVD (descomposición de valores singulares), NIPALS (mínimos cuadrados parciales no lineales iterativos), y algunos otros que usan la covarianza de la matriz de datos como el EVD (descomposición de valores propios) (Jolliffe, 1986).

La implementación de ACP a través de algoritmos como SVD y EVD que extraen las componentes principales de manera simultánea se vuelven rápidamente difíciles de implementar a medida que el conjunto de datos aumenta, a diferencia del algoritmo NIPALS que realizan este cálculo de manera secuencial (Andrecut, 2009).

Algoritmos iterativos como el algoritmo de Lanczos no son muy estables debido a que la pérdida de ortogonalidad afecta directamente a los vectores propios, en algunos casos ocasionando que el nuevo vector generado sea linealmente dependiente del conjunto que ya está construido. Por lo tanto, algunos valores propios resultantes no son buenas

aproximaciones de los valores propios de la matriz original. Si se implementa este algoritmo habría que enfrentarse a la pérdida de ortogonalidad, tratar de recuperar la ortogonalidad después de que se genera la base y además identificar los valores propios "con errores" para posteriormente eliminarlos (Saad, 1992).

Otros algoritmos iterativos como Jacobi pueden ser muy costosos computacionalmente debido a la complejidad de este mismo, por ende la implementación del algoritmo Jacobi se realiza en su mayoría de forma paralela (Sameh, 1971). Por lo tanto, el algoritmo NIPALS podría ser una solución aproximada más eficiente para el cálculo de las componentes principales.

Transformando NIPALS para grandes volúmenes de datos

Al implementar el algoritmo NIPALS para una base de datos de gran dimensión se debe tener en cuenta: La correcta partición en los procesos del algoritmo de forma que se mantenga la idea de realizar un único ACP, el cálculo del rango de la matriz y el cálculo del error de precisión con que trabajará el algoritmo a fin de conservar la ortogonalidad tanto en vectores propios como en componentes principales.
Teniendo en cuenta lo anterior:

5.4.1. Correcta partición del algoritmo

NIPALS presenta dos operaciones dentro de su algoritmo ubicadas en la parte iterativa que utilizan la matriz $X_{h_{nxp}}$: Primero, en la multiplicación con el vector $t_{h_{nx1}}$, el cual corresponde al componente principal y, cuando multiplica por el vector $P_{h_{px1}}$ que es el vector propio.

En la mayoría de los casos el número de individuos es mucho mayor que el número de variables, por lo tanto la operación con mayor grado de dificultad se presenta en la multiplicación de la matriz $X'_{h_{nxp}}$ con el vector $t_{h_{nx1}}$ (ver gráfico **5.1**) .

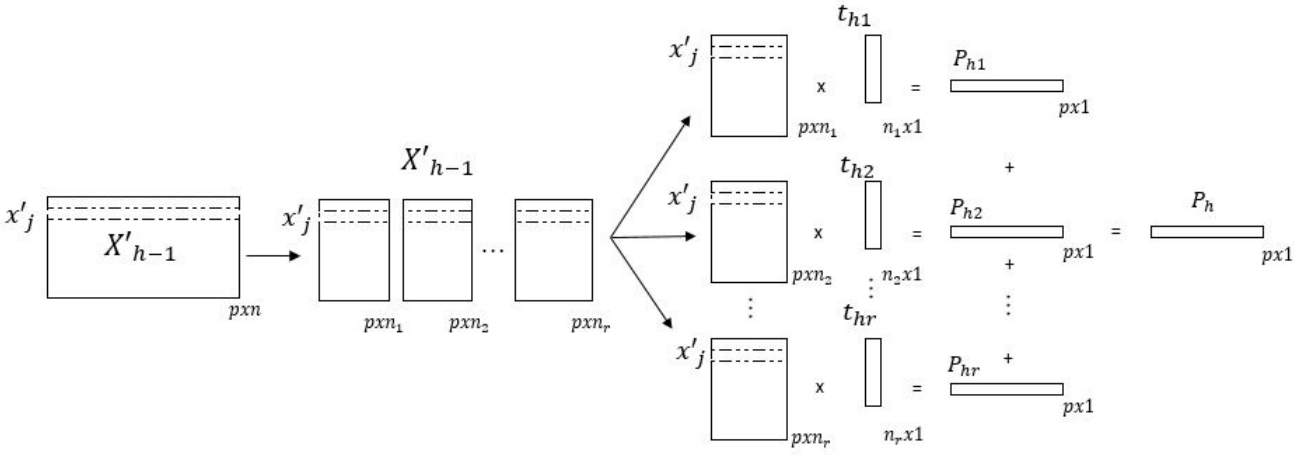
Debido a lo anterior, particionar la matriz permite que las operaciones a realizar se ejecuten en matrices más pequeñas evitando así la problemática de los límites de memoria de R en alojar vectores de gran tamaño (R Core Team, 2017). En cierta forma, la idea de particionar la matriz se basa en el concepto "paralelo" de operar las particiones para luego encontrar un resultado único.

La cantidad de particiones de la matriz, el cual se define con la letra r, dependerá exclusivamente del usuario debido a que esta cantidad está ligada a la capacidad del computador. Por ejemplo, si un computador no es capaz de realizar una multiplicación de una matriz de tamaño 50x100.000 por un vector 100.000x1, deberá particionar la matriz en

más de dos bloques si se tiene una base de datos con 200.000 individuos y 50 variables. Por lo tanto, encontrar una partición óptima se torna compleja puesto que la partición adecuada para unos usuarios no lo será para otros. Lo que implicaría más iteraciones para completar la multiplicación de las matrices en los computadores de menor capacidad.

A continuación, se muestra en el gráfico **5.1** un esquema de cómo se particiona la matriz para realizar la multiplicación entre la traspuesta de la matriz y th:

Figura 5.1: Partición de la matriz de datos



Teniendo en cuenta que la multiplicación vectorial de dos vectores cualquiera es:

$$y = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} . \begin{bmatrix} a_1 a_2 \dots a_n \end{bmatrix} = \sum_{i=1}^{n} b_i . a_i \tag{5.1}$$

Y se puede expresar como en la ecuación 5.2, gracias a las propiedades de las sumatorias (Spivak, 1996):

$$y = \begin{bmatrix} b_1 \\ \vdots \\ b_k \\ - \\ b_{k+1} \\ \vdots \\ b_n \end{bmatrix} . \begin{bmatrix} a_1 \dots a_k | a_{k+1} \dots a_n \end{bmatrix} = \sum_{i=1}^{k} b_i . a_i + \sum_{i=k+1}^{n} b_i . a_i \tag{5.2}$$

Haciendo uso de este concepto, $P_{h1} + P_{h2} + ... + P_{hr}$ da como resultado al vector P_h mostrado en la figura **5.1**.

5.4.2. El cálculo del rango de la matriz

El número de componentes principales que se pueden calcular en una matriz de datos es igual al rango de la matriz, es decir, al número de columnas o filas independientes. Normalmente para el cálculo del rango se utiliza la descomposición QR, pero cuando la matriz es muy grande el cálculo se torna complejo respecto con el tiempo de entrega de los resultados y ralentización del computador (Sayadi et al., 2014).

Fácilmente se puede igualar el rango de la matriz con el número de columnas (si p $\leq$ n) y así, dejar que el algoritmo Nipals itere hasta alcanzar la cantidad fijada. Este proceso no sería conveniente debido a que sería menos eficiente con respecto al tiempo de demora del algoritmo, puesto que iterará veces innecesarias cuyos valores propios para las columnas dependientes serán aproximadamente cero, debido a que estas variables (columnas dependientes) no aportan nueva información acerca de la base de datos, por lo tanto la inercia que recoge sus respectivas componentes es igual a cero.

Para solucionar este inconveniente se calculará el rango de la matriz X^TX, la cual tiene el mismo rango de X. Esta operación reduciría el tamaño de la matriz puesto que al multiplicar la traspuesta de X_{pxn} por ella misma queda una matriz de tamaño pxp a la cual es fácil de calcular el rango.

5.4.3. Cálculo del error de precisión ε

Debido a que NIPALS es un algoritmo iterativo para hallar las componentes principales, valores y vectores propios, la acumulación de errores que dan por cada aproximación, generan un error total final que de acuerdo a (Kramer, 1998) implica la pérdida de ortogonalidad en las componentes principales en grandes matrices, por lo tanto se realizará un análisis de la ortogonalidad tanto de los vectores propios como de las componentes principales del algoritmo NIPALS clásico a medida que aumenta el tamaño de la matriz de datos (tamaño mediante el cual el algoritmo puede arrojar resultados), inicialmente el análisis se realizará manteniendo fijo el número de variables y variando el número de individuos, posteriormente el análisis se realizará fijando el número de individuos y haciendo variar el número de variables.

Se mostrará que dicha "pérdida de ortogonalidad" es producto únicamente del error de precisión que es siempre definido por el investigador y que en efecto a mayor dimensión de la matriz, más pequeño tendrá que ser el error y por lo tanto mayor número de iteraciones y más esfuerzo computacional será necesario si se quiere conservar un nivel de precisión en específico. De acuerdo al análisis de ortogonalidad que se observe en los resultados se fijará el nivel de precisión y en base a ello se determinará el error de cálculos ε en función del número de individuos y variables de la matriz de entrada, dicho error de precisión es calculado a partir de la norma de las diferencias existentes entre el cálculo del componente en una

iteración y en la siguiente, lo que es igual a $\sqrt{\sum(t_{i,j} - t_{i,(j+1)})^2}$ donde $t_{i,j}$ es el componente principal del eje i hallado en la j-ésima iteración y $t_{i,(j+1)}$ es el mismo componente en la j+1-ésima iteración, así el error ε mediría en cuántos decimales debe coincidir el cálculo del componente entre una y otra iteración para lograr la convergencia en el cálculo de éstos y decir que se garantiza la ortogonalidad en componentes principales.

Con el objetivo de comparar el algoritmo NIPALS en términos de velocidad de procesamiento y eficiencia en los resultados obtenidos se empleará el algoritmo clásico de Gram - Schmidt (CGS) (NIPALS-GS), el cual puede ser adaptado para computarse en un solo procesador, dicho algoritmo calcula las componentes principales y vectores propios en cada paso de la iteración y los re-ortogonaliza construyendo recursivamente un conjunto de vectores de base ortonormal para el subespacio abarcado por un conjunto dado de vectores normalizados linealmente independientes (Andrecut, 2009), de lo anterior Gram-Schmidt garantiza la ortogonalidad tanto en las componentes principales como en los vectores propios, sin embargo se mostrará que este proceso es computacionalmente mucho más demandante que fijar un error de precisión ε muy pequeño en el algoritmo NIPALS e iterar hasta lograr el nivel de ortogonalidad deseado, dado que la re-ortogonalización de Gram-Schmidt es un paso adicional llevado a cabo en el algoritmo.

5.5. Análisis del algoritmo vía NIPALS

Teniendo en cuenta cada una de las intervenciones realizadas al algoritmo NIPALS, el **Algoritmo 1** muestra el pseudocódigo formulado.

Se implementará el algoritmo vía NIPALS a las matrices simuladas de diferentes tamaños variando el número de individuos y variables. Para cada implementación se tomará el tiempo que demore el algoritmo en arrojar los resultados y además, la cantidad de iteraciones que realiza para cada una de ellas. Con esto, se pretende analizar cómo afecta el aumento de individuos y variables en el tiempo de ejecución de NIPALS y en la cantidad de iteraciones.

Debido a la complejidad de calcular el tiempo que demora cada iteracion en el algoritmo, se asume que el tiempo por cada una de estas es uniforme. Por consiguiente, se analizará el tiempo por iteración, la cual se calcula como el cociente de ambos para las diferentes matrices simuladas esperando encontrar algún comportamiento en particular. Posteriormente, con la matriz que se había fijado en un comienzo con 100.000 individuos y 20 variables, se calcularán los valores propios, vectores propios y componentes principales mediante el algoritmo vía NIPALS. Se analizará la ortogonalidad de los vectores y componentes principales.

Se compararán estos resultados con los resultados procedentes de la función dudi.pca, se

observará el promedio de las diferencias de los vectores propios y componentes principales de ambos métodos.

Entradas: x=Matriz estandarizada
n=Número de filas de x
p=Número de columnas de x
a=Rango de $x^t x$
r=Número de particiones a utilizar
ε=Error de precisión

$x_0 = x$
para *h=1,2,...,a* **hacer**
  $P_0 = 0$
  t_h=Columna h de x_{h-1}
  nm=1
  mientras $nm \geq \varepsilon$ **hacer**
    p=0
    si *n es par* **entonces**
      para *i =1,2,..,r* **hacer**
        t_{11}=t_h desde $\frac{n}{r}(i-1)+1$ hasta $\frac{n}{r}i$
        m=x_{h-1} desde $\frac{n}{r}(i-1)+1$ hasta $\frac{n}{r}i$ para las filas y para las columnas desde 1 hasta p.
        p=p+$m't_{11}$
      fin
    en otro caso
      para *i =1,2,..,r-1* **hacer**
        t_{11}=t_h desde $\frac{n}{r}(i-1)+1$ hasta $\frac{n}{r}i$
        m=x_{h-1} desde $\frac{n}{r}(i-1)+1$ hasta $\frac{n}{r}i$ para las filas y para las columnas desde 1 hasta p.
      fin
      t_{11}=t_h desde $\frac{n}{r}(r-1)+1$ hasta n
      m=x_{h-1} desde $\frac{n}{r}(r-1)+1$ hasta n para las filas y para las columnas desde 1 hasta p.
      p=p+$m't_{11}$
    fin
    P_h=$\dfrac{p}{t'_h t_h}$
    P_h=$\dfrac{P_h}{\|P_h\|}$
    t_h=$x_{h-1}P_h$
    nm=$\|P_h - P_{h-1}\|$
  fin
  $x_h = x_{h-1} - t_h P'_h$
fin
Salidas : P, t

Algoritmo 1: Pseudocódigo NIPALS

5.6. Aplicación del algoritmo final con una matriz de gran dimensión

Para finalizar, se simulará una matriz con 1.000.000 individuos y 20 variables para aplicar el algoritmo vía NIPALS.

Se analizarán los resultados de NIPALS y se verificarán que las propiedades propias del ACP se cumplan. Como primera instancia, se observarán los valores propios de la matriz esperando que la suma de todos ellos sea igual a p (número de variables) y que además sean decrecientes. Luego, se analizará la ortogonalidad de los vectores propios y componentes principales.

Capítulo 6

RESULTADOS

Este capítulo comprende desde el análisis de una matriz simulada con dependencia cópula, el estudio de los tiempos de ejecución de la función dudi.pca, creación del algoritmo vía NIPALS, hasta la ejecución de este algoritmo en una gran base de datos simulada.

Para la creación del algoritmo vía NIPALS se realizaron continuamente pruebas para enfrentar las problemáticas anteriormente mencionadas que resultan al trabajar con grandes cantidades de información: Se expone la importancia del cálculo del rango de la matriz y el cálculo del error de precisión que mantenga la ortogonalidad tanto de componentes principales como de vectores propios. También se mostrará una comparación del algoritmo vía NIPALS y el algoritmo NIPALS-GS creado por (Andrecut, 2009).

6.1. Constitución de la base de datos

Esta sección aborda la descripción de la creación de la base de datos simulada idónea para aplicar un análisis de componentes principales para grandes volúmenes de datos.

En un principio se simuló una base con 100.000 individuos y 20 variables que solo contenía variables aleatorias de distribución normal con distintas medias y varianzas. Sin embargo, al analizar los valores propios de ésta arrojados por la función dudi.pca que se muestran en tabla **6.1**, se observó que todos ellos eran muy cercanos a uno, por lo que ante un caso así como ya se mencionó, todas las variables recogen la misma proporción de inercia. Adicional a lo anterior, al analizar las matrices **6.2** y **6.3** que tienen las correlaciones entre variables para la matriz simulada, se encontraron únicamente correlaciones débiles, siendo en todos los casos menores al 1 %. Dado que la matriz de correlaciones es simétrica solo se mostraron los valores del lado superior de la diagonal principal para facilitar su análisis.

Tabla 6.1: Valores propios sin dependencia cópula

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1.2537	1.2053	1.1883	1.1560	1.1247	1.0968	1.0677	1.0565	1.0357	1.016

11	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
0.9857	0.9604	0.9528	0.9142	0.8959	0.8677	0.8398	0.8236	0.8025	0.7550

Tabla 6.2: Matriz de correlaciones sin dependencia cópula

Variables	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	1	0.0003153	0.0002680	0.0013492	-0.0051250	-0.00001640	0.0024508	0.001797	-0.001537	-0.002703
2		1	-0.00278375	0.0005385	-0.0039052	0.0069107	0.002409	-0.0042115	0.007932	0.001790
3			1	-0.001935	-0.001769	0.000245	0.00087413	0.001385	-0.0072102	0.000776
4				1	0.0018546	-0.0025485	-0.00419374	-0.0005307	0.0042488	-0.0049312
5					1	0.003331	0.0004194	-0.0046182	-0.000800	-0.0012202
6						1	-0.001511	-0.0028523	0.003411	0.003044
7							1	-0.002067	0.0029700	-0.00460
8								1	0.00104	0.002786
9									1	-0.00086
10										1
11										
12										
13										
14										
15										
16										
17										
18										
19										
20										

Tabla 6.3: Matriz de correlaciones sin dependencia cópula

Variables	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
1	-0.00784	-0.00248	-0.004322	-0.00140	0.00134	0.001906	-0.0015557	-0.007124	0.0004885	0.0000682
2	-0.000011	-0.001493	0.000228	-0.001026	-0.004968	-0.0012218	-0.000864	0.000535	-0.0036715	-0.00444
3	0.004086	-0.001760	-0.00139	0.002684	-0.001906	0.0072178	0.004866	0.000704	0.0003434	-0.0018958
4	0.005314	0.000300	0.004048	-0.0029996	-0.0010240	-0.0068767	-0.004658	-0.00397498	0.002058	0.0016181
5	-0.00165	0.0007951	0.0057028	0.0022824	-0.00041156	0.0018506	-0.0024678	0.0028630	-0.003660	0.002583
6	-0.0042672	-0.0026603	0.0002420	0.001712	0.00267646	0.00136053	-0.001387265	0.00037562	0.0025305	0.0020003
7	0.0002292	-0.006921	0.0045330	0.0014792	-0.0009026	0.0003189	-0.0045005	-0.00054887	-0.000998	-0.003166
8	-0.004131	0.000690	-0.0054140	-0.002337	-0.0018529	0.000437	-0.0033172	0.00355	0.000013	0.002862
9	-0.003188	-0.004621	0.000366	-0.006365	-0.004058	-0.007950	0.003919	-0.0006003	-0.0016800	-0.00351
10	-0.0032676	-0.0001491	0.0022821	-0.002877	0.003301	0.007379	-0.0028084	-0.00357	0.0043649	0.0008043
11	1	-0.001259	0.00514908	0.0058579	0.000754	0.0004487	-0.0004271	-0.000564	-0.00563	-0.002875
12		1	0.0033293	-0.0001207	-0.0041086	-0.000497	-0.0004271	-0.0005642	-0.005630	-0.002875
13			1	-0.000618	0.001279	0.002087	0.001164	-0.001237	0.00829	-0.002338
14				1	-0.005335	0.000825	-0.007912	0.001191	0.005017	-0.0000017
15					1	-0.004687	0.002407	-0.0017628	-0.004702	-0.0004420
16						1	0.000436	-0.001018	0.001681	0.00493
17							1	-0.003175	0.004778	0.000053
18								1	0.0007575	-0.0010369
19									1	-0.000341
20										1

Ante estos resultados, se creó la base de datos mediante una función cópula elíptica, más específicamente una función cópula normal con dependencias fuertes, moderadas y

medianamente débiles. Las medias y las varianzas fueron elegidas aleatoriamente.

Se crearon dos grupos de 8 variables con dependencia fuerte entre los valores de 0.7 y 1, dependencias moderadas entre 0.5 y 0.69 y un grupo de 4 variables con dependencias moderadamente débiles entre los valores 0.3 y 0.49, esto con el fin de encontrar correlaciones muy altas, altas y moderadas. Además, se encontrarán correlaciones bajas entre variables de diferentes grupos, puesto que los grupos creados son independientes entre sí, de esta manera, la base simulada tendría diferentes niveles de correlación.

Las medias y las varianzas para las variables de la base de datos se simularon de manera aleatoria entre 0 y 50 para las medias y, 0.01 y 25 para las varianzas.

En la tabla **6.4** se muestra nuevamente el cálculo de los valores propios por medio de la función dudi.pca, esta vez a la base que se simulo con los diferentes niveles de dependencia cópula, cuyo comportamiento es claramente diferente al de los mostrados en la tabla **6.1**. Teniendo que la matriz es de rango completo, la suma de los valores propios debe ser igual al número de variables de la matriz, en este caso igual a 20; partiendo de esto es posible conocer que la suma de los tres valores propios es $6.5841 + 4.8649 + 2.0391 = 13.4881$, lo que es equivalente a que los tres primeros valores propios recogen un $67.44\,\%$ de la inercia total ($13.4881/20 = 0.6744$).

Tabla 6.4: Valores propios con dependencia cópula

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
6.5841	4.8649	2.0391	0.7097	0.6448	0.6064	0.5464	0.52169	0.4739	0.4496
11	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
0.4346	0.4138	0.3904	0.3183	0.3065	0.24837	0.2419	0.1238	0.0754	0.0053

En las tablas **6.5** y **6.6** se observan variables con correlaciones muy altas, altas, moderadas y bajas, tal como se crearon los niveles de dependencia de acuerdo a las funciones cópula. La matriz simulada es adecuada para realizar un análisis de componentes principales, ya que existirá información redundante; pocos factores explicarán gran parte de la variabilidad total.

Tabla 6.5: Matriz de correlaciones

Variables	1	2	3	4	5	6	7	8	9	10
1	1	**0.82208**	**0.76372**	**0.7085**	**0.7568**	**0.8020**	**0.8849**	**0.7956**	-0.001286	0.00089
2		1	**0.70621**	**0.80826**	**0.72087**	**0.7655**	**0.7786**	**0.85791**	0.0003134	0.00277
3			1	**0.7898**	**0.7565**	**0.8903**	**0.8061**	**0.8093**	-0.00004895	0.00030
4				1	**0.8332**	**0.8248**	**0.8184**	**0.8414**	-0.000636	-0.00012
5					1	**0.8382**	**0.7604**	**0.7809**	0.0005204	-0.00261
6						1	**0.7125**	**0.8140**	0.001038	0.00156
7							1	**0.88182**	-0.001553	-0.0015
8								1	0.00104	0.0027
9									1	**0.5058**
10										1
11										
12										
13										
14										
15										
16										
17										
18										
19										
20										

Tabla 6.6: Matriz de correlaciones

Variables	11	12	13	14	15	16	17	18	19	20
1	0.00208	-0.0033	-0.0041	0.00068	0.0012	0.00053	-0.0035	-0.0010	0.00502	-0.0012
2	0.0035	-0.0041	-0.0013	0.00466	0.0008	0.0031	-0.0033	-0.0021	0.00336	-0.00018
3	0.00454	-0.00220	-0.0039	-0.000009	0.00137	0.0028	0.00026	-0.0014	0.003707	-0.00208
4	0.0042	-0.00550	-0.0027	0.00134	0.00128	0.00056	-0.0020	-0.0015	0.002295	-0.00123
5	0.0026	-0.0047	-0.0022	0.00201	0.00206	0.0017	-0.0031	0.00029	0.00489	0.0014
6	0.0044	-0.0019	-0.0023	0.00103	0.00316	0.00422	-0.0017	-0.00024	0.00368	-0.00135
7	0.0033	-0.0045	-0.0041	0.00111	0.00105	-0.0003	-0.0028	-0.0024	0.00357	-0.00208
8	0.00543	-0.0027	-0.00088	0.00376	0.00269	0.0045	-0.00307	-0.0014	0.0033	-0.0021
9	**0.5062**	**0.5726**	**0.5469**	**0.563**	**0.5640**	**0.4994**	-0.0051	-0.00059	0.000096	-0.00381
10	**0.5552**	**0.57493**	**0.5664**	**0.5242**	**0.6000**	**0.5037**	-0.0041	-0.00033	0.0025	-0.00093
11	1	**0.5439**	**0.5673**	**0.5133**	**0.5618**	**0.5516**	-0.0067	-0.0022	-0.00033	-0.0038
12		1	**0.5715**	**0.5373**	**0.5583**	**0.5911**	-0.0061	-0.0011	-0.00203	-0.00068
13			1	**0.5880**	**0.5795**	**0.5091**	-0.0020	0.00055	-0.00098	0.00066
14				1	**0.5444**	**0.5882**	-0.0074	0.00044	-0.00224	0.00077
15					1	**0.5650**	-0.00077	-0.0013	-0.00062	0.00011
16						1	-0.0037	0.00225	-0.000781	0.00017
17							1	**0.33032**	**0.3620**	**0.37489**
18								1	**0.3085**	**0.3736**
19									1	**0.32723**
20										1

6.2. Análisis del tiempo de ejecución de dudi.pca

Para conocer el tiempo de ejecución de la función dudi.pca se generaron matrices de distintos tamaños variando la cantidad de individuos y la cantidad de variables. Para cada una de estas matrices se realizó el análisis de componentes principales mediante la función dudi.pca, los tiempos de ejecución se calcularon mediante la función system.time que está contenida en el programa estadístico R.

A continuación, en la tabla **6.7** se muestran los resultados del tiempo de demora de

la función dudi.pca dado un número de individuos y un número de variables.

Tabla 6.7: Tiempos de ejecución en **segundos** de dudi.pca

	Variables					
Individuos	**10**	**15**	**20**	**25**	**30**	**35**
50.000	0.22	0.34	0.54	0.69	0.97	0.99
500.000	3.31	4.25	5.76	7.39	9.21	20.6
1.000.000	5.72	9.24	12.14	16.23	19.66	22.93
2.000.000	11.74	26.02	28.04	89.02	153.42	-
3.000.000	20.36	63.8	111.23	-	-	-
4.000.000	28.92	97.52	-	-	-	-
5.000.000	34.72	-	-	-	-	-
6.000.000	45.05	-	-	-	-	-

En la tabla **6.7** se observan algunas celdas que contienen un guion, este guion indica que no se pudo realizar un ACP mediante la función dudi.pca a la matriz simulada con la dimensión correspondiente a esa celda. Para estas matrices simuladas la función arrojó el siguiente mensaje: "cannot allocate vector of size 1024.0 Mb", el cual significa que R no pudo alojar un vector de tamaño 1024.0 Mb.

Este tipo de mensajes de error indican una falla al obtener memoria, ya sea porque el tamaño excedió el límite de espacio de direcciones para un proceso o, porque el sistema no pudo proporcionar la memoria, el cual es el caso más probable (R Core Team, 2017).

La función dudi.pca se ve más afectada cuando se aumenta el número de variables que cuando se aumenta el número de individuos, como se puede observar en la tabla **6.7** esta función arroja resultados hasta con 6 millones de individuos y 10 variables, pero con 2 millones individuos y 35 variables, la función no logra entregar resultados.

6.3. Creación del algoritmo

6.3.1. Rango de una matriz

Con la base de datos simulada de 100..000 individuos y 20 variables, se omitieron tres variables las cuales fueron remplazadas por variables dependientes usando algunas columnas de la matriz de datos. Esto con el fin de ilustrar la necesidad del cálculo del rango de la matriz para ayudar a optimizar el algoritmo NIPALS.

En la tabla **6.9** se observan los valores propios resultantes al calcular los 20 componentes principales de la matriz que incluye las variables dependientes, en este caso, el número

de componentes a calcular es el número de columnas de la base de datos. Como era de esperarse, los tres últimos valores propios dieron cero y representa para el algoritmo 221 iteraciones de más (ver tabla **6.8**) con un tiempo de 3.4 minutos a comparación del tiempo de 4.5 minutos que resultó al calcular solo el número de componentes principales que es igual al rango de la matriz (17 componentes), lo que equivaldría a un aumento del tiempo de 1.1 minutos.

Tabla 6.8: Cantidad de iteraciones por columna (variables)

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
92	40	34	326	513	280	669	361	575	821	603	558	152	704	139	1128	2	8	97	116

Tabla 6.9: Valores propios de la matriz con variables dependientes

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
6.5841	4.8649	2.0391	0.7097	0.6448	0.6064	0.5464	0.52169	0.4739	0.4496

11	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
0.4346	0.4138	0.3904	0.3183	0.3065	0.24837	0.2419	0	0	0

Aunque la diferencia del tiempo no es tan elevada cuando la cantidad de componentes principales a calcular es el número de variables dependiente e independientes respecto a calcular la cantidad de componentes principales igual al rango, se evidencia fácilmente que por cada columna dependiente NIPALS tendrá trabajo computacional extra que realizar. A medida que la cantidad de individuos se eleve como también las columnas dependientes las diferencias aumentarán rápidamente y por lo tanto, es mejor evitar trabajo de más cuyo resultado en la práctica no será útil.

6.3.2. Cálculo del error de precisión ε

Para el respectivo análisis de ortogonalidad del algoritmo NIPALS y mostrar que en efecto esta depende únicamente del error de precisión ε que defina el investigador, se tiene en la tabla **6.10** la multiplicación de las componentes principales para una base con 100 individuos y 5 variables utilizando un $\varepsilon = 0.01$, que para el caso donde $t_i \neq t_j$ deben ser igual a 0 para garantizar la ortogonalidad; como se puede ver todas las multiplicaciones dan valores cercanos a cero, no obstante esta proximidad se observa más en las componentes que se encuentran distantes entre sí como $t_1 * t_5$, mientras que para las componentes contiguos entre sí, la ortogonalidad no es igual de próxima a cero, como ocurre entre t_1 y t_2 o t_4 y t_5.

Tabla 6.10: Ortogonalidad entre componentes matriz 100 x 5 con $\varepsilon = 0.01$

Componentes	t1	t2	t3	t4	t5
t1	$4.08e^{2}$	$6.47e^{-2}$	$3.46e^{-3}$	$-7.58e^{-5}$	$-1.27e^{-5}$
t2	-	$5.05e^{1}$	$3.03e^{-2}$	$9.86e^{-4}$	$1.73e^{-6}$
t3	-	-	$2.12e^{1}$	$-3.44e^{-2}$	$-4.31e^{-5}$
t4	-	-	-	15.0933	$1.26e^{-2}$
t5	-	-	-	-	$5.06e^{0}$

Ahora, la tabla **6.11** muestra la ortogonalidad entre componentes principales hallados a partir del algoritmo NIPALS, esta vez a partir de una base con las mismas dimensiones que en la tabla **6.10** pero fijando un error de precisión para el algoritmo de $\varepsilon = 0.00001$ y tal como se puede ver la ortogonalidad en las componentes ahora es más cercana a cero, logrando converger a cero a medida que se aumenta el error y como se puede notar las componentes contiguas son más cercanos a cero que en el caso anterior, comparando el primer por el segundo componente, en el caso anterior era igual a 0.0647 y ahora es igual a $1.5185e^{-5}$; lo anterior muestra entonces que al fijar ε en un valor pequeño, el nivel de ortogonalidad entre componentes empieza a converger a cero.

Tabla 6.11: Ortogonalidad entre componentes matriz 100 x 5 con $\varepsilon = 0.00001$

Componentes	t1	t2	t3	t4	t5
t1	$4.08e^{2}$	$1.52e^{-5}$	$2.49e^{-8}$	$-1.10e^{-10}$	$-3.16e^{-13}$
t2	-	$5.05e^{1}$	$3.04e^{-5}$	$7.36e^{-8}$	$1.41e^{-13}$
t3	-	-	$2.12e^{1}$	$-3.68e^{-5}$	$-6.54e^{-11}$
t4	-	-	-	15.0933	$1.78e^{-5}$
t5	-	-	-	-	$5.06e^{0}$

Tabla 6.12: ε vs número de individuos de matriz, fijando un nivel de ortogonalidad

n	$t_1 * t_2$	ε
10	-3,28 e^{-9}	$\frac{1}{10^{8}}$
100	-2,17 e^{-9}	$\frac{1}{10^{9}}$
1000	-6,04 e^{-9}	$\frac{1}{10^{10}}$
10000	-1,75 e^{-9}	$\frac{1}{10^{11}}$
100000	2,64 e^{-9}	$\frac{1}{10^{12}}$

Con el fin de determinar cuál debe ser el nivel de precisión para el algoritmo y que este se encuentre en función del tamaño de la matriz de datos, se corrieron varias simulaciones variando el número de individuos en la matriz de datos, a fin de que la multiplicación entre las componentes principales contiguos, en este caso la multiplicación del primer y el segundo

componente principal sea igual o menor a $9.9e^{-8}$. En la tabla **6.12** se muestran los resultados tomando tamaños de muestra de 10, 100, 1.000, 10.000 y 100.000 fijando el número de variables (columnas) en 20, es posible observar que a mayor número de individuos n, menor debe ser el error de precisión si se quiere mantener un nivel de ortogonalidad constante; tal y como se puede apreciar al aumentar el número de individuos en un dígito (pasar de decenas a centenas o de centenas a miles), el nivel de error ε se hace diez veces más pequeño, se llega entonces a que el nivel de error está definido por la ecuación 6.1

$$\varepsilon = \frac{1}{10^8 10^d} \tag{6.1}$$

Donde d es la cantidad de dígitos asociados al número de individuos de la matriz de datos y donde ε es el nivel de error que se le debe fijar al algoritmo NIPALS para mantener la ortogonalidad entre el primer y el segundo componente principal, menor a $9.9e^{-8}$.

Ahora puesto que ya se analizó cómo influye el aumento en el número de individuos a la ortogonalidad y se obtuvo una ecuación que determina un determinado nivel de precisión ε de acuerdo a este, se desea ahora observar cómo influye el número de variables en la misma ortogonalidad, por lo tanto como se puede ver en la tabla **6.13** se tiene la multiplicación de los dos primeros componentes principales en una matriz de 1.000 individuos con 10, 15, 20, 25 y 30 variables respectivamente, así se puede observar que a pesar de que se aumenta el número de variables, el nivel de ortogonalidad se mantiene, notándose que es muy similar en todos los casos y comprobando que ante el número de variables que se trabaja en este proyecto, el número de variables no influye en la pérdida de ortogonalidad de las componentes principales.

Tabla 6.13: ε vs número de variables en matriz, fijando un nivel de ortogonalidad

Variables	$t1 * t2$	ε
10	$-4.69e^{-9}$	$\frac{1}{10^{10}}$
15	$-5.61e^{-9}$	$\frac{1}{10^{10}}$
20	$-6.03e^{-9}$	$\frac{1}{10^{10}}$
25	$-6.51e^{-9}$	$\frac{1}{10^{10}}$
30	$7.11e^{-9}$	$\frac{1}{10^{10}}$

Otra alternativa que se tomó en cuenta fue el algoritmo propuesto por (Andrecut, 2009) basado en el proceso de re-ortogonalización de Gram Schmidt (GS). Se simularon varias matrices con diferentes tamaños para comparar ambos métodos NIPALS-GS y el algoritmo vía NIPALS para grandes volúmenes de datos. En el primer método se fijó un error de precisión de 0.0000001,contrario a NIPALS que usa un error variante resultante de la ecuación 6.1 que está en función del número de individuos.

En la tabla **6.14** se puede observar como el tiempo de procesamiento para NIPALS-GS es mayor cuando las matrices aumentan de tamaño. En una matriz pequeña de 100 individuos con 10, 20 y 30 variables NIPALS-GS alcanza a ser igual o menor en tiempo que NIPALS. Pero, cuando se comienzan a generar matrices de mayor dimensión NIPALS-GS comienza a tardar muchas veces más que el algoritmo NIPALS. Por ejemplo, con 5.000 individuos y 20 variables NIPALS-GS tardó aproximadamente 24.28 veces más, cuyo tiempo fue de 4.6 minutos frente a 10.3 segundos, respectivamente.

Tabla 6.14: Tiempos de ejecución NIPALS vs NIPALS-GS

Nipals \ Nipals-GS	Variables		
Individuos	10	20	30
100	0.08 sg 0.08 sg	0.34 sg 0.29 sg	0.55 sg 0.52 sg
1000	0.33 sg 1.66 sg	2.36 sg 11 sg	8.75 sg 32.97 sg
5000	1.69 sg 37.11 sg	10.03 sg 4.06 min	25.94 sg 21.46 min
10000	2.60 sg 2.59 min	1.60 min 13.88 min	1.26 min 30.76 min
50000	14.56 sg -	3.6 min -	7.87 min -

En matrices de 50000 individuos con 10, 20 y 30 variables, NIPALS-GS no pudo arrojar resultados puesto que el algoritmo crea matrices de gran dimensión para contener todos los vectores que se necesitan para realizar el proceso de Gram-Schmidt, por lo tanto el programa estadístico R no pudo alojar esas matrices de gran dimensión.

Se compararon los resultados de ambos métodos cuyos valores propios resultaron idénticos, componentes principales y vectores propios obtenidos resultaron muy semejantes respecto a ambos métodos. Posterior a esto, se analizó el nivel de ortogonalidad entre NIPALS y GS-NIPALS, encontrando muy buenos resultados en ambos casos.

En la tabla **6.15** se anotó el resultado de la multiplicación de los dos primeros componentes visualizando las diferencias que existen entre la ortogonalidad de los métodos comparados. En un principio, la multiplicación tiende más a cero (-5.329071e^{-15})con la ayuda de Gram-Schmidt en una matriz pequeña con 100 individuos para 10, 20 y 30 variables. A medida que aumenta el número de individuos, el algoritmo NIPALS-GS no logra mantener la misma ortogonalidad el cual sí se presenta en el algoritmo NIPALS, puesto que este último tiene un ε que tiene en cuenta el incremento del tamaño de la matriz.

Al finalizar el estudio NIPALS-GS pasó de -5.329071e^{-15} a -1.085763 e^{-12} perdiendo ortogonalidad al aumentar el tamaño de la matriz, en cambio NIPALS se mantuvo con una ortogonalidad que inició en -2.469882e^{-08} y terminó en -2.354444e^{-09}.

El algoritmo NIPALS-GS obtiene una buena ortogonalidad en sus componentes principales con un ε pequeño pero re-ortogonalizar parece implicar un mayor esfuerzo computacional y un aumento en el tiempo de ejecución bastante elevada. Además, en el transcurso del estudio se observó que las iteraciones para este método se redujeron significativamente, lo cual implica que re-ortogonalizar dentro de estas iteraciones demora más que aumentando el número de iteraciones para la convergencia. Es de resaltar que aunque la velocidad de procesamiento depende en gran medida de la máquina empleada y se podría pensar que en un determinado momento la diferencia entre los dos algoritmos sea insignificativa, es claro que NIPALS-GS es un algoritmo que realiza un paso adicional en el proceso de re-ortogonalizar y que independiente de la máquina empleada, la sencillez con que esta construido el algoritmo NIPALS y ante un ε apropiado, este es un algoritmo más fácil de emplear y más eficiente en cuanto a velocidad de procesamiento se habla.

Tabla 6.15: Multiplicación de los dos primeros componentes principales: NIPALS vs NIPALS-GS

Nipals\ Nipals-GS	Variables		
Individuos	10	20	30
100	-1.214930e^{-08} -5.329071e^{-15}	-2.173999e^{-09} 1.176836e^{-14}	-2.469882e^{-08} 1.5241e^{-13}
1000	-4.690096e^{-09} 5.77316e^{-15}	-6.039826e^{-09} 2.4869e^{-13}	-7.111863e^{-08} 6.7501e^{-12}
5000	-1.105416e^{-08} -3.002043e^{-13}	-1.456292e^{-08} -1.04805e^{-12}	-1.573864e^{-08} -3.44002 e^{-12}
10000	-1.653424ee^{-09} -4.443113e^{-13}	-1.749030e^{-09} 1.624922e^{-12}	-2.354444e^{-09} -1.085763 e^{-12}
50000	-3.738890e^{-09} -	-3.746255e^{-09} -	-6.044518e^{-09} -

Ya habiendo realizado el respectivo análisis para la ortogonalidad de las componentes principales, se hace este mismo análisis para los vectores propios, que de igual forma deben cumplir con ser ortogonales entre sí y al multiplicarse entre ellos mismos, deben cumplir con ser unitarios $u_i' * u_i = 1$, así, mediante la misma ecuación que resume el error de precisión para la ortogonalidad de las componentes principales, se probó la ortogonalidad de los vectores propios donde la tabla **6.16** muestra la ortogonalidad entre los primeros 5 vectores propios de una matriz de 1.000 individuos con 20 variables.

Tabla 6.16: Ortogononalidad entre vectores de una matriz 1000 * 20

Vectores propios	**P1**	**P2**	**P3**	**P4**	**P5**
P1	1	2,4937 e^{-18}	1,7347e^{-18}	2,4286 e^{-17}	6,9389 e^{-18}
P2	-	1	5,5836 e^{-18}	1,9949 e^{-17}	2,4286 e^{-17}
P3	-	-	1	3,8164 e^{-17}	3,4694 e^{-17}
P4	-	-	-	1	1,6653 e^{-17}
P5	-	-	-	-	1

6.4. Análisis del algoritmo vía NIPALS

En esta sección, se analizarán los tiempos de ejecución y el número de iteraciones del algoritmo vía NIPALS variando el número de individuos y número de variables, pero manteniendo el rango de los parámetros y las dependencias de las variables ya anteriormente usadas en la matriz de 100000 individuos y 20 variables. Además, para esta última matriz se analizarán los valores propios, vectores propios y componentes principales verificando y comparando que sus resultados sean iguales o cercanos a los entregados por dudi.pca.

En la tabla **6.17** se muestra los tiempos de ejecución entregados por el algoritmo para los diferentes tamaños.

Tabla 6.17: Tiempos de ejecución del algoritmo vía NIPALS

	Variables					
Individuos	**10**	**15**	**20**	**25**	**30**	**35**
100	0.08 sg	0.11 sg	0.34 sg	0.35 sg	0.55 sg	0.70 sg
1000	0.33 sg	0.64 sg	2.36 sg	3.89 sg	8.75 sg	7.94 sg
5000	1.69 sg	2.25 sg	10.03 sg	16.12 sg	25.94 sg	39.36 sg
10000	2.60 sg	5.05 sg	15.67 sg	32.22 sg	1.26 min	1.95 min
50000	14.56 sg	32.15 sg	1.60 min	5.28 min	7.87 min	27.22 min
100000	31.53 sg	63.87 sg	3.6 min	9.06 min	17.10 min	73.24 min

Los tiempos mostrados en la tabla **6.17** incrementan a medida que aumentan las variables y los individuos. El tiempo se ve más afectando cuando tanto como las variables y los individuos aumentan a la vez más que cuando una de estas se mantienen fijas mientras la otra aumenta.

En la tabla **6.18** se muestra la cantidad de iteraciones que se realizaron para alcanzar la convergencia de los vectores y componentes principales. Se observa que no existe algún patrón en las iteraciones cuando los individuos aumentan de tamaño. Se pensaría que por cada aumento de individuos las iteraciones deberían aumentar, pero como se observa en el caso de 1000 y 5000 individuos, las iteraciones disminuyeron. Cuando las variables aumentan de tamaño el número de iteraciones comienza a crecer.

Tabla 6.18: Número de iteraciones

	Variables					
Individuos	**10**	**15**	**20**	**25**	**30**	**35**
100	726	1257	5557	5372	5878	8230
1000	1364	1993	6745	10641	20705	16394
5000	1965	1807	7778	5802	12580	25945
10000	1508	2067	5904	8896	17345	20836
50000	1593	2114	6509	15580	18118	57822
100000	1714	2351	7703	15041	22392	84506

A continuación, se muestra en la tabla **6.19** el tiempo por iteración que demora en cada una de las matrices simuladas. El tiempo de una iteración de una matriz pequeña de 100 individuos y 10 variables fue de 0.00011 segundos, cuando la matriz cambió a ser de 100.000 individuos y 35 variables el tiempo cambió a 0.052 segundos, por lo tanto, cada vez que aumenta el tamaño de la matriz las iteraciones cada vez demoran más. Esto puede deberse a que las operaciones que están dentro de las iteraciones se tornarán más complejas debido al gran tamaño de las columnas que posee la matriz, por lo tanto si se desea aumentar la rapidez por iteración se debe aumentar la velocidad de las operaciones.

Tabla 6.19: Tiempo por iteracion en segundos

	Variables					
Individuos	**10**	**15**	**20**	**25**	**30**	**35**
100	0.00011	0.000087	0.00006	0.000065	0.0000935	0.000085
1000	0.000241	0.000321	0.000349	0.00151	0.000422	0.000484
5000	0.00086	0.00124	0.00128	0.002778	0.002062	0.00246
10000	0.001724	0.00244	0.002654	0.00362	0.00435	0.00561
50000	0.009655	0.015	0.014	0.0203	0.026	0.02824
100000	0.01839	0.027	0.028	0.1078	0.0458	0.052

Para la matriz fijada en un comienzo de tamaño 100.000x20 se implementó el algoritmo NIPALS dando como resultado los siguientes valores propios (ver tabla **6.20**):

Tabla 6.20: Valores propios de NIPALS

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
6.5841	4.8649	2.0391	0.7097	0.6448	0.6064	0.5464	0.52169	0.4739	0.4496

11	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
0.4346	0.4138	0.3904	0.3183	0.3065	0.24837	0.2419	0.1238	0.0754	0.0053

Los valores propios resultantes del algoritmo NIPALS dieron exactamente iguales a los valores propios arrojados por la función dudi.pca que se observan en la tabla **6.4**. Al sumar los tres primeros valores propios con más inercia se obtiene 13.4881, lo que estaría indicando que los tres primeros ejes factoriales recogen cerca del 67.44 % de la inercia total.

De la misma forma los vectores propios y componentes principales resultaron bastante aproximados entre NIPALS y dudi.pca. La media de las diferencias de los vectores propios y componentes principales entre estos dos métodos fue de aproximadamente $2.685782e^{-15}$ y $4.94101e^{-16}$, respectivamente.

En la tabla **6.21** y **6.22** se muestran solo los 5 primeros valores de los 6 primeros vectores propios y componentes principales de NIPALS.

Tabla 6.21: Vectores propios NIPALS

1	2	3	4	5	6
0.3507698764	$1.304412\ e^{-0.3}$	$1.538881e^{-0.4}$	-0.0011301388	$3.683821\ e^{-0.3}$	-0.0031251521
0.3468104332	$1.552336e^{-0.4}$	$-2.737713e^{-0.4}$	0.0009783271	$-6.462997e^{-0.4}$	-0.0051548561
0.3502998598	$6.889612e^{-0.4}$	$5.918985e^{-0.4}$	-0.0018660572	$-3.717854e^{-0.3}$	0.0081354960
0.3558376947	$1.030494e^{-0.3}$	$-2.260791e^{-0.4}$	0.0022486522	$-3.585334e^{-0.3}$	0.0016765093
0.3460209366	$9.571208e^{-0.4}$	$1.433925e^{-0.3}$	0.0034707431	$1.059976e^{-0.3}$	-0.0037389803

Tabla 6.22: Componentes principales NIPALS

1	2	3	4	5	6
3.969771	1.310391	0.8636024	-1.074985	-0.1489145	-0.9376484
-1.551815	-1.078336	0.2174479	0.03162996	-0.9957016	0.1725546
3.570599	2.587299	-0.7578205	1.861531	-0.6695214	0.3031023
3.470740	1.293961	-0.7443158	-0.4804690	-0.5314552	-0.1420161
-0.2336147	-0.8403335	-4.124021	0.7700732	0.6459060	0.3367533

El producto punto entre pares de vectores propios distintos dan valores entre $1e^{-15}$ y $1e^{-18}$ que al ser muy cercanos a cero son ortogonales. En la tabla **6.23**, se muestra la ortogonalidad de los 6 primeros vectores propios.

Tabla 6.23: Ortogonalidad de los vectores

Variables	1	2	3	4	5	6
1	1	$2.154958e^{-18}$	$-1.284780e^{-17}$	$-7.535205e^{-18}$	$-1.301043e^{-17}$	$3.740497e^{-18}$
2		1	$2.580401e^{-17}$	$-5.876376e^{-17}$	$1.843144e^{-17}$	$2.320193e^{-17}$
3			1	$-1.110223e^{-15}$	$3.053113e^{-16}$	$3.330669e^{-16}$
4				1	$-3.316791e^{-15}$	$3.608225e^{-16}$
5					1	$7.771561e^{-16}$
6						1

La ortogonalidad de las componentes se mantuvieron entre un valor fijo del error de precisión el cual fue de1 e^{-12}, la ortogonalidad estuvo entre los valores de $1e^{-09}$ y $1e^{-12}$. En la tabla **6.24** se muestra la ortogonalidad entre los 6 primeros componentes principales.

Tabla 6.24: Ortogonalidad de las componentes principales

Vectores propios	1	2	3	4	5	6
1	6.584159	$5.077867e^{-09}$	$-3.578471e^{-12}$	$-5.536904e^{-12}$	$4.974687e^{-11}$	$2.506440e^{-12}$
2		4.864913	$3.904619e^{-10}$	$-1.494338e^{-11}$	$-3.899103e^{-13}$	$2.461320e^{-11}$
3			2.039148	$-3.353211e^{-10}$	$-4.588250e^{-10}$	$-6.499319e^{-10}$
4				7.097984	$2.372009e^{-10}$	$1.822422e^{-10}$
5					6.448650	$2.036606e^{-10}$
6						6.064730

6.5. Aplicación del algoritmo NIPALS con una matriz de gran dimensión

Se creó una matriz con 1000000 individuos y 20 variables para aplicar el algoritmo NIPALS para grandes volúmenes de datos. El error correspondiente para este tamaño de la matriz fue de $1e^{-13}$, a partir de este error se obtuvieron los valores propios , vectores propios y componentes principales.
El tiempo de ejecución que tomó el algoritmo para encontrar todos las componentes principales fue 40.12 minutos.

La matriz de correlaciones resultante de la base de datos de 1.000.000 x 20 se muestra en las tablas **6.25** y **6.26**. Se observa claramente correlaciones de todo tipo, muy altas, altas, moderadas y bajas.

Tabla 6.25: Matriz 1 de correlaciones de la matriz 1.000.000 x 20

Variables	1	2	3	4	5	6	7	8	9	10
1	1	0.836606	0.805372	0.809506	0.826233	0.739183	0.945251	0.83215	-0.000179	0,001112
2		1	0,736572	0.86173	0.782448	0.874946	0.790798	0.743835	-0.000245	0,0008209
3			1	0.742142	0.796706	0.770436	0.861962	0.767474	0.000082	0.000846
4				1	0.837916	0,761658	0.812991	0.901886	-0.000676	0.000767
5					1	0.871459	0.76413	0.872867	-0.000238	0.000317
6						1	0.700904	0.728926	-0.000034	0.000119
7							1	0.789599	-0,000055	0.001239
8								1	0.501352	0.684664
9									1	-0.000482
10										1
11										
12										
13										
14										
15										
16										
17										
18										
19										
20										

Tabla 6.26: Matriz 2 de correlaciones de la matriz 1.000.000 x 20

Variables	11	12	13	14	15	16	17	18	19	20
1	0.000792	-0.000002	-0.000336	0.001074	0.000274	0.000867	-0.000801	0.001467	0.001783	0.000454
2	0.000230	-0.000656	-0.000935	0.000819	0.000187	0.001072	-0.000585	0.001337	0.002051	-0.000029
3	0.000347	-0.000856	-0.000023	0.000837	-0.000736	0.000904	-0.001033	0.001562	0.001837	0.00013
4	0.000212	-0.001219	-0.00076	0.001140	0.000142	0.000946	-0.0001003	0.001065	0.001428	-0.000697
5	0.000461	-0.000849	-0.000216	0.000735	-0.000622	0.000643	-0.000608	0.001133	0.00159	-0.000537
6	0.000462	-0.000736	-0.00058	0.000526	-0.0004771	0.000664	-0.0007847	0.001413	0.002202	-0.000601
7	0.000618	-0.000273	-0.000153	0.001039	0.0001098	0.000880	-0.0005595	0.001503	0.001392	0.0000301
8	0.000616	-0.000906	-0.000188	0.000909	-0.000094	0.000837	-0.0001644	0.001134	0.001721	-0.000155
9	0.684664	0.674867	0.670942	0.633259	0.603481	0.563987	-0.0006847	-0.00075	0.000126	0.000352
10	0.581236	0.559257	0.5951005	0.597086	0.55954	0.6603701	0.0015929	0.0012460	0.0015886	0.00045783
11	1	0.605989	0.605337	0.595944	0.570820	0.536547	-0.001617	-0.0014854	-0.0002943	0.0002338
12		1	0.657548	0.508042	0.6.8566	0.628684	-0.0004538	0.0002341	0.001109	0.00012604
13			1	0.532368	0.5.53105	0.6349493	-0.00009913	0.000890	0.0012527	-0.00078232
14				1	0.5.28065	0.6825813	-0.0004338	-0.000219	-0.0006596	0.00034699
15					1	0.54316	-0.0013509	-0.000586	-0.00004348	-0.00065134
16						1	-0.0003733	0.00053889	0.00067357	0.000845400
17							1	0.482724	0.42204	0.341660
18								1	0.42620	0.3865
19									1	0.363140
20										1

Se puede apreciar en la tabla **6.27** que los valores propios resultantes de NIPALS son decrecientes y la suma de ellos son 20, lo cual indica que el cálculo de los valores propios se realizó de manera correcta. Observando los tres primeros valores se obtiene que la suma de ellos es igual a 14.0493, lo cual indica que los tres primeros ejes factoriales recogen cerca del 70.25 % de la inercia total.

Tabla 6.27: Valores propios de NIPALS-1 millón de individuos y 20 variables

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
6.6435	5.1910	2.2148	0.6795	0.64238	0.5932	0.5252	0.5124	0.45708	0.4334

11	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
0.42125	0.34345	0.30027	0.2873	0.25527	0.2074	0.1634	0.0687	0.0488	0.011

Los vectores propios se multiplicaron entre ellos para observar la ortogonalidad presente. Los resultados indican que los vectores propios son ortogonales entre sí a pesar de que la matriz analizada haya pasado de 100000 a 1000000 de individuos con 20 variables. En la tabla **6.28** se muestra solo los resultados de los 8 primeros vectores propios, cuya ortogonalidad se encuentra entre los valores $1.168905e^{-19}$ y $6.897261e^{-15}$.

Tabla 6.28: Ortogonalidad de los vectores- 1 millón de individuos y 20 variables

Variables	1	2	3	4	5	6	7	8
1	1	$6.431298e^{-19}$	$1.214476e^{-17}$	$-1.362368e^{-17}$	$2.394364e^{-18}$	$-2.534386e^{-17}$	$-2.386213e^{-17}$	$1.323807e^{-17}$
2		1	$7.182839e^{-18}$	$5.827587e^{-19}$	$-1.858549e^{-15}$	$1.575820e^{-17}$	$6.756009e^{-15}$	$1.168905e^{-19}$
3			1	$-2.664535e^{-15}$	$3.209238e^{-17}$	$5.558054e^{-15}$	$5.837616e^{-16}$	$6.897261e^{-15}$
4			1	$4.597017e^{-17}$	$3.608225e^{-16}$	$-9.853229e^{-16}$	$-2.457615e^{-16}$	$1.221245e^{-15}$
5				1	$7.771561e^{-16}$	$-4.575333e^{-17}$	$-3.291429e^{-15}$	$2.200930e^{-16}$
6					1	$4.003620e^{-16}$	$4.447831e^{-15}$	
7							$-1.991442e^{-16}$	
8							1	

Al analizar la ortogonalidad de las componentes principales mostrados en la tabla **6.29** se encontró que estos mantienen la ortogonalidad entre los valores aproximados a $9.906118e^{-12}$ y $1.861448e^{-08}$, en general todos las ortogonalidades resultaron menores a $1e^{0.8}$, lo que indica que el error de precisión fijado por la ecuación 6.1 mantiene la ortogonalidad a pesar del aumento del tamaño de la matriz.

Tabla 6.29: Ortogonalidad de las componentes principales- 1 millón de individuos y 20 variables

Variables	1	2	3	4	5	6	7	8
1	6.643529	$-2.625207e^{-10}$	$-8.462031e^{-11}$	$-1.082782e^{-10}$	$-1.530693e^{-10}$	$-8.072321e^{-12}$	$5.401786e^{-11}$	$5.966294e^{-11}$
2		5.191050	$-2.057412e^{-10}$	$-5.292145e^{-11}$	$1.808935e^{-08}$	$-2.782812e^{-10}$	$-1.139317e^{-09}$	$-2.000236e^{-10}$
3			2.214820	$-4.359111e^{-09}$	$-7.660472e^{-11}$	$-2.622517e^{-09}$	$-8.045891e^{-10}$	$1.861448e^{-08}$
4				6.795224	$2.812367e^{-11}$	$5.159544e^{-10}$	$3.667793e^{-11}$	$1.020106e^{-09}$
5					6.423831	$-3.580786e^{-11}$	$-3.751136e^{-09}$	$-1.236891e^{-10}$
6						5.932290	$9.906118e^{-12}$	$9.551625e^{-10}$
7							5.252403	$-2.577521e^{-10}$
8								5.124234

A pesar de la gran dimensión de la matriz de datos, los vectores propios y componentes principales mantuvieron la ortogonalidad deseada. Los valores propios se calcularon

correctamente y el tiempo de ejecución que demoró NIPALS para entregar los resultados aumentó significativamente pero esto es debido al tamaño de la matriz estudiada.

Capítulo 7

Conclusiones

En este trabajo se implementó un algoritmo vía NIPALS para grandes volúmenes de datos analizando los resultados a través de matrices de diferentes tamaños para luego aplicar el algoritmo a una matriz de 1000000 de individuos y 20 variables. De los resultados obtenidos en el capitulo 6 se llega a una serie de conclusiones mostradas a continuación.

7.1. Conclusiones generales

La función dudi.pca del paquete ade4 encargada de realizar análisis de componentes principales en el software estadístico R sufre problemas de almacenamiento ante grandes volúmenes de datos, pues no es capaz de almacenar vectores de alta dimensionalidad, adicional a ello esta se encuentra más afectada por el número de variables que por el número de individuos, siendo capaz de procesar a lo sumo una matriz con 2.000.000 de individuos y 30 variables. A través de éste proyecto se logro consolidar una propuesta algorítmica vía NIPALS en base a la partición de matrices que sirve de insumo a proyectos que trabajen con el procesamiento en paralelo o distribuido a fin de lograr la realización del análisis de componentes principales a grandes volúmenes de datos, solucionando los problemas de almacenamiento existentes cuando se realiza ACP a través de la matriz de correlaciones.

Se encontró inicialmente que la ortogonalidad entre componentes se ve más afectada en las componentes continuas, al hacer el producto de la primera con la segunda componente, la segunda con la tercera y así sucesivamente. Sin embargo, el algoritmo permite a medida que se adecua el error de precisión ε a los tamaños de las matrices de entrada, dar solución a esta problemática manteniendo la ortogonalidad entre todas las componentes principales; cabe resaltar que el error de precisión mostró ser sensible ante cambios en el número de individuos, sin embargo respecto a las variables, el número de estas no afecta de forma significativa el rendimiento.

El método NIPALS-GS mostró ser menos efectivo que el algoritmo vía NIPALS en relación al tiempo de procesamiento, esto gracias a que el primero utiliza la reortogonalización de Gram–Schmidt y se encarga de que los componentes y vectores propios sean ortogonales una vez que el algoritmo NIPALS ya lo hace.

7.2. Recomendaciones

Como ya se mencionó anteriormente dentro de este mismo trabajo, el análisis de grandes volúmenes de datos puede ser abarcado desde distintos enfoques; se encuentra el caso donde el número de individuos es mucho más grande que el número de variables ($n >> p$), caso que fue estudiado a lo largo de este trabajo, otro enfoque desde el que puede abarcarse este problema es donde el número de variables es mucho mayor al número de individuos ($p >> n$) y para finalizar está el caso donde tanto el número de variables como de individuos son de gran dimensión; cada uno de estos enfoques puede resultar de interés ante distintas problemáticas en las diferentes áreas del conocimiento, por lo que para futuros proyectos abarcar la problemática desde estos tipos de enfoques puede resultar muy interesante.

En este trabajo se logró consolidar un algoritmo para realizar análisis de componentes principales vía NIPALS, no obstante este se vio limitado en ocasiones debido a que a mayor número de variables e individuos, mayor era el coste computacional y el tiempo de procesamiento se elevaba drásticamente debido a las características propias de la máquina con que se trabajó, por lo tanto adaptar este algoritmo a un sistema de cómputo en paralelo (interconexión de varias máquinas para resolver un mismo problema) o distribuido (uso de los diferentes procesadores de la máquina para distribuir las labores) podría ser sumamente útil, más aún cuando la creciente en volúmenes de datos es cada vez mayor.

Por otra parte, este proyecto se desarrolló exclusivamente sobre el análisis de componentes principales y los resultados fueron principalmente de tipo teórico ante un caso de estudio simulado, por lo tanto sería interesante adaptar los resultados de este proyecto a un caso de estudio real. Teniendo en cuenta que todas las técnicas de análisis multivariado parten del ACP, se podría hacer uso del algoritmo en otras técnicas, como podrían ser el análisis de correspondencias, análisis factorial simple y múltiple, entre otros.

A. Código en R del algoritmo propuesto vía NIPALS

```
### CREACIÓN DE LA MATRIZ SIMULADA ######

library(copula)
library(corpcor)
mat<-function(p,n,d1,d2){
  v=((p-1)*p)/(2)
  uni11<-runif(v,min=d1,max=d2)
  norm.cop <- normalCopula(uni11, dim=p,dispstr="un")
  sig<-getSigma(norm.cop1)
  x1=is.positive.definite(sig)
  while(x1==!TRUE){
    uni11<-runif(v,min=d1,max=d2)
    norm.cop <- normalCopula(uni11, dim=p,dispstr="un")
    sig<-getSigma(norm.cop)
    x1=is.positive.definite(sig)
  }
  medias<-c(0:50); varianzas<-c(0.1:5)
  coef<-rep(list(list(mean=3,sd=10)),p)
  mediasc<-sample(medias,p); varianzasc<-sample(varianzas,p,replace=TRUE)
  for(i in 1:p){
    coef[i]=list(list(mean=mediasc[i],sd=varianzasc[i]))
  }
  nom<-rep("norm",p)
  mv.NE <- mvdc(norm.cop,nom,coef)
  x<- rMvdc(n, mv.NE)
  print(x)
}

mat(4,100,0.3,0.85)
```

```
### FIN - CREACIÓN DE LA MATRIZ SIMULADA ####
n=100
p=5
a<-qr(x)$rank
r<-2              ##número de particiones
coc=n%/%r
res=n%%r  ###par e impar
## CODIGO QUE FIJA EL ERROR DE ACUERDO AL TAMAÑO DE LA MATRIZ ##
nt=n
j=0
while(nt>=1){
  nt=nt/10
  j=j+1
}
di=j-2
nma=((1)/((10)^(8+di)))
## FIN DE CODIGO QUE FIJA EL ERROR DE ACUERDO AL TAMAÑO DE LA MATRIZ ##
#Estandarizar
Xo<-(t(x)-apply(x,2,mean))/apply(x,2,sd)
Xo<-sqrt(n/(n-1))*t(Xo)

T<- matrix(1,n,a)
P <- matrix(1,p,a)
contar<-c()
system.time(               ####para mirar el tiempo de ejecución
  for(h in 1:a){
    t1<-Xo[,h]
    miu<-0
    nm<-1
    IT<-0
    while(nm>nma){
      P12<-miu
      pp<-0
    if(res==0){
        for(i in 1:r){
         t11<-t1[(coc*(i-1)+1):(coc*i)]
          mylist<-Xo[(coc*(i-1)+1):(coc*i),1:p]
          pp<-pp+crossprod(mylist,t11)
        }
      }else{
```

```
        for(i in 1:(r-1)){
          t11<-t1[(coc*(i-1)+1):(coc*i)]
         mylist<-Xo[(coc*(i-1)+1):(coc*i),1:p]
          pp<-pp+crossprod(mylist,t11)
        }
        t11<-t1[(coc*(r-1)+1):n]
        mylist<-Xo[((coc*(r-1)+1):n),1:p]
        pp<-pp+crossprod(mylist,t11)

      }

      p2<-sum(t1*t1)
      Pli<-pp/p2
      P1<-Pli%*%(sqrt(colSums(Pli*Pli)))^(-1)
      t1<-Xo%*%P1
      lambda=t1
      miu<-lambda
      nm<-sqrt(sum((P12-lambda)^2))
      IT<-IT+1
    }
    contar[h]<-IT
    T[,h] <- t1
    P[,h] <- P1
    X1 <- Xo-tcrossprod(t1,P1) # deflacta
    Xo <- X1

  }
)
```

Bibliografía

Abraham, G. & Inouye, M. (2014). Fast principal component analysis of large-scale genome-wide data. *PloS one*, 9(4), e93766.

Aiu.edu (2016). Sistemas operativos, procesos concurrentes-unidad iii. [Internet; descargado 13-octubre-2016].

Aluja, B., Morineau, T., et al. (1999). *Aprender de los datos: el análisis de componentes principales: una aproximación desde el Data Mining*. Number Sirsi) i9788483120224.

Andrecut, M. (2009). Parallel gpu implementation of iterative pca algorithms. *Journal of Computational Biology*, 16(11), 1593–1599.

Ayyad, C., Mateu, J., & Porcu, E. (2008). Inferencia y modelización mediante cópulas. *Trabajo de investigación, Universitat Jaume, Departamento de Matemáticas, Castellón*.

Brent, R. P. & Luk, F. T. (1982). *A systolic architecture for almost linear-time solution of the symmetric eigenvalue problem*. Technical report, Cornell University.

Chavarro, D. M. (2012). Método para elegir una cópula arquimediana óptima. *Universidad Nacional de Colombia, Facultad de Ciencias, Departamento de Estadística, Bogotá, Colombia*.

Chessel, D., Dufour, A. B., Thioulouse, J., et al. (2004). The ade4 package-i-one-table methods. *R news*, 4(1), 5–10.

Dray, S. & Dufour, A. (2007). The ade4 package: implementing the duality diagram for ecologists. *Journal of Statistical Software*, 22(4), 1–20.

Dray, S., Dufour, A.-B., et al. (2007). The ade4 package: implementing the duality diagram for ecologists. *Journal of statistical software*, 22(4), 1–20.

Fauzi, S. (2011). Revisión de técnicas estadísticas para el análisis de datos de gran dimensión. Master's thesis, Universidad de Granada.

Figueroa Mata, G., Carrera Retana, L. E., & Jiménez Romero, A. (2012). Análisis de componentes principales en paralelo1.

Franco, T. L. & Hidalgo, R. (2003). *Análisis Estadístico de Datos de Caracterización Morfológica de Recursos Fitogenéticos-Boletin Tecnico IPGRI No. 8*. Number 8. Bioversity International.

Gartner (2017). Big data: cómo procesar grandes volumenes de información. http://www.eveliux.com/mx/Big-Data-como-procesar-grandes-volumenes-de-informacion.html.

Gaviria, S. (2016). Colombia entra a las grandes ligas del big data: Simøn gaviria muñoz. `https://www.dnp.gov.co/Paginas/"Colombia-entra-a-las-grandes-ligas-del-Big-Data"--Simón-Gaviria-Mu~noz-.aspx`.

González Rojas, V. M. (2014). Análisis conjunto de múltiples tablas de datos mixtos mediante pls.

Halko, N., Martinsson, P.-G., Shkolnisky, Y., & Tygert, M. (2011a). An algorithm for the principal component analysis of large data sets. *SIAM Journal on Scientific computing*, 33(5), 2580–2594.

Halko, N., Martinsson, P.-G., & Tropp, J. A. (2011b). Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2), 217–288.

Hernández, J. A. (2016). Métodos de reducción de dimensionalidad: Análisis comparativo de los métodos apc, acpp y acpk. *Uniciencia*, 30(1), 115–122.

Hilbert, M. & López, P. (2011). The world's technological capacity to store, communicate, and compute information. *science*, 332(6025), 60–65.

Hofert, M., Kojadinovic, I., Maechler, M., & Yan, J. (2017). *copula: Multivariate Dependence with Copulas*. R package version 0.999-16.

IBM (2017). Ibm watson work, tecnologia cognitiva en el puesto de trabajo. `https://www-03.ibm.com/press/es/es/pressrelease/51779.wss`.

Jackson, J. E. (2005). *A user's guide to principal components*, volume 587. John Wiley & Sons.

Jolliffe, I. (2002). *Principal component analysis*. Wiley Online Library.

Jolliffe, I. (2005). Principal component analysis: Wiley online library.

Jolliffe, I. T. (1986). Principal component analysis and factor analysis. In *Principal component analysis* (pp. 115–128). Springer.

Kettaneh, N., Berglund, A., & Wold, S. (2005). Pca and pls with very large data sets. *Computational Statistics & Data Analysis*, 48(1), 69–85.

Kolman, B., Saumeth, G. A. C., & Saenger, A. (1981). *Algebra lineal*. Technical report, Fondo Educativo Interamericano.

Kramer, R. (1998). *Chemometric techniques for quantitative analysis*. CRC Press.

Lanczos, C. (1950). *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*. United States Governm. Press Office Los Angeles, CA.

Lay, D. C., Murrieta, J. M., & Murrieta, J. E. M. (2007). *Algebra lineal y sus aplicaciones*. Pearson educación.

Lebart, L., PIRON, A., Lebart, M., Morineau, A., & Piron, M. (2000). *Statistique exploratoire multidimensionnelle*. Dunod,.

López, J. C. L. (2014). La moda del big data: ¿en qué consiste en realidad? url http://www.eleconomista.es/tecnologia/noticias/5578707/02/14/La-moda-del-Big-Data-En-que-cons Accedido 27-02-2014.

Martınez, H. & Pérez, R. (1990). Introducción al álgebra lineal numérica. *Universidad del Cauca*.

Martinsson, P.-G. & Voronin, S. (2016). A randomized blocked algorithm for efficiently computing rank-revealing factorizations of matrices. *SIAM Journal on Scientific Computing*, 38(5), S485–S507.

Molina, J. & García, J. (2006). Técnicas de análisis de datos: aplicaciones prácticas utilizando microsoft excel y weka.

Morineau, A. & Aluja, T. (1999). Aprender de los datos: el análisis de componentes principales. *Editorial Universitaria de Barcelona, Barcelona,*.

Newman, M. & Ojalvo, I. (1970). Vibration modes of large structures by an automatic matrix-reductionmethod. *AIAA Journal*, 8(7), 1234–1239.

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Rayón, Á. (2016). Cuando los algoritmos se convierten en cajas negras. url https://blogs.deusto.es/bigdata/cuando-los-algoritmos-se-convierten-en-cajas-negras/. Accedido 26-05-2017.

Russolillo, G. et al. (2012). Non-metric partial least squares. *Electronic Journal of Statistics*, 6, 1641–1669.

Saad, Y. (1992). *Numerical methods for large eigenvalue problems*. Manchester University Press.

Sameh, A. H. (1971). On jacobi and jacobi-like algorithms for a parallel computer. *Mathematics of computation*, 25(115), 579–590.

Sayadi, T., Hamman, C., & Schmid, P. (2014). Parallel qr algorithm for data-driven decompositions. In *Center for Turbulence Research, Proceedings of the Summer Program* (pp. 335–343).

Silva Mata, F. J., Revilla Eng, A., Talavera Bustamante, I., Augier, Á., & Berretti, S. (2017). Reconocimiento biométrico de rostros mediante el análisis de datos funcionales de sus modelos tridimensionales. *Revista Cubana de Ciencias Informáticas*, 11(1), 1–14.

Sklar, M. (1959). Fonctions de repartition an dimensions et leurs marges. *Publ. inst. statist. univ. Paris*, 8, 229–231.

Spivak, M. (1996). *Cálculo infinitesimal*. Reverté.

Statsoft (2015). Principal component analysis and partial least squares technical notes. `http://documentation.statsoft.com/STATISTICAHelp.aspx?path=mspc/PCAandPLSTechnicalDetails`.

Sullivan, M. (1997). *Trigonometría y geometría analítica*. Pearson Educación.

Tanenbaum, A. S., Velasco, Ó. A. P., & Guerrero, G. (1996). *Sistemas operativos distribuidos*. Number QA76. 76063. T35. 3 1996. Prentice Hall México;.

Venkataraman, S., Yang, Z., Liu, D., Liang, E., Falaki, H., Meng, X., Xin, R., Ghodsi, A., Franklin, M., Stoica, I., et al. (2016). Sparkr: Scaling r programs with spark. In *Proceedings of the 2016 International Conference on Management of Data* (pp. 1099–1104).: ACM.

Villardón, J. (2002). Análisis de componentes principales. *Cataluña: UOC, Departamento de Estadística*, 32.

Wikipedia (2016). Cuda — wikipedia, la enciclopedia libre. `http://es.wikipedia.org/w/index.php?title=CUDA&oldid=94600454`.

Wold, S., Esbensen, K., & Geladi, P. (1987). Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3), 37–52.