Casa abierta al tiempo

UNIVERSIDAD AUTÓNOMA METROPOLITANA
UNIDAD IZTAPALAPA

CÁLCULO EFICIENTE DE PCA CON SVD Y LAPACK EN UN SISTEMA DE BASES DE DATOS RELACIONAL

TESIS QUE PRESENTA

LIC. EDGAR MARTÍNEZ ENCARNACIÓN

PARA OBTENER EL GRADO DE

MAESTRO EN CIENCIAS

(CIENCIAS Y TECNOLOGÍAS DE LA INFORMACIÓN)

Asesores:

Dr. René Mac Kinney Romero

Dr. Carlos Ordoñez Mondragón

Sinodales:

Presidente: Dr. Javier García García

Secretario: Dra. Reyna Carolina Medina Ramírez

Secretario: M. en C. Omar Lucio Cabrera Jiménez

Vocal:Dr. René Mac Kinney Romero

UNIVERSIDAD AUTÓNOMA METROPOLITANA

DIVISIÓN DE CIENCIAS BÁSICAS E INGENIERÍA

MÉXICO D.F. ENERO 2013

Resumen

La eficiencia y escalabilidad en la ejecución de métodos numéricos dentro de un Manejador de Bases de Datos (DBMS *Database Management System*) es una tarea compleja, ya que su arquitectura no esta diseñada para realizar cálculos numéricos intensos. Además el DBMS esta optimizado para evaluar consultas SQL (*Structured Query Language*) en tablas, no para realizar cálculos matriciales. Dada la dificultad de programar y optimizar diversos métodos numéricos, se propone integrar la biblioteca de métodos numéricos LAPACK (Linear Algebra Package) en un DBMS. La investigación se enfoca en resolver el Análisis de Componentes Principales (PCA *Principal Component Analysis*) para llevar a cabo la reducción de dimensionalidad (técnica de pre-procesamiento de datos) en grandes conjuntos de datos mediante su Descomposición en Valores Singulares (SVD *Singular Value Decomposition*), utilizando su respectiva matriz de correlación. Se comparan alternativas para resumir el conjunto de datos de entrada y cómo llamar de manera eficiente los métodos disponibles en la biblioteca LAPACK que resuelven SVD al aprovechar los mecanismos disponibles en un DBMS como: Procedimientos Almacenados (SP *Stored Procedures*), Funciones Definidas por el Usuario (UDFs *User Defined Functions*), Agregados Definidos por el Usuario (UDA *User Defined Aggregates*). Se muestra la factibilidad de resolver PCA al resumir el conjunto de datos con arreglos actualizados de manera incremental en una UDA, para después realizar SVD en memoria RAM llamando a la biblioteca LAPACK con una UDF. Además, comprobamos que mediante la explotación de una variante paralela de LAPACK (ScaLAPACK) es posible llevar a cabo un procesamiento paralelo en múltiples núcleos disponibles en la CPU. Como resultado se obtuvo un marco de trabajo independiente del DBMS que requiere una sola lectura del conjunto de datos, presenta una escalabilidad lineal, tiene posibilidad de ejecutarse en paralelo y funciona en cualquier DBMS siempre que soporte UDA.

Agradecimientos

Este trabajo no habría sido posible sin el apoyo y el estímulo del Dr. Carlos Ordoñez Mondragón, bajo cuya supervisión escogí este tema. Gracias a su apoyo, ayuda y consejos considero que maduré bastante tanto en mi vida personal como académica. Además he tenido la oportunidad de vivir gratas experiencias que nunca hubiera podido tener en México.

Agradezco al Dr. René Mac Kinney Romero por su apoyo, paciencia y orientación en cada uno de los aspectos del presente trabajo. Al equipo de Investigación DBMS en la Universidad de Houston: Dr. Carlos García Alvarado, M. en C. Naveen Mohanam por su enorme ayuda, consejos y amistad. A los profesores Dr. Javier García García, Dra. Carolina Medina Ramírez y M. en C. Omar Lucio Cabrera Jiménez por sus ideas y comentarios aportados para el mejoramiento de la tesis.

Deseo agradecer al Consejo Nacional de Ciencia y Tecnología CONACYT, a la Universidad Autónoma Metropolitana, y al Instituto de Ciencia y Tecnología del Distrito Federal ICyTDF por el apoyo económico, por las oportunidades brindadas para la realización de la maestría y la invaluable ayuda que siempre tuvieron para conmigo.

No puedo terminar sin agradecer a mi familia, y a todas las personas que he conocido a lo largo de mi vida, he aprendido mucho de todos y cada uno de ellos.

Contenido

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

1.1. Síntesis

De acuerdo a Demmel, Golub y Van Loan la mayoría de los modelos matemáticos, estadísticos y de Minería de Datos (*DM Data Mining*) están basados en cálculos de álgebra lineal [25, 42]. Programar estos métodos numéricos dentro de un Manejador de Bases de Datos (*DBMS Data Base Management System*) no es una tarea sencilla. Hay varias razones fundamentales: SQL (*SQL Structured Query Language*) es un lenguaje lento y rígido para cálculos numéricos intensos, el DBMS está optimizado para evaluar consultas SQL en tablas, no para realizar cálculos matriciales, la exportación de tablas de gran tamaño fuera de un DBMS es y seguirá siendo un cuello de botella (suponiendo la existencia de una base de datos analítica centralizada). A pesar de que las consultas SQL pueden ser optimizadas para modelos matemáticos con matrices diagonales [76, 80] o modelos simples como los árboles de decisión [85]. La optimización de consultas es una tarea más compleja para matrices densas no diagonales, como los dados como entrada a la Descomposición en Valores Singulares (*SVD Singular Value Decomposition*). A un nivel fundamental, el álgebra relacional [38, 57] y las matrices [25, 42] son abstracciones matemáticas diferentes, que requieren distintos lenguajes de programación para realizar sus cálculos.

1.2. Motivación

En general, es benéfico utilizar bibliotecas de métodos numéricos como el Paquete de Álgebra Lineal (*LAPACK Linear Algebra Package*) o los Subprogramas Básicos de Álgebra Lineal (*BLAS Basic Linear Algebra Subprograms*), de esta manera se evita modificar el código fuente interno del DBMS, optimizar e incluso desarrollar diversos métodos numéricos existentes como SVD. El modificar la arquitectura interna del DBMS y sus subsistemas para que soporte cálculos numéricos es una tarea compleja, que puede implicar un tiempo considerable de desarrollo. Por otro lado, el uso de arreglos no esta disponible en SQL [9, 28], sin embargo, si lo está en forma de código fuente dentro de Funciones Definidas por el Usuario (*UDFs User Defined Functions*). Estos arreglos se almacenan y manipulan directamente en memoria RAM, utilizando lenguajes de programación tradicionales como C++ y C# [30, 38, 86]. Sobre la base de esta motivación, se estudia cómo integrar estas bibliotecas matemáticas optimizadas en el DBMS para realizar el Análisis de Componentes Principales (*PCA Principal Component Analysis*) aprovechando los mecanismos ofrecidos en un DBMS como Procedimientos Almacenados (SP Stored Procedures *Stored Procedures*), UDFs, Agregados Definidos por el Usuario (UDA User-defined Aggregates *User Defined Aggregates*), en particular en el DBMS MS SQL Server. En este trabajo se muestra un algoritmo eficiente y escalable basado en obtener las matrices de sumarización del conjunto de datos $\mathbf{X}$ utilizando UDA [64, 90] y la resolución de SVD de su matriz de correlación llamando a la librería LAPACK a través de UDFs, además de mostrar optimizaciones orientadas a Bases de Datos para resolver PCA con grandes volúmenes de datos en un DBMS. El algoritmo presentado puede resolver modelos de PCA un orden de magnitud mayor y más rápido utilizando consultas SQL [68, 73, 78, 84].

1.3. Minería de Datos

La generación de datos, resultado de actividades empresariales, gubernamentales, de investigación, entre otras, ha provocado un incremento en el interés por identificar información útil y representativa dentro de grandes conjuntos de datos (Véase Fig. 1.1). Actualmente existe un crecimiento en la capacidad de generar y almacenar datos, debido al poder de procesamiento de las computadoras así como al bajo costo de almacenamiento. Sin embargo, dentro de estos volúmenes de datos existe una gran cantidad de información

oculta, que puede ser de gran importancia estratégica a la que no se puede acceder por técnicas clásicas de recuperación de la información [7, 11] DM es el proceso que da como resultado el descubrimiento de nuevos patrones anteriormente desconocidos. Los patrones pueden ser vistos como una especie de resumen de los datos de entrada, y pueden ser utilizados en otros análisis. El proceso de DM involucra el análisis automático o semiautomático en grandes conjuntos de datos [31]. El desarrollo de DM tiene la necesidad de convertir esta gran cantidad de datos en información útil. Por esta razón este campo ha evolucionado junto con el almacenamiento de base de datos. Sin embargo, DM se ha ido separando de las transacciones y operaciones del DBMS [33].

Figura 1.1: Generación de datos en diferentes rubros.

Aunque el propósito inicial de las Bases de Datos fue el procesamiento de transacciones, los datos recogidos son inútiles si la información no puede ser extraída de ellas. Por lo tanto, existe una demanda de herramientas para el análisis de estos datos y sistemas que no interfieran con la tarea de recolección de datos. Los datos pueden ser resumidos y almacenados para ser reutilizados en diferentes análisis. Por otro lado existe un conjunto

de varias técnicas que trabajan sobre la arquitectura de los sistemas de Minería de Datos. Estas técnicas interactúan con una Base de Datos o un Almacén de Datos (*DWH Data Warehouse*) para extraer conocimiento a partir de los datos almacenados [48].

El Descubrimiento de Conocimiento en Bases de Datos (*KDD Knowledge Discovery Data in Databases*) da un significado a patrones encontrados en el proceso de Minería de Datos. Así el valor real de los datos reside en la información que se puede extraer de ellos, esta información ayudará a mejorar la comprensión de los fenómenos que nos rodean [36, 41]. KDD realiza las tareas de preparación de los datos, interpretación de los resultados obtenidos, extracción de tendencias de interés y eventos, creación de informes útiles, el apoyo de toma de decisiones y el aprovechamiento de información obtenida para representar el conocimiento descubierto mediante reglas (Véase Fig. 1.2). Es necesario implementar técnicas que permitan convertir este gran volumen de datos en conocimiento, para revelar información valiosa para la toma de decisiones, estrategias empresariales y de investigación [60]. Se considera que los sistemas de DM reales son aquellas herramientas capaces de realizar análisis de grandes cantidades de datos. El Aprendizaje Maquinal (*ML Machine Learning*) y las herramientas estadísticas pueden ser consideradas versiones a escala de una herramienta de Minería de Datos, ya que no toman en consideración limitaciones de memoria y el tamaño de los datos [32, 33, 36].

Existe una fuerte motivación para diseñar e implementar sistemas con pequeños conjuntos de características. Sin dejar de lado, que al mismo tiempo, hay una opuesta necesidad de incluir un conjunto suficiente de características para lograr un alto rendimiento. Esto ha motivado el desarrollo de una variedad de técnicas que ofrecen encontrar el subconjunto óptimo a partir de un conjunto inicial de características [33]. Sin embargo, muchas de esas técnicas pueden manipular solamente ciertos tipos de datos. Usualmente, los sistemas automatizados de identificación evalúan grupos conformados por diferentes características o atributos extraídos del fenómeno de estudio, tanto el número de mediciones, así como cuando el número de variables en cada medición, es decir, la dimensionalidad de los datos es muy grande [88]. Como cada característica o atributo del conjunto de datos que se incluye en el análisis, puede incrementar el costo y el tiempo de proceso de los sistemas. En la mayoría de estos procesos se contemplan conjuntos de características amplios, que conllevan al empleo de grandes recursos computacionales, tanto en la etapa de caracterización como en las posteriores de almacenamiento, transmisión y procesamiento de los

datos [5]. Además, un alto número de características puede generar sobre-entrenamiento en la etapa de clasificación. Por tales razones, es aconsejable reducir la dimensión de los datos. Así, debe mantenerse la dimensión del espacio de características tan pequeña como sea posible, en consideración a la precisión en la clasificación automática. Es así como un número limitado de características simplifica la representación tanto del patrón, como de los parámetros de clasificación, resultando en una extracción y análisis menos denso y extenuante, permitiendo tener un clasificador más rápido que utiliza menos memoria. Sin embargo, una reducción excesiva en el número de características podría llevar a una pérdida de información, empobreciendo la precisión del sistema de reconocimiento o clasificación.

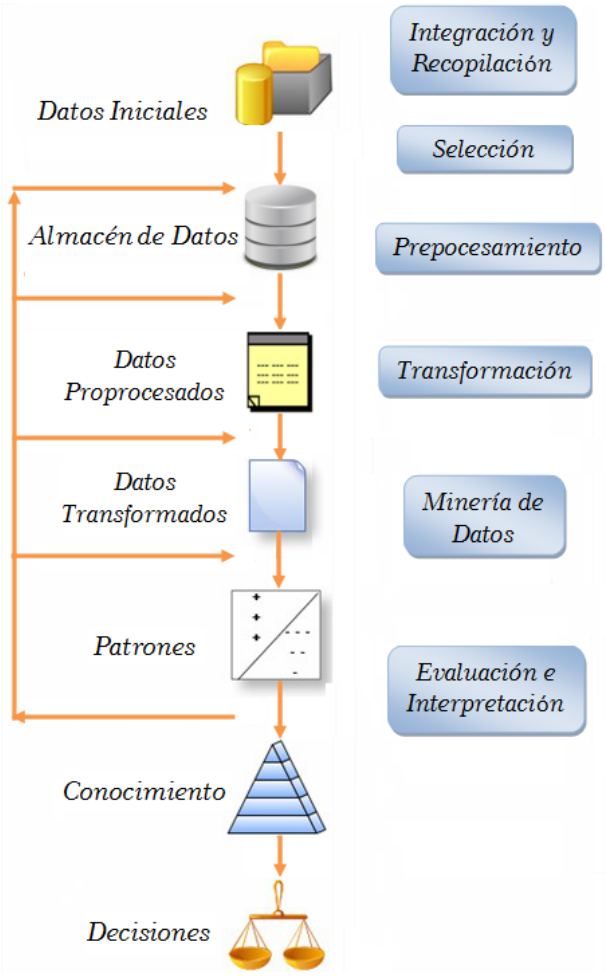

Figura 1.2: Fases del proceso de Descubrimiento de Conocimiento en Bases de Datos.

La Reducción de Dimensionalidad (*DR Dimensionality Reduction*) es utilizada frecuentemente como una etapa de preprocesamiento en el entrenamiento de sistemas, consiste en escoger un subconjunto de variables, de tal manera, que el espacio de características o atributos quede reducido de acuerdo a un criterio, y el subconjunto represente mejor el espacio inicial de entrenamiento, es decir, la reducción de dimensionalidad permite elegir las variables más influyentes que describen con un número mucho menor de atributos al conjunto original de datos [88].

1.4. Visión General

La presente tesis esta organizada de la siguiente manera. En los capítulos 2 y 3 se presentan definiciones y se introducen las capacidades analíticas de un DBMS. El capítulo 4 presenta una visión general de como resolver PCA en SQL. También introduce un algoritmo paralelo que resume el conjunto de datos, obtiene la matriz de correlación y explora diversas alternativas para integrar LAPACK. El capítulo 5 presenta una evaluación experimental, haciendo hincapié en la escalabilidad de conjuntos de datos de alta dimensionalidad. Finalmente el capítulo 6 resume los hallazgos y los temas de investigación para el trabajo futuro.

Capítulo 2

Antecedentes y Definiciones

A continuación se explica el modelo matemático PCA, la factorización SVD, así como el cálculo de la matriz de correlación y las estadísticas suficientes necesarias para su cálculo [47, 49]. Se discute como calcular SVD en una matriz de correlación, la cual tiene una relación teórica con PCA que es el modelo estadístico fundamental estudiado. También se describe un DBMS, así como las UDFs, principalmente las UDA y su importancia como mecanismos que permiten extender la funcionalidad en un DBMS. Por último, se presentan conceptos básicos de la biblioteca de métodos numéricos LAPACK.

2.1. Análisis de Componentes Principales

PCA es un modelo matemático utilizado en el desarrollo de varias aplicaciones para el Procesamiento de Imágenes [40], Compresión de Datos [19], Reconocimiento de Patrones [98], Clustering [27], Clasificación [51], y Predicciones del estado del tiempo [96]. PCA realiza una transformación lineal para convertir un conjunto de observaciones de variables posiblemente correlacionadas en un conjunto de valores de variables linealmente independientes entre si (i.e. variables no correlacionadas) llamadas componentes principales, ordenadas de acuerdo a su varianza. Esta transformación está definida de tal manera que el primer componente principal representa la mayor variabilidad en los datos. Cada componente subsiguiente representa al resto de la variabilidad del conjunto de datos, es decir, los componentes principales son independientes. Dado que el número de componentes principales es menor o igual al número de variables originales, se eligen los k vectores más

representativos [55, 67]. Sin embargo, como el interés estriba en reducir la dimensionalidad del mismo, se trata de explicar un porcentaje elevado de la variación temporal del vector utilizando un número reducido de componentes principales (Véase Fig. 2.1).

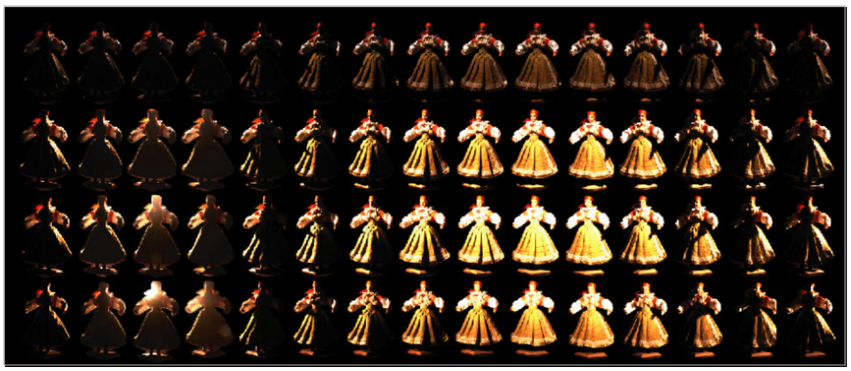

Figura 2.1: Ejemplo PCA: Imágenes con distinta iluminación.

Así, PCA encuentra un nuevo conjunto de dimensiones ortogonales representado por la combinación lineal de las dimensiones del conjunto de datos de entrada (Véase Fig. 2.2).

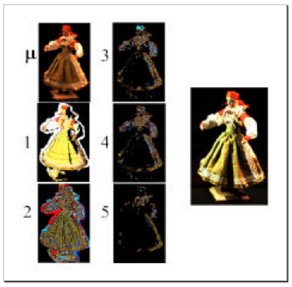

Figura 2.2: Ejemplo PCA: Mayor varianza de las imágenes captada por cinco componentes principales.

PCA proyecta el conjunto de datos en un nuevo espacio dimensional menor, preservando y exhibiendo relaciones entre las dimensiones, así como la variabilidad en el conjunto de datos original. Se garantiza que los componentes principales son independientes solo si el conjunto de datos tiene una distribución Gaussiana multivariante [40].

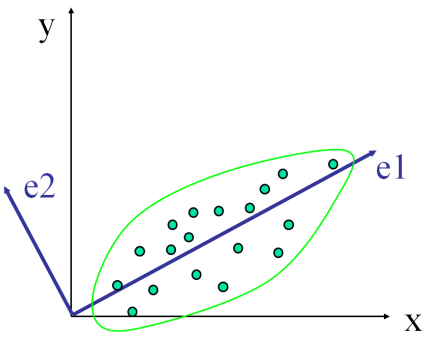

Figura 2.3: Proyección de datos en $\mathbb{R}^2$ a $\mathbb{R}^1$.

El modelo PCA, también conocido como método de Karhunen-Loeve o descomposición ortogonal apropiada [34], extrae información relevante de los datos altamente correlacionados. Proporciona un espacio de dimensión inferior que facilita el análisis de los datos (Véase Figuras 2.3,2.4). De acuerdo a Boutsidis, Mahoney y Drineas, en comparación con otras técnicas de reducción de dimensionalidad, PCA ha demostrado ser eficiente cuando se trabaja con datos escasos y de alta dimensionalidad [14, 68].

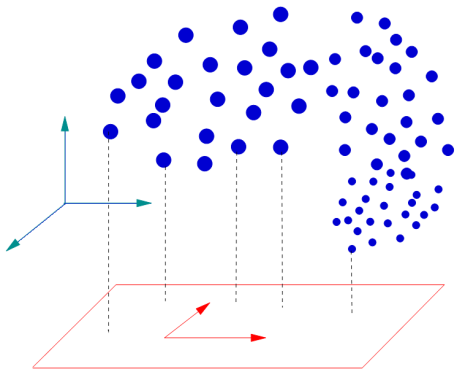

Figura 2.4: Proyección de datos en $\mathbb{R}^3$ a $\mathbb{R}^2$.

Dado que la varianza depende de la escala de las variables, comúnmente se normaliza primero cada variable para tener una media de cero

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i = 0 \tag{2.1}$$

y desviación estándar de uno

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2} = 1 \tag{2.2}$$

Después de la normalización las variables originales, posiblemente con diferentes unidades de medida, están en unidades comparables.

2.2. Estadísticas Suficientes de la Gaussiana Multivariada

Las Estadísticas Suficientes de la Gaussiana Multivariada de un conjunto de datos (n,L y Q) son variables aleatorias y pueden ser multidimensionales y no dependen de parámetros [10, 35, 95]. Dada una familia $\mathbf{F}$ de distribuciones completamente especificada, excepto por un numero limitado de parámetros desconocidos, y un conjunto de datos queremos saber:

1. Si los datos son compatibles con la familia de distribuciones $\mathbf{F}$ (Selección del modelo).

2. Asumiendo que los datos son compatibles, es de interés conocer que puede concluirse con respecto a los parámetros desconocidos (Estimación y test de hipótesis).

El objetivo de las estadísticas suficientes es condensar información ya que proporcionan tanta información acerca del parámetro como la propia muestra o conjunto de datos. Sea $\mathbf{X} = \{x_1, ..., x_n\}$ una muestra aleatoria de

$$f_x(x, \theta), \theta \in \Theta \subseteq \Re \tag{2.3}$$

$$T_i(x), i = 1, \ldots, n \tag{2.4}$$

las T_i son conjuntamente suficientes para

$$\theta \; si \; f_{x|T=t}(x, t, \theta) = h(x) \tag{2.5}$$

no depende de θ.

Para una distribución normal univariante los únicos parámetros son la media μ y la varianza σ (o desviación estándar), por lo que esta información determina por completo la distribución. La suma de los puntos x dividida por el número de puntos da la media, por lo que la suma de los puntos x contiene suficiente información para obtener la media. La suma de los valores al cuadrado x^2, de manera similar puede ser utilizada para encontrar la varianza (junto con la media o la suma de los puntos x y el tamaño del conjunto de datos o muestra), es decir, la suma de los puntos x y la suma de los puntos al cuadrado x^2 son estadísticas suficientes.

Para una distribución normal multivariante, los parámetros son un vector de medias y una matriz de varianzas y covarianzas. De la misma manera que en el caso de una distribución normal univariante, el vector de la suma de las columnas, dividido por el tamaño del conjunto de datos da el vector de medias y una combinación cuadrática (o producto cruz) de los puntos con la suma de las columnas y tamaño del conjunto de datos dará la matriz de varianza/covarianza [37]. Suponiendo que los datos provienen de una distribución normal multivariante, entonces las estadísticas suficientes darán toda la información requerida, esto trae ciertos beneficios, ya que una manera de lidiar con grandes conjuntos de datos es sólo actualizar las estadísticas suficientes sin tener que mantener más de un par de filas de la datos de la memoria en un momento [61].

Existen dos matrices fundamentales para todos los modelos estadísticos, la suma lineal de los puntos L y la suma cuadrática del producto cruz de los puntos Q [74].

Sea L un vector columna $d \times 1$ con la suma lineal de los puntos de la forma L_i, donde $i = 1, 2, ..., d$.

$$L = \sum_{i=1}^{n} x_i = \begin{bmatrix} \Sigma x_1 \\ \Sigma x_2 \\ \vdots \\ \Sigma x_d \end{bmatrix} \tag{2.6}$$

Sea Q una matriz $d \times d$ con la suma cuadrática del producto cruz de los puntos de la forma Q_{ij}

$$Q = \mathbf{X}\mathbf{X}^\top = \sum_{i=1}^{n} x_i x_i^\top \tag{2.7}$$

$$Q = \begin{bmatrix} \Sigma x_1^2 & \Sigma x_1 x_2 & \cdots & \Sigma x_1 x_d \\ \Sigma x_2 x_1 & \Sigma x_2^2 & \cdots & \Sigma x_2 x_d \\ \vdots & \vdots & \ddots & \vdots \\ \Sigma x_d x_1 & \Sigma x_d x_2 & \cdots & \Sigma x_d^2 \end{bmatrix} \tag{2.8}$$

La matriz Q tiene sumas de cuadrados en la diagonal y sumas de productos cruz fuera de la diagonal. La principal propiedad de L y Q es que son mucho menores que $\mathbf{X}$ cuando $d \ll n$. Además resumen varias características esenciales de $\mathbf{X}$ que pueden ser explotadas por técnicas estadísticas, al sustituir L por cada suma $\sum x_i$ y cada producto $\mathbf{X}\mathbf{X}^\top$ por Q. Al conocer n (el número de ejemplos, instancias o filas), es posible conocer las estadísticas suficientes de la Gaussiana multivariada (NLQ o *Summarization matrices*) [81].

$$n = \sum_{i=1}^{n} 1 \tag{2.9}$$

2.3. La matriz de correlación

La matriz de correlación ρ es una matriz cuadrada y simétrica de tamaño $d \times d$ cuyos valores son la medición de la dependencia lineal entre d dimensiones de los datos originales. Estos coeficientes indican la dependencia y dirección de la relación lineal entre dos variables en el intervalo $[-1, 1]$. Esta matriz tiene unos en la diagonal y fuera de ella los coeficiente de correlación entre las variables (Véase Fig. 2.5).

$$\rho = \begin{bmatrix} 1 & \rho_{12} & \cdots & \rho_{1d} \\ \rho_{21} & 1 & \cdots & \rho_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{d1} & \rho_{d2} & \cdots & 1 \end{bmatrix}$$

Figura 2.5: Matriz de correlación ρ.

Así, el coeficiente de correlación entre las dimensiones a y b del conjunto de datos $\mathbf{X}$ está dada por la ecuación (2.10).

$$\rho_{ab} = \frac{n\Sigma x_{ai} x_{bi} - \Sigma x_{ai} \Sigma x_{bi}}{\sqrt{n\Sigma (x_{ai})^2 - (\Sigma x_{ai})^2}\sqrt{n\Sigma (x_{bi})^2 - (\Sigma x_{bi})^2}} \tag{2.10}$$

La ecuación (2.10) puede ser reescrita en términos de las matrices de sumarización: n, L y Q descritas en la sección 2.2. De esta manera la matriz de correlación ρ puede ser calculada en una sola lectura del conjunto de datos [75].

$$\rho_{ab} = \frac{nQ_{ab} - L_a L_b}{\sqrt{nQ_{aa} - L_a^2}\sqrt{nQ_{bb} - L_b^2}} \tag{2.11}$$

A continuación se detalla el cálculo de la factorización SVD y la relación con la matriz de correlación para resolver PCA.

2.4. Descomposición en Valores Singulares

SVD es una técnica importante utilizada para la factorización de una matriz rectangular real o una matriz compleja, para calcular la pseudoinversa de una matriz, resolver ecuaciones lineales homogéneas y mínimos cuadrados. También tiene aplicación en tareas relacionadas con PCA, Reconocimiento de Patrones y Procesamiento de imágenes para el análisis espectral de valor singular. SVD también tiene una variedad de aplicaciones en computación científica, procesamiento de señales, control automático y muchas otras áreas [58, 89]. La resolución de problemas PCA es equivalente a resolver SVD en la matriz de covarianza o correlación. Sea $X = \{x_1, ..., x_n\}$ el conjunto de datos de entrada con n puntos, donde cada punto tiene d dimensiones. $\mathbf{X}$ es una matriz $d \times n$, donde cada punto x_i es representado con un vector columna (equivalente a una matriz $d \times 1$). Utilizamos los subíndices i y j para indicar la posición del valor en una matriz, de tal forma que x_{ij} es el valor en la fila i y en la columna j de $\mathbf{X}$. La transpuesta de una matriz se denota $\top$, de manera similar la norma de un vector se denota como $||x_i||$. Dada una matriz $\mathbf{X} \in \Re^{m \times n}$ la factorización SVD de la matriz $\mathbf{X}$ satisface:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top \tag{2.12}$$

donde $\mathbf{U} \in \Re^{m \times m}$ es unitaria,$\mathbf{V} \in \Re^{n \times n}$ es unitaria y $\mathbf{\Sigma} \in \Re^{m \times n}$ es diagonal, las columnas $u_i, \cdots, u_n$ de $\mathbf{U}$ se denominan vectores singulares izquierdos. Las columnas $v_i, \cdots, v_n$ de $\mathbf{V}$ se denominan vectores singulares derechos. Los elementos σ_i se denominan valores singulares, estos son no negativos y se presentan en orden decreciente $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p$, donde p = min(m,n) (Véase Figs. 2.6 y 2.7).

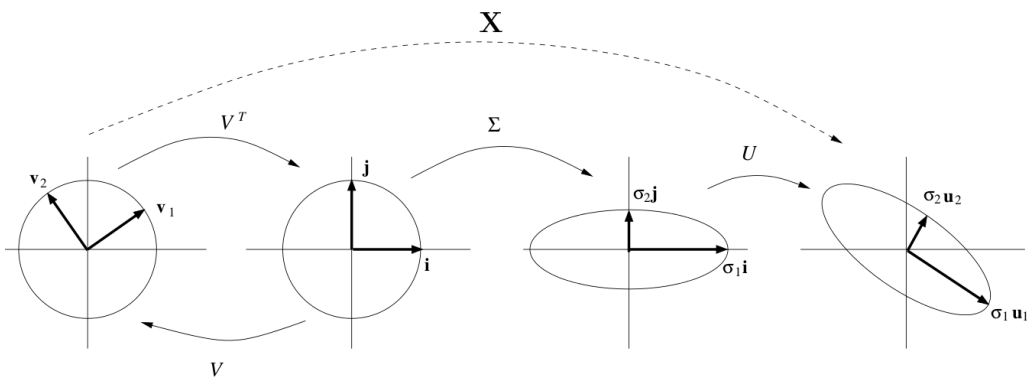

Figura 2.6: Representación Geométrica de SVD.

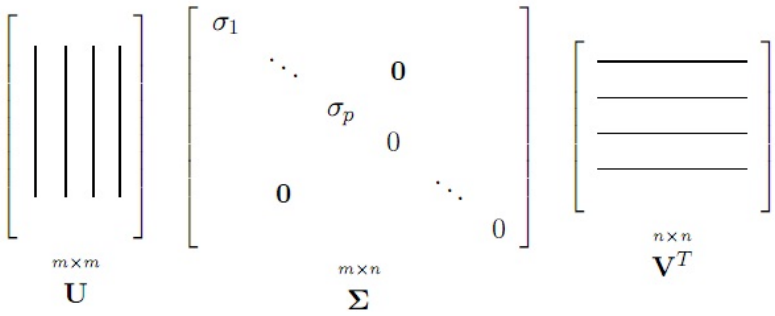

Figura 2.7: Notación matricial de $\mathbf{U}$, $\mathbf{\Sigma}$ y $\mathbf{V}^\top$.

Sea la matriz $\mathbf{X} \in \Re^{m \times n}$ una matriz simétrica. Al aplicar una transposición a la ecuación (2.12), se obtiene la siguiente ecuación:

$$\mathbf{X}^\top = (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top)^\top = (\mathbf{V}\mathbf{\Sigma}\mathbf{U}^\top) \tag{2.13}$$

Multiplicando las ecuaciones (2.12) y (2.13), y dado que $\mathbf{X} = \mathbf{X}^\top$, es decir, $\mathbf{X}\mathbf{X}^\top = \mathbf{X}^\top\mathbf{X} = \mathbf{X}^2$ obtenemos:

$$\mathbf{X}^2 = \mathbf{X}\mathbf{X}^\top = (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top)(\mathbf{V}\mathbf{\Sigma}\mathbf{U}^\top) \tag{2.14}$$

Dado que $\mathbf{V}\mathbf{V}^\top = \mathbf{I}$

$$\mathbf{X}\mathbf{X}^\top = \mathbf{U}\mathbf{\Sigma}^2\mathbf{U}^\top \tag{2.15}$$

De manera similar

$$\mathbf{X}^2 = \mathbf{X}^\top\mathbf{X} = (\mathbf{V}\mathbf{\Sigma}\mathbf{U}^\top)(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top) \tag{2.16}$$

$$\mathbf{X}^\top \mathbf{X} = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^\top \tag{2.17}$$

Tanto como $\mathbf{U}$ y $\mathbf{V}$ contienen los vectores propios que diagonalizan ortogonalmente $\mathbf{X}^2$

$$\sigma(\mathbf{X}^2) = \sigma_1^2, \sigma_2^2, \cdots, \sigma_n^2 \tag{2.18}$$

$$\mathbf{X}^2 v_i = \sigma_i^2 v_i;\ \mathbf{X}^2 u_i = \sigma_i^2 u_i \tag{2.19}$$

Sea la matriz simétrica $\rho \in \Re^{d\times d}$, donde $\rho = \rho^\top$, y sea $\rho = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ una SVD de ρ, con $\sigma_1 > \sigma_2 > \cdots > \sigma_d$, los valores singulares distintos y m_i, $i = 1, \cdots, p$ sus respectivas multiplicidades, de modo que $\Sigma = diag[\sigma_1 I_{m1}, \cdots, \sigma_p I_{mp}]$ y

$$\mathbf{U} = [\mathbf{U}_1|\mathbf{U}_2|\mathbf{U}_3|\cdots|\mathbf{U}_p] \tag{2.20}$$

$$\mathbf{V} = [\mathbf{V}_1|\mathbf{V}_2|\mathbf{V}_3|\cdots|\mathbf{V}_p] \tag{2.21}$$

donde $\mathbf{U}_i$,$\mathbf{V}_i \in \Re^{n\times m_i}$ son las columnas que contienen los vectores propios (*eigenvectors*) que corresponden a los valores propios (*eigenvalues*) σ_i. Dado que ρ_{ab} es equivalente a $\mathbf{X}\mathbf{X}^\top$ cuando $\mathbf{X}$ es estandarizada con una puntuación estándar (*z-score*) que es una medida estadística que cuantifica la distancia en un punto de la media en términos de desviaciones estándar. La ecuación (2.12) muestra la factorización SVD de PCA donde $\mathbf{U}$ y $\mathbf{V}^\top$ son los vectores propios de la matriz $d \times d$. Σ^2 es una matriz diagonal de dimensión d que contiene los valores propios. Utilizando la matriz de correlación para $\mathbf{X}\mathbf{X}^\top$, el problema radica en encontrar un conjunto de vectores propios $\mathbf{U}$ y un conjunto de valores propios en la diagonal de la matriz Σ^2, como se observa en las ecuaciones (2.15) y (2.17) [83]. En esta tesis se utiliza la factorización SVD de la matriz de correlación simétrica ρ para resolver PCA que busca la proyección según la cual los datos queden mejor representados en términos de mínimos cuadrados.

2.5. Relación de PCA y SVD

Existe una relación directa entre PCA y SVD en el caso donde los componentes principales son calculados a partir de la matriz de covarianza. Como se mencionó en la sección 2.4. Para una matriz $\mathbf{X}$ la factorización SVD satisface $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$. Donde $\mathbf{U}$ contiene los vectores propios de $\mathbf{X}\mathbf{X}^\top$ y $\mathbf{V}^\top$ contiene los vectores propios de $\mathbf{X}^\top\mathbf{X}$. Al resumir

el conjunto de datos $\mathbf{X}$ con las estadísticas suficientes y obtener la matriz de correlación ρ, el problema de calcular SVD se reduce a realizar la Descomposición Ortogonal Propia (*Eigen Decomposition*) sobre ρ:

$$\mathbf{X}\mathbf{X}^\top = \rho = \mathbf{G}\Lambda\mathbf{G}^\top \tag{2.22}$$

$$\mathbf{G}\Lambda\mathbf{G}^\top = \sum_{i=1}^{n} g_i \lambda_i g_i^\top = \sum_{i=1}^{n} g_i \left|\lambda_i\right| \operatorname{sign}(\lambda_i) g_i^\top \tag{2.23}$$

donde g_i son las columnas de la matriz $\mathbf{G}$, los vectores singulares izquierdos u_i son g_i y los vectores singulares derechos v_i son $sign(\lambda)g_i$ y cada elemento de $\mathbf{\Lambda}$: $\lambda_i = \sigma_i^2$. Entonces al centrar cada columna de la matriz de datos de entrada $\mathbf{X}$. $\mathbf{X}\mathbf{X}^\top = \Sigma_i g_i g_i^T$ es proporcional a la matriz de covarianza de las variables de g_i. La diagonalización de $\mathbf{X}\mathbf{X}^\top$ produce $\mathbf{V}^\top$, la cual produce los componentes principales de $\{g_i\}$. Así, los vectores singulares derechos $\{v_k\}$ son los mismos que los componentes principales de $\{g_i\}$. Los valores propios de $\mathbf{X}\mathbf{X}^\top$ son equivalentes a σ_k^2, que son proporcionales a las variaciones de los componentes principales [24, 45]. En general, los componentes principales son aquellos cuyos valores propios $\lambda_i \geq 1$ o aquellos que retienen la mayor parte de la varianza (típicamente el 90 %) [17, 40].

2.5.1. Reducción de Dimensionalidad

Como se mencionó anteriormente PCA realiza un mapeo lineal de los datos a un espacio dimensional inferior de tal manera que la varianza de los datos en esta representación se maximiza. En la práctica, la matriz de correlación se construye para después calcular los vectores propios de esta matriz. Los vectores propios que corresponden a los valores propios más grandes, se utilizan para reconstruir una gran parte de la varianza de los datos originales. El espacio original ha sido reducido al espacio generado por menos vectores propios.

A continuación se explica a detalle la definición de un DBMS y posteriormente la representación de modelos de DM en Bases de Datos.

2.6. Manejador de Bases de Datos

Los DBMS son un tipo de software específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan como se muestra en la Fig. 2.8. Un DBMS es una colección de numerosas rutinas de software interrelacionadas, cada una de las cuales es responsable de una tarea específica. El objetivo primordial de un DBMS es proporcionar un entorno que sea a la vez conveniente y eficiente para ser utilizado al extraer, almacenar y manipular información de la base de datos. Todas las peticiones de acceso a la base, se manejan de manera centralizada por medio del DBMS [8].

Figura 2.8: Esquema de un DBMS.

2.6.1. Base de Datos

Una Base de Datos es una colección de archivos interrelacionados, creados por un DBMS. El contenido de una base de datos incluye datos (almacenados en archivos) de una organización de tal manera que los datos estén disponibles para los usuarios. El objetivo del uso de la base de datos es eliminar inconsistencia en los datos. Los componentes principales de un sistema de base de datos son el Hardware, el Sistema DBMS y los datos a manejar, así como el personal encargado del manejo del sistema. Una Base de Datos Relacional es una base de datos en donde todos los datos visibles al usuario están organizados estrictamente como tablas de valores, y en donde todas las operaciones de la base de datos actúan sobre estas tablas, es decir, es una colección de relaciones normalizadas de diversos grados que varían con el tiempo.

2.6.2. Almacén de Datos

Un Almacén de Datos es una colección de datos que recoge información de múltiples sistemas fuentes u operacionales dispersos, y cuya actividad se centra en la toma de decisiones, es decir, un Almacén de Datos es una colección de datos que está formada por hechos (n filas) y dimensiones (d columnas). Las dimensiones son los elementos para ubicar datos que participan en el análisis y los hechos son los valores que se desean analizar [65].

2.6.3. Lenguaje de Consulta Esctructurado

SQL es el lenguaje estándar ANSI/ISO de definición, manipulación y control de Bases de Datos relacionales[53, 56]. Es un lenguaje declarativo; sólo hay que indicar qué se quiere hacer. En cambio, en los lenguajes procedimentales es necesario especificar cómo hay que hacer cualquier acción sobre la Base de Datos. El SQL es un lenguaje muy parecido al lenguaje natural; concretamente, se parece al inglés, y es muy expresivo. Por estas razones, y como lenguaje estándar, el SQL es un lenguaje con el que se puede acceder a todos los sistemas relacionales comerciales. Un programa de SQL reafirma relaciones lógicas preexistentes entre los elementos de datos como objetos de datos tabulares, a continuación, especifica subconjuntos y como combinar objetos de datos tabulares para producir un resultado. El álgebra relacional básica, lógica de conjuntos, y la lógica Booleana tienen un papel importante en los programas de SQL. La estructura de una consulta SQL fomenta la reutilización de las mismas estructuras básicas para construir una nueva estructura lógica que incluye una solución en la forma de un objeto de datos tabulares. SQL opera en estructuras de datos por columnas y filas, es decir SQL lee las filas de una tabla, una por una y decide si mantener todos, una parte o ninguno de los datos en cada columna. Esto puede parecer restrictivo, pero las estructuras de datos tabulares pueden representar todos los datos, y un procesamiento elemento por elemento, aunque tedioso, puede replicar los resultados de varios programas. Los Sistemas de Programación Procedimentales (*PPS Procedural Programming Systems*) colocan los datos en una secuencia física, frecuentemente ordenando por variables clave (*key variables*), para preparar un archivo para un procesamiento eficiente, en contraste, SQL utiliza ordenamientos lógicos de los datos en una o más columnas (también llamadas *llaves*) para localizar y agregar datos en otras

columnas y en las otras tablas. Las llaves primarias tienen distintos valores en las filas, esta característica permite identificar instancias de una entidad representadas por una tabla [1].

Las consultas son las operaciones más comunes en SQL. Se utilizan para recuperar datos de tablas con un criterio específico. Los DBMSs Data Base Management Systems usualmente incluyen optimizadores de consultas con el objetivo de encontrar un resultado, reduciendo al mínimo las operaciones físicas en disco. La operación `SELECT` se utiliza para definir una proyección de los atributos o dimensiones de los registros que satisfacen una condición dada. Para definir más operaciones del cálculo relacional en una declaración, SQL tiene las siguientes cláusulas:

- **FROM:** Define las tablas a consultar, así como combinaciones, INNER, OUTTER y NATURAL JOIN del álgebra relacional.
- **WHERE:** Define la operación de selección del álgebra relacional. Aquí se especifica la condición de selección.
- **GROUP BY:** Especifica los atributos por los cuales se debe agrupar el resultado al momento de calcular una función de agregación.
- **HAVING:** Define una condición que involucra grupos.
- **ORDER BY:** Ordena los datos resultantes por ciertos atributos.

Otras operaciones incluidas en SQL son inserciones, actualizaciones y borrado. Las instrucciones `INSERT` se utilizan para agregar registros a las tablas existentes. Esta es la operación menos costosa, ya que las estructuras de datos, indexación y almacenamiento se implementan para trabajar de manera eficiente. Por otro lado, las operaciones `DELETE` puede ser definidas con una restricción para los datos que se borren. La eliminación es una tarea costosa, ya que requiere la actualización de las estructuras de datos, índices y tablas de dispersión. Esto puede ser resuelto cuando se realizan una operación de truncamiento `TRUNCATE`, eliminando todos los elementos en una tabla específica, evitando la reconstrucción de las estructuras de datos. Por último, tenemos que la operación `UPDATE`. Esta operación involucra ubicar el registro en el disco, cambiar su valor, y la actualización de índices y tablas de dispersión.

2.7. Representación de modelos de Minería de Datos en Bases de Datos

Los avances en la investigación en minería de datos han hecho posible implementar diversas operaciones de minería de datos de manera eficiente en grandes Bases de Datos. Los desarrolladores de aplicaciones de Bases de Datos son capaces de construir y utilizar modelos de minería de datos para tareas de predicción y de análisis, además de compartir los modelos con otras aplicaciones. En los últimos años se han desarrollado muchas técnicas de minería de datos para grandes conjuntos de datos, los algoritmos consideran que los datos están almacenados en disco y están conscientes de una jerarquía de memoria (no suponen que todos los datos deban residir en la memoria) [2].

Existen varias razones para la aplicación de técnicas de minería de datos en SQL.

1. SQL es un lenguaje *declarativo* (no imperativo) orientado a conjuntos de datos, en el que lo importante es definir qué se desea hacer, y no cómo hacerlo, y no hay necesidad de manejar los datos a bajo nivel.

2. La mayoría de los algoritmos implementados en lenguajes procedimentales, como C, C++ o Java asumen que todos los datos pueden ser alojados en memoria. Esto crea un problema, cuando los datos que deben explorarse son más grandes que la memoria disponible, lo que en la realidad siempre es cierto, aunque algunos algoritmos inteligentes utilizan Muestreo (*Sampling*) o de Aprendizaje Gradual (*Incremental Learning*), la capacidad de manejar grandes cantidades o volúmenes de datos sigue siendo un problema con estos algoritmos.

3. Acceder a los datos de las fuentes como un Almacén de Datos y transformar los datos fuera de la Base de Datos es otro problema importante. Al usar SQL este problema puede ser minimizado, ya que los datos serán analizados dentro de la Base de Datos, sin necesidad de gastos adicionales a lo que se denomina sobrecarga (*overhead*) [21].

Como se mencionó anteriormente un DBMS es un grupo de programas que permiten controlar y manipular Bases de Datos. SQL es el estándar de los proveedores para manipular grandes conjuntos de datos. Con el fin de representar a las matrices como tablas

en un DBMS, es posible definir las columnas como dimensiones y las filas como registros. Para la matriz de entrada $\mathbf{X}$ la tabla de almacenamiento, `TX`, contiene los atributos $\{X_1, X_2, ..., X_d\}$. Esta matriz se utiliza para calcular la matriz de correlación que es un parámetro del algoritmo para encontrar SVD y resolver PCA. Otra forma de almacenar las matrices es a través de una disposición vertical [72]. Las matrices en disposición vertical $\{i, j, val\}$ son útiles cuando la dimensión del conjunto de datos es de la forma $d > n$. En esta tesis no se investiga este caso particular, sin embargo, es importante notar que realizar esta transformación puede ayudar a resolver conjuntos de datos con $d > n$.

2.8. Mecanismos de Extensibilidad en la arquitectura del DBMS

Los cálculos de modelos estadísticos generalmente se realizan fuera de un DBMS debido a su complejidad matemática, la mayoría del trabajo en el análisis de grandes conjuntos de datos ha propuesto algoritmos y técnicas eficientes que trabajan fuera del DBMS en archivos planos, es decir, los usuarios en general, exportan los conjuntos de datos a una herramienta estadística o de minería de datos, realizando la mayor o todo el análisis fuera del DBMS [23]. SQL se basa en el cálculo de álgebra relacional e incluye comandos para la Definición de Datos (*DDL Data Definition Language*), Definición de Vistas (*VDL View Definition language*), y Manipulación de Datos (*DML Data Manipulation Language*) [12, 13]. SQL podría ser la razón principal detrás del éxito de los DBMS Relacionales (*RDBMS Relational Data Base Management System*) [23], ya que la migración entre los proveedores de Bases de Datos se lleva a cabo con relativa facilidad. Sin embargo SQL es rígido y lento para procesar datos no relacionales. La mayoría de los DBMS proporcionan capacidades para incrustar código o agregar funcionalidad a SQL mediante lenguajes de programación como C++ y C# [30, 38, 86]. Estos mecanismos incluyen UDA, Función con Valores de Tabla (*TVF Table Valued Function*) y SP. Una vez que las UDFs se agregan al sistema pueden ser utilizadas libremente en consultas SQL. La principal ventaja de utilizar UDFs es su eficacia en la manipulación de datos en el DBMS. Proporciona una forma fácil de añadir implementaciones específicas a la funcionalidad de Base de Datos [38].

2.9. Funciones Definidas por el Usuario

Las UDFs se describen como una porción de código que permite extender la funcionalidad en un DBMS. Cada DBMS incluye su propia lista de funciones definidas, sin embargo, es posible agregar otras más. Esto permite a los usuarios finales extender el DBMS con funcionalidad de Minería de Datos. Como se mencionó anteriormente las UDFs se programan en lenguajes de programación de alto nivel y pueden ser llamadas en una sentencia `SELECT`, como cualquier otra función de SQL. Existen dos clases principales de UDFs:

1. **Escalares:** que toman uno o más valores como parámetro de entrada y producen un valor de salida por cada fila.

2. **Agregación:** que toman una colección de valores como entrada y producen un único valor de salida (Véase Fig. 2.9 que calcula el salario total de todos los empleados en la Tabla *Tabla-Empleados*).

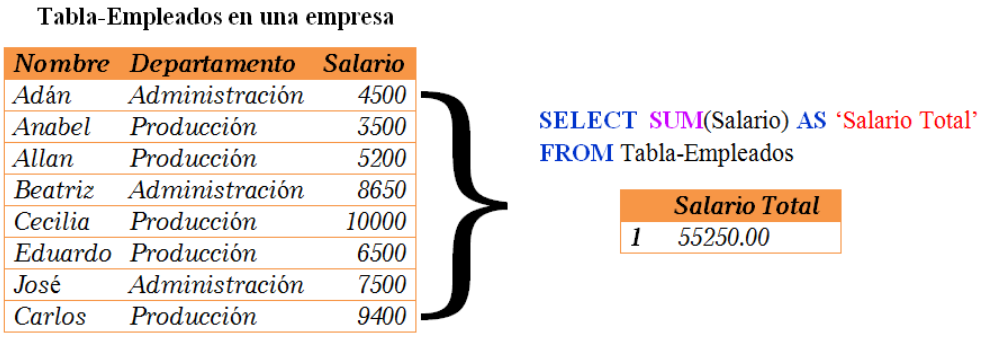

Figura 2.9: Ejemplo de función de Agregación en SQL.

Esta investigación se centra en el uso de UDA para acelerar el cálculo del modelo PCA. Las UDA son automáticamente ejecutadas en paralelo aprovechando las capacidades multiproceso del DBMS. Por otro lado, las UDFs tienen restricciones y limitaciones importantes. No pueden realizar ninguna operación de E/S, de esta manera se protege el almacenamiento interno de datos. En general, las UDFs sólo pueden devolver un valor o un tipo de datos simple. Es decir, no es posible devolver un vector o una matriz. Actualmente, los parámetros de entrada en las UDFs en la mayoría de los DBMS sólo pueden ser tipos simples (números o cadenas), de tal forma que el uso de vectores es limitado o no

está disponible. Las funciones escalares no pueden mantener valores en la memoria principal, mientras el DBMS escanea múltiples filas. Esto implica que la función sólo puede mantener variables temporales en su respectiva *Stack* o Pila (zona de la memoria donde se guardan variables locales, parámetros, etc.). Por el contrario, las UDA pueden mantener variables en la memoria *Heap* o Montículo al momento de leer una tabla. Las UDFs no pueden acceder a memoria fuera de su memoria Heap o Stack asignada. La cantidad de memoria que puede ser asignada es relativamente reducida. Finalmente, Las UDFs no son portábles, pero su código puede ser fácilmente reescrito a través de diversos DBMSs.

2.9.1. Funciones de Agregación Definidas por el Usuario

Las UDA realizan un cálculo de un conjunto de valores y devuelven un valor único. Para crear una UDA, se debe implementar el contrato de agregación descrito a continuación. El contrato de agregación incluye el mecanismo para guardar el estado intermedio de la agregación, así como el mecanismo para acumular nuevos valores, que consta de cuatro métodos: **INIT**, **ACCUMULATE**, **MERGE** y **TERMINATE** [92].

1. **INIT:** Función sin parámetros, que inicializa cualquier variable requerida por la agregación. Es decir, se invoca una vez para cada grupo que se vaya a agregar.

2. **ACCUMULATE:** Se invoca una vez para cada valor del grupo que se vaya a agregar. Actualiza el estado de la instancia para reflejar la acumulación del valor de argumento que se proporciona. Mientras la tabla es escaneada, los diversos hilos administrados por el DBMS acumulan las filas.

3. **MERGE:** Mezcla los valores acumulados de los hilos en un resultado principal. Se mezclan ambas, variables locales y estructuras de datos locales en un resultado agregado global.

4. **TERMINATE:** Función sin parámetros. En este paso todos los resultados parciales de los hilos han sido mezclados, así termina el cálculo de agregados y devuelve el resultado de la agregación al usuario.

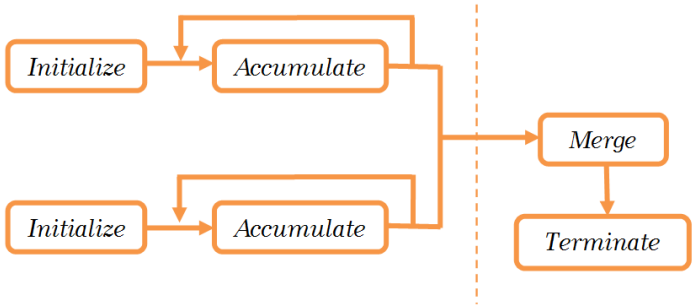

Figura 2.10: Pasos de las UDA.

2.9.2. Procedimientos Almacenados

Los SP son similares a los procedimientos de otros lenguajes de programación en el sentido de que pueden: Aceptar parámetros de entrada y devolver varios valores en forma de parámetros de salida al lote o al procedimiento que realiza la llamada. Contener instrucciones de programación que realicen operaciones en la base de datos, incluidas las llamadas a otros procedimientos. Devolver un valor de estado a un lote o a un procedimiento que realiza una llamada para indicar si la operación se ha realizado correctamente o se han producido errores (y el motivo de éstos). Difieren de las funciones en que no devuelven valores en lugar de sus nombres ni pueden utilizarse directamente en una expresión. Las UDA y SP se procesan en tiempo de ejecución de un manera diferente. Las UDA permiten un procesamiento multi-hilo, dejando la gestión de los hilos al DBMS. Por otro lado, una TVF lee el conjunto de datos de entrada como un flujo de datos (*Data Stream*). De tal forma que no existe un paralelismo implícito. Una TVF materializa los resultados en una tabla, haciendo uso de una interfaz de cursor (lector/escritor). Esto evita la exportación de datos y permite definir índices en la tabla resultante. Los Tipos Definidos por el Usuario (*UDTs User Defined Types*) permiten a los desarrolladores extender el sistema de tipos de datos escalares en un estilo orientado a objetos. Los UDTs normalmente consisten en varias columnas (similares a la estructura de una clase en el paradigma orientado a objetos) y simulan vectores. En este trabajo se utiliza el DBMS MS SQL, en particular el Entorno en Tiempo de Ejecución de Lenguaje Común (*CLR Common Language Runtime*) que es el núcleo de Microsoft .NET Framework y proporciona el entorno de ejecución de todo el código de .NET Framework. El código que se ejecuta en CLR se conoce como código administrado. El CLR proporciona diversas funciones y servicios necesarios pa-

ra la ejecución de los programas, como compilación Justo A Tiempo (*JIT Just-In-Time*), asignación y administración de memoria, aplicación de la seguridad de tipos, control de excepciones, administración de subprocesos y seguridad. Con el CLR hospedado en MS SQL Server (lo que se denomina integración con CLR), puede crear SP, Desencadenadores (*Triggers*), UDFs, UDTs y UDA en código administrado (*Managed Code*). Como el código administrado se compila a código nativo antes de su ejecución, en algunas situaciones puede conseguir aumentos significativos del rendimiento (Véase Fig. 2.11). MS SQL Server ofrece compatibilidad con la depuración de objetos Transact-SQL y CLR de la base de datos. La depuración funciona entre lenguajes; los usuarios pueden entrar sin problemas en los objetos CLR desde Transact-SQL y viceversa [3, 12].

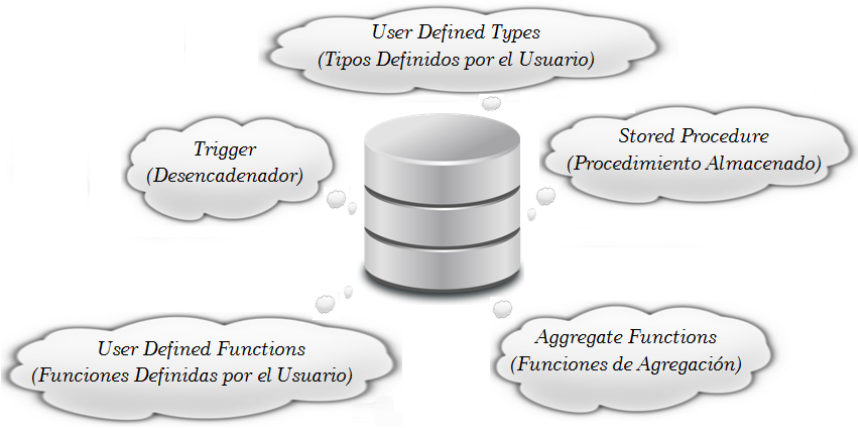

Figura 2.11: Objetos de Base de Datos Common Language Runtime (*CLR Objects*).

2.10. Bibliotecas de Métodos Numéricos y LAPACK

Existen bibliotecas especializadas externas las cuales son piezas de software que han sido desarrolladas con el tiempo para optimizar el tiempo de ejecución de los algoritmos, la mayoría de estas bibliotecas no están diseñadas para superar las limitaciones de memoria. Esta limitación se hace evidente cuando se intenta utilizar las bibliotecas para análisis estadísticos y minería de datos a grandes conjuntos de datos [47]. La mayoría de los trabajos de investigación se han centrado en la modificación de los algoritmos existentes para datos dispersos, y optimizar metodologías para resolver SVD [22]. Una de las

extensiones recientes de la investigación de PCA es la selección de características [14]. La selección de características encuentra un subconjunto del conjunto original de atributos que representa las características de los datos con un mínimo de pérdida de información. Teniendo en cuenta que hoy en día la mayor parte de la información es capturada utilizando un DBMS. Estas metodologías carecen de herramientas que realicen debidamente un preprocesamiento necesario para ejecutar sus algoritmos [71].

Con el fin de realizar operaciones sobre los datos contenidos en una base de datos, las herramientas externas requieren ser importadas en un formato específico [93]. Esto se hace mediante la exportación de datos en el DBMS como archivos planos o el uso de herramientas de conectividad de base de datos [87]. Los datos importados se utilizan para crear matrices por lo general almacenados en memoria principal para un acceso rápido. Los paquetes estadísticos se pueden utilizar para hacer todo o parte de las operaciones. Las principales funciones que necesitamos para resolver PCA son calcular la matriz de correlación y la descomposición de valores propios u otra función equivalente para resolver SVD. La mayoría de las herramientas de minería de datos y estadísticos tienen sus propias implementaciones de PCA. Por otra parte, algunos incluyen funciones para llevar a cabo otros análisis de DM de estos datos.

El paquete de Álgebra Lineal LAPACK es una biblioteca escrita en (*FORTRAN Formula Translating System*) que hace uso de las rutinas de bajo nivel de las librerías BLAS para realizar las operaciones básicas de álgebra lineal. La implementación llamada referencia de BLAS puede ser descargado desde Netlib. Por otra parte existe otra biblioteca popular llamada Intel © (*MKL Math Kernel Library*) 10.3 altamente optimizada, que contiene diversas rutinas matemáticas que resuelven modelos matemáticos como mínimos cuadrados, inversa de una matriz, factorización LU, entre otros [52]. Hay otras bibliotecas que son traducciones directas de LAPACK a otros lenguajes de programación como Dot-Numerics, que es un código fuente de LAPACK .NET escrito en C#. LAPACK y BLAS son las bibliotecas estándar que se utilizan por debajo de muchas aplicaciones o herramientas como (*MATLAB Matrix Laboratory*) o R. Estas bibliotecas de métodos numéricos son altamente escalables y portábles.

Capítulo 3

Trabajo Relacionado

Hay una amplia gama de aplicaciones exitosas basadas en el uso de PCA para el procesamiento de imágenes [40] y reconocimiento de patrones [98]. DM también ha explotado DR para la compresión de datos [19], Agrupación o Clustering [27] y clasificación [51]. Un tema interesante que ha surgido es la relación entre PCA y K-medias (*K-means*). PCA ha demostrado ser una solución de K-medias. Por lo tanto, las direcciones de los componentes principales se utilizan como un grupo de centroides en un subespacio. Este enfoque ayuda a encontrar soluciones cercanas a la óptima, ya que el subespacio proyectado es precisamente la ubicación de los grupos o Clusters. Además, PCA no sólo se utiliza para la reducción de dimensionalidad, sino también para la selección de variables [14]. Encontrar un subconjunto de atributos que representen mejor a un conjunto de datos es un problema NP-completo que puede ser atacado mediante PCA. Aunque ha habido una considerable cantidad de investigación en el Aprendizaje Maquinal y en la Minería de Datos para desarrollar técnicas eficaces y precisas, la mayoría de las investigaciones de minería de datos se ha centrado en proponer algoritmos eficientes bajo el supuesto de que el conjunto de datos es un archivo plano fuera del DBMS. La estadística y la Minería de Datos han prestado poca atención al análisis de grandes volúmenes de datos [20, 94]. La mayor parte de la investigación para integrar técnicas de DM en el DBMS, ha sido incorporar reglas de asociación [7], Clustering [6, 97], y árboles de decisión [39]. Un análisis de las alternativas para implementar bases de datos con funcionalidad de DM se presenta en [70]. Los DBMS tienen diferentes opciones para agregar funcionalidad a sus sistemas [87]. Por lo tanto, las implementaciones sólo pueden ser portables entre diversos DBMS si se utiliza un

estándar como SQL [91]. Se han presentado varias propuestas para agregar funcionalidad a SQL [50, 71]. La ejecución en paralelo de los cálculos agregados con UDA se analiza en [54]. Por otra parte, operaciones matemáticas, tales como operaciones vectoriales, se han implementado con éxito en el DBMS [77]. Técnicas de DM, tales como Procesamiento Analítico en Linea (*OLAP On-Line Analytical Processing*), Algoritmo EM [76, 96] y Algoritmos de clasificadores de árboles de decisión [85], han demostrado ser muy eficientes cuando se realizan dentro del DBMS [18].

Por otro lado existe una cantidad considerable de investigación en el rendimiento de Bases de Datos para optimizar el procesamiento de consultas en infraestructuras de cómputo modernas [20, 94]. Tradicionalmente para resolver PCA se exporta el conjunto de datos como archivos planos compatibles con las especificaciones de las herramientas estadísticas. Dado que el conjunto de datos se carga a memoria principal, el manejo de esta cantidad de información es un problema importante para realizar un análisis correcto del conjunto de datos. Debido a sus limitaciones para realizar operaciones con matrices y vectores complejas. Dada la complejidad del problema, las herramientas disponibles requieren importar parte o todos los datos, ya sea a memoria principal o como archivos binarios planos. Importar archivos de datos o cargar datos a memoria principal es una tarea poco práctica y costosa con grandes conjuntos de datos. Un enfoque relacionado para resolver SVD de una manera eficiente es utilizar una CPU o una GPU dependiendo el tamaño de la matriz [29, 62]. La mayor parte de la investigación se ha centrado en la utilización de LAPACK fuera del DBMS con arquitecturas multi-core [4, 66]. La investigación se enfoca en acelerar las operaciones físicas del DBMS a nivel de hardware, principalmente operaciones que explotan el uso de memoria RAM [4, 66]. Por esta razón, no existen intentos para llamar LAPACK a través de UDFs dentro de un DBMS [15, 16].

A continuación se muestra la factibilidad de resolver PCA a través de SVD de una matriz de correlación derivada de un gran conjunto de datos almacenado como una tabla relacional dentro de un DBMS [76, 80]. Se estudian y comparan varias alternativas para sumarizar grandes cantidades de datos utilizando UDFs, actualmente disponibles en un DBMS, y cómo llamar LAPACK de manera eficiente utilizando UDFs (Véase Fig. 3.1).

El algoritmo consiste en tres fases:

1. Calcular las Estadísticas Suficientes del Conjunto de Datos.

2. Obtener la Matriz de Correlación del Conjunto de Datos.

3. Resolver SVD con LAPACK utilizando la matriz de correlación como entrada.

Con base en este enfoque, el algoritmo puede analizar de manera eficiente un gran conjunto de datos leyendo el conjunto de datos una sola vez, eliminando la necesidad de exportar el Conjunto de Datos $\mathbf{X}$.

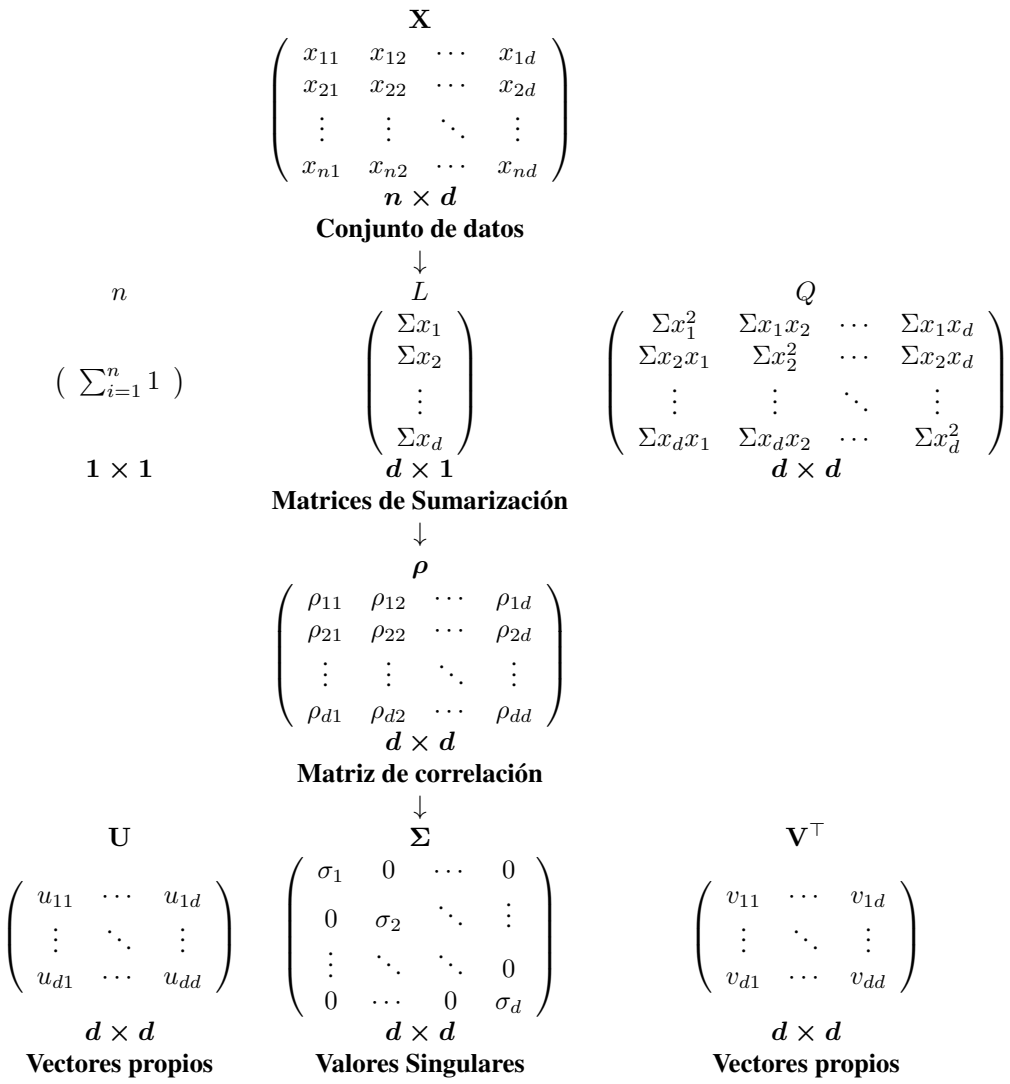

Figura 3.1: Notación matricial del Cálculo de SVD

Capítulo 4

Cálculo de PCA con Procedimientos Almacenados y LAPACK

Basándose en el estado del arte, la mejor manera de resolver SVD a partir de grandes volúmenes de datos es calcular los estadísticas suficientes n,L y Q, realizando una lectura de los datos en un tiempo polinomial $O(d^2n)$, después, efectuar el resto de los cálculos en $O(d^3)$ sobre la matriz de correlación o covarianza de tamaño $d \times d$ [75]. Un paso intermedio es reorganizar las entradas de la matriz de correlación en memoria principal. Se defenderá la idea de que la mejor forma de calcular SVD es integrar y utilizar la biblioteca LAPACK en el DBMS mediante tres pasos principales:

1. Calcular las matrices de sumarización n,L y Q.

2. Calcular y pasar la matriz de correlación ρ a LAPACK como un bloque en RAM.

3. Llamar al método SVD disponible en la biblioteca LAPACK.

Entrada: Conjunto de Datos $\mathbf{X}$: $\texttt{TX}(x_1, x_2, \ldots, x_d)$, Dimensión d
1 : (n, L, Q) $\leftarrow$ Matrices de Sumarización($\texttt{TX}$)
2 : (Matriz ρ) $\leftarrow$ Matriz de Correlación (n, L, Q)
(Vector ρ) $\leftarrow$ Orden Principal por Columnas(Matriz ρ)
3 : ($\mathbf{U}$, $\mathbf{\Sigma}$, $\mathbf{V}^\top$) $\leftarrow$ SVD(Vector ρ)
Salida: Vectores propios $\mathbf{U}$, $\mathbf{V}^\top$ y Valores Singulares $\mathbf{\Sigma}$

Figura 4.1: Algoritmo de PCA resuelto con SVD y LAPACK dentro del DBMS.

En las siguientes secciones, se comenta la implementación de cada paso del algoritmo, y como llamar LAPACK en el DBMS para calcular SVD (Véase figura 4.1).

4.1. Cálculo de las Matrices de Sumarización

Las Matrices de Sumarización definidas en la Sección 2.2 pueden ser calculadas utilizando tres métodos [22, 63, 82]. El Primer método utiliza consultas SQL. El Segundo método utiliza la función de lector y matrices en un enfoque de SP. El Tercer método utiliza UDA con UDTs como parámetro y como salida. Para calcular las matrices de sumarización en los enfoques SP y UDA, se utilizas una estructura de datos para almacenar n en una variable de precisión flotante, L en un arreglo de tamaño d y Q en una matriz bidimensional de tamaño $d \times d$. Se asume de antemano que esta estructura de datos puede ser almacenada en memoria principal. A continuación se detallan estos métodos.

i	*x1*	*x2*	*x3*	*x4*	*x5*	*x6*
1	*365*	*526*	*472*	*527*	*483*	*113*
2	*397*	*491*	*483*	*488*	*488*	*522*
3	*467*	*516*	*980*	*177*	*294*	*233*
4	*466*	*526*	*480*	*427*	*434*	*683*
5	*467*	*513*	*487*	*473*	*494*	*353*
6	*369*	*446*	*460*	*447*	*454*	*986*
7	*467*	*516*	*580*	*577*	*694*	*173*
8	*461*	*566*	*440*	*467*	*474*	*283*
9	*367*	*417*	*483*	*477*	*498*	*893*
10	*182*	*886*	*727*	*688*	*497*	*181*
11	*396*	*426*	*672*	*527*	*783*	*113*
12	*397*	*499*	*583*	*488*	*888*	*283*
13	*376*	*481*	*480*	*377*	*994*	*133*
14	*666*	*726*	*380*	*227*	*834*	*284*
15	*356*	*613*	*287*	*173*	*794*	*153*

i	j	val
1	1	365
2	1	397
3	1	467
⋮	⋮	⋮
1	2	526
2	2	491
3	2	516
⋮	⋮	⋮
1	15	113
2	15	522
3	15	233
⋮	⋮	⋮
15	15	153

Figura 4.2: Conjunto de datos $\mathbf{X}$ en disposición horizontal y vertical, $n = 15$ y $d = 6$.

4.1.1. Método con Consultas SQL

Suponemos que el conjunto de datos $\mathbf{X}$ está almacenado en una tabla en una disposición horizontal (*Horizontal Layout*) con d dimensiones por fila (Véase Fig. 4.2), de modo

que la lectura del conjunto de Datos basada en bloques sea rápida. El cálculo de n, L y Q puede ser expresado en términos de la agregación `SUM`. Una sola consulta SQL es utilizada para calcular todos los valores de las matrices de sumarización. Una propiedad de Q es que es simétrica, por esta razón es suficiente calcular solo los elementos de la parte superior o inferior triangular. Este enfoque consiste de una única instrucción `SELECT` con $1 + d + \frac{d(d+1)}{2}$ valores de agregación usando `TX`. Aquí, todos los valores de n, L y Q se calculan en el mismo agregado, dando como resultado un solo registro en disposición horizontal. Este enfoque requiere una sola lectura del conjunto de datos $\mathbf{X}$. Sin embargo, si el número de atributos d excede el número de columnas del DBMS, los cálculos deben tratarse como consultas y tablas separadas. En esta solución la primer sentencia obtiene el tamaño de $\mathbf{X}$ contando el número de filas, después para obtener L se suman las columnas del conjunto de datos $\mathbf{X}$ (en este caso son d sentencias debido a que $\mathbf{X}$ tiene d columnas empezando por $x_1, x_2, ..., x_d$), para obtener a Q se realizan $\frac{d(d+1)}{2}$ sentencias, realizando la suma de la multiplicación de dos columnas. Cuando se incluyen todos los valores de sumarización se crea la siguiente consulta SQL:

```
SELECT
    SUM(1.0)                                  /*Obtiene n*/
   ,SUM(x_1),SUM(x_2),...,SUM(x_d)            /*Obtiene L*/
   ,SUM(x_1 * x_1)                            /*Obtiene Q*/
   ,SUM(x_2 * x_1),SUM(x_2 * x_2)
   ,SUM(x_3 * x_1),SUM(x_3 * x_2),SUM(x_3 * x_3)
   ⋮
   ,SUM(x_d * x_1),SUM(x_d * x_2),...,SUM(x_d * x_d)
FROM TX;
```

Figura 4.3: Consulta SQL para obtener n, L y Q.

Aunque este método es teóricamente interesante y portáble, se mostrará que es muy lento [68, 75]. Además, este método se enfrenta a limitaciones del DBMS en el número máximo de columnas disponibles cuando $\mathbf{X}$ esta almacenada en un disposición horizontal (Véase Fig. 4.2). En la Figura 4.4 se ofrece otra forma de obtener n, L y Q en disposición vertical (*Vertical Layout*), es decir, cada resultado val se almacena con su respectivo indice i,j.

```
SELECT SUM(1.0) FROM TX;                    /*n*/
SELECT 1, SUM(x1) FROM TX UNION ALL /*L*/
⋮
SELECT d, SUM(xd) FROM TX;
SELECT a.x i, b.x j, SUM(a.s * b.s) FROM (   /*Q*/
        SELECT i, 1 x, x1, s FROM TX UNION ALL
        ⋮
        SELECT i, d x, xd, s FROM TX;
) a INNER JOIN (
        SELECT i, 1 x, x1, s FROM TX UNION ALL
        ⋮
        SELECT i, d x, xd, s FROM TX;
) b ON a.i = b.i AND a.x >= b.x GROUP BY a.x,b.x;
```

Figura 4.4: Consulta SQL para obtener n, L y Q en disposición vertical.

4.1.2. Método con SP

El método con SP utiliza una función Lector (*Reader*) para leer los registros de la tabla `TX` uno a la vez, después se calculan los valores de n, L y Q con los registros obtenidos (Véase Fig. 4.5). Los SP se ejecutan de manera secuencial en un solo hilo. Permiten la creación de listas, arreglos, o cualquier objeto Enumerable para después proyectarlo como una tabla. Los registros de `TX` son leídos uno por uno, mientras los valores son agregados a las matrices de sumarización.

```
double N;
double[] L;
double[,] Q;
int d;
double[,]rho;
public NLQ(int dimension) {
  d = dimension;
  N = 0.0;
  L = new double[d];
  Q = new double[d, d];
}
```

Entrada: $record$, d
1 :n += 1 /*n*/
2: For i=0 to d /*L y Q*/
3: L_i += $record_i$
4: For j=0 to i
5: $Q_{i,j}$=$Q_{j,i}$= $record_i$*$record_j$
6: End
7:End
Salida: n, L, Q

Figura 4.5: Definición de la clase NLQ y Función Reader.

Para cada registro en el conjunto de datos de entrada, almacenado como un arreglo x_i

de tamaño d, el cálculo de n, L y Q se realiza de la siguiente manera:

$n = n + 1$	El valor de n es incrementado.
$L = L + x_i$	El valor actual del registro es agregado a L.
$Q = Q + x_i \times x_i^T$	El producto cruz es agregado a Q.

4.1.3. Método con UDA

Como se mencionó anteriormente los tres cálculos de sumarización necesarios esenciales para DM y modelos estadísticos [43] son: n, L y Q. Estas matrices pueden ser calculadas en una sola lectura del conjunto de datos. Por otra parte las agregaciones en n, L y Q son distributivas [44]; en consecuencia, pueden calcularse en paralelo sobre diferentes secciones del conjunto de datos, donde la agregación global está dada por la adición de todos los resultados parciales [54]. Este método requiere como parámetro x_i como UDTs donde cada dimensión se convierte en un atributo (simulando un vector)[68, 79]. Los valores para los d atributos de un registro de datos deben ser almacenados como un objeto binario [79].

```
SELECT Agg( 'd'+ CAST(
    CAST(x1 AS VARCHAR)+ ','+
    CAST(x2 AS VARCHAR)+ ','+
    ⋮
    CAST(xd AS VARCHAR) AS Type)  )
FROM TX;
/*Declaración de UDTs para almacenar un objeto n,L,Q*/
DECLARE @nlq NLQ_OBJ_UDT;
/*UDA NLQ_Agg_UDF_6d de dimension 6* usando a data_Isolet_6_1M como entrada TX */
SELECT @nlq = dbo.NLQ_Agg_UDF_6d(
   NumberArray_6d::LoadValues(6, x1, x2, x3, x4, x5, x6)    )
FROM dbo.data_Isolet_6_1M ;
```

Figura 4.6: Consulta SQL y definición UDA.

Este método reserva y actualiza L, Q en memoria realizando un procesamiento multihilo. Posteriormente calcula la Matriz de correlación para permitir una llamada eficiente a LAPACK. La siguiente consulta SQL calcula n, L, Q y ρ donde d es la dimensión del conjunto de datos $\mathbf{X}$:

```
[Serializable] [Microsoft.SqlServer.Server.SqlUserDefinedAggregate ( Format.UserDefined,
IsInvariantToOrder=false,IsInvariantToNulls=true,IsInvariantToDuplicates=false,
IsNullEmpty=true,MaxbyteSize=-1) ]
public class NLQ_AggUDF_6d:IBnarySerialize {
  NLQ_Object_AggUDF store;
  public void Init(){
     store.m_Null = true;
  }
  public void Accumulate(NumberArray_6d nb){
     if(store.IsNull){
        store = new NLQ_Object_AggUDF(nb._d);
     }
     if( ! nb.IsNull) {
        /*Calcula n*/
        store.N += 1.0;
        store.d = nb._d;
        /*Calcula L y Q*/
        for( int i = 1; i <= stored.d; i++) {
           store.L[i] += Convert.ToDouble(nb._X[i]);
           for( int j = i; i <= stored.d; j++)
              store.Q[i,j] += Convert.ToDouble(nb._X[i]) * Convert.ToDouble(nb._X[j]);
         }
     }
  }
  public void Merge(NLQ_AggUDF_6d Group){
     if(store.IsNull){
        store = new NLQ_Object_AggUDF(Group.store.d);
     }
      /*Mezcla las soluciones de los hilos*/
     store.d = Group.store.d;
     store.N += Group.store.N;
        for( int i = 1; i <= stored.d; i++) {
          store.L[i] += Group.store.L[i];
          for( int j = i; i <= stored.d; j++) {
             store.Q[i,j] = store.Q[j,i] += Group.store.Q[i, j]);
           }
       }
  }
  [SqlFunction(Name = "Terminate")]
  [return:SqlFacet(MaxSize= -1)]
  public NLQ_Object_AGGUDF Terminate(){
     return store;
  }
```

Figura 4.7: Contrato de Agregación en UDA.

n
15

i	L_i
1	6199
2	8148
3	7994
4	6540
5	9103
6	5386

i	Q_{i1}	Q_{i2}	Q_{i3}	Q_{i4}	Q_{i5}	Q_{i6}
1	2704845	3342738	3259110	2595537	3798085	2226899
2	3342738	4632954	4352309	3565488	4950946	2751565
3	3259110	4352309	4636102	3525065	4688269	2731475
4	2595537	3565488	3525065	3144420	3915322	2376084
5	3798085	4950946	4688269	3915322	6113363	2915463
6	2226899	2751565	2731475	2376084	2915463	3057676

Figura 4.8: Matrices n, L, Q del Conjunto de datos $\mathbf{X}$.

4.2. Cálculo de la Matriz de Correlación

La matriz de correlación ρ se calcula de manera secuencial utilizando arreglos en el SP y en la UDA (Véase Fig. 4.10). Una vez que las matrices de sumarización n, L y Q han sido mezcladas en la sección **MERGE**, la matriz de correlación es calculada secuencialmente en la sección de **TERMINATE** con el objeto n, L, Q para realizar la llamada a LAPACK. En la Figura 4.9 se ofrece otra forma de obtener ρ utilizando n, L y Q (Véase Figura 4.4) en disposición vertical.

```
SELECT
        (N.n*Qab.q)-(La.l*Lb.l)
  ------------------------------------------------------------------------
  SQRT((N.n*Qaa.q)-(pow(La.l,2))*SQRT((N.n*Qbb.q)-(pow(Lb.l,2))
FROM N
CROSS JOIN L as La
CROSS JOIN L as Lb
JOIN Q as Qab ON   Qab.i = La.i  AND Qab.j = Lb.i
JOIN Q as Qaa ON   Qaa.i = La.i  AND Qaa.j = La.i
JOIN Q as Qbb ON   Qbb.i = Lb.i  AND Qbb.j = Lb.i;
```

Figura 4.9: Consulta SQL para Calcular la Matriz de Correlación ρ a partir de n,L y Q.

Las UDA utilizan múltiples hilos para distribuir la carga de trabajo. Los hilos concurrentes garantizan resultados sin condiciones de carrera o bloqueos. Todos los hilos comparten memoria para actualizar el cálculo global de la agregación [54, 59, 90].

Entrada: n, L, Q, d
1 :Rho[d,d]
2: For $a = 0$ to d
3: For $b = 0$ to a
4: Rho[a, b] = Rho[b, a]= $\frac{nQ_{ab} - L_a L_b}{\sqrt{nQ_{aa} - L_a^2}\sqrt{nQ_{bb} - L_b^2}}$
5: End
6:End
Salida: Rho

Figura 4.10: Función Rho que calcula ρ.

Con el fin de utilizar LAPACK en el DBMS las filas de la matriz de correlación ρ deben estar alojadas en memoria contigua (i.e, bloque ininterrumpido en memoria principal), y en un estilo FORTRAN de Orden Principal por Columnas (*Column Major Order*). Para lograr esto se crea un arreglo unidimensional de tamaño $d \times d$.

Entrada: ρ, d
1:flatRho [d*d];
2:For $i = 1$ to d
3: For $j \leq i$ to d
4: flatRho[$i * d + j$] = flatRho[$j * d + i$] = Rho[i, j];
5: End
6:End
Salida: flatRho

Figura 4.11: Matriz ρ en un bloque contiguo de memoria.

Para cualquier índice dado i,j de una matriz de tamaño $d \times d$, la dirección es $i * d + j$. Como ρ es una matriz simétrica $\rho_{ij} = \rho_{ji}$, es decir, la correlación de la columna i con la columna j es la misma que la correlación de la columna j, con la columna i, los valores almacenados en memoria serán los mismos en el Orden Principal por Columnas o en el Orden Principal por Filas (Véase Fig 4.11).

En el Orden Principal por Columnas de FORTRAN el último indice cambia más lento mientras que el primer índice cambia con mayor rapidez (Véase Fig. 4.12). Como ambos

arreglos están en memoria principal y la conversión de una matriz bidimensional a un arreglo unidimensional ocurre en memoria principal, el tiempo requerido que toma realizar este proceso es insignificante.

Matriz A

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix}$$

$\mathbf{3 \times 3}$

$\downarrow$

Vector A

$$[A_{11}, A_{21}, A_{31}, A_{12}, A_{22}, A_{32}, A_{13}, A_{23}, A_{33}]$$

$\mathbf{1 \times 9}$

Figura 4.12: Conversión de una matriz **A** a un vector unidimensional **A** con un Orden Principal por Columnas.

i	ρ i1	ρ i2	ρ i3	ρ i4	ρ i5	ρ i6
1	1	-0.142753907555	-0.19213818633	-0.52385326877	0.124444699394	0.002606132276
2	-0.142753907555	1	0.035741740609	0.052631092693	0.017746700663	-0.361033166149
3	-0.192138186332	0.035741740609	1	0.1195821077820	-0.346476373927	-0.213738934536
4	-0.52385326877	0.052631092693	0.1195821077820	1	-0.128989440426	0.048428923563
5	0.124444699394	0.017746700663	-0.346476373927	-0.128989440426	1	-0.434021957325
6	0.002606132276	-0.361033166149	-0.213738934536	0.048428923563	-0.434021957325	1

Figura 4.13: Matriz de correlación ρ de la Tabla **X** (`TX`).

4.3. Llamada al método SVD desde la Biblioteca LAPACK

Como se mencionó en la sección 2.10, LAPACK es una biblioteca de métodos numéricos ampliamente conocida escrita en FORTRAN que hace uso de BLAS. BLAS y LAPACK son software open-source (disponible en Netlib). Existe otra popular biblioteca llamada Intel© MKL 10.3, que es una biblioteca optimizada para las arquitecturas multi-core Intel©. El núcleo de la biblioteca incluye BLAS, LAPACK y LAPACK Escalable (*ScaLAPACK Scalable LAPACK*). Existe otra alternativa llamada DotNumerics, la cual es el código fuente de LAPACK reescrito en C# para el ambiente de programación de

Microsoft .NET. Dada la importancia y uso actual de dichas bibliotecas estudiamos como utilizar LAPACK, MKL y el código fuente DotNumerics C# vía UDFs para resolver SVD. Debemos enfatizar que LAPACK y BLAS son bibliotecas estándar, eficientes, altamente escalables y portables que garantizan alta precisión, utilizadas debajo de herramientas matemáticas como MATLAB o R.

Existen implementaciones de SP para resolver SVD que no utilizan bibliotecas de métodos numéricos [46]. Basados en la investigación, existen tres formas posibles de llamar de manera eficiente la biblioteca LAPACK:

1. Código FORTRAN de LAPACK de Netlib utilizando una clase de contención (*wrapper*) para mandar a llamar a las bibliotecas dinámicas .

2. Código Administrado (*Managed Code*) y traducido DotNumerics en C#.

3. Biblioteca de métodos numéricos proporcionada por Intel© MKL No Administrado (*Unmanaged Code*).

4.3.1. Código FORTRAN de LAPACK

Netlib ofrece la biblioteca LAPACK que proporciona una versión de BLAS la cual no esta optimizada para cualquier arquitectura. Esta implementación de referencia a BLAS puede ser varios órdenes de magnitud más lento que otras implementaciones optimizadas, en cuanto a factorización de matrices y otras operaciones computacionalmente intensivas. Los pasos requeridos para llamar la función SVD de LAPACK dentro del DBMS son:

1. Descargar las bibliotecas precompiladas de Netlib (liblapack.lib, liblapack.dll, libblas.lib libblas.dll).

2. Crear una clase, cuyo propósito es mapear y llamar la rutina de LAPACK.

3. Agregar las bibliotecas al proyecto en C++ “Linker - General - Additional Library Directory“ y “Linker - Input - Additional Dependencies“ (Véase Fig. 4.14).

4. Generar una Biblioteca de Enlace Dinámico (*DLL Dynamic-link library*) con las funciones LAPACK requeridas (Véase Fig. 4.15).

5. Llamar el DLL generado desde el SP escrito en C# con el atributo *DllImport*.

```
extern“C“ void dgesvd_(
    char* jobu, char* jobvt, int* m, int* n,
    double* a,int* lda,double* s, double* u, int* ldu,
    double* vt, int* ldvt,double* work, int* lwork, int* info
)
extern“C“{
    _declspec(dllexport) double* svd(
            double arr[],int dimension,
            double u[], double s[]
    );
}
```

Figura 4.14: Declaración de la función externa dgesvd_ y exportación de la función personalizada svd.

El código siguiente demuestra como utilizar la función SVD disponible en el Netlib dgesvd en un programa C++ y crear otra función personalizada svd a partir de esta. La función recibe ρ como vector, la dimensión y los vectores u y s que servirán para guardar el resultado como parámetros de entrada.

```
extern double* svd(double arr[],int dimension, double u[], double s[] ) {
  int m = dimension, n = dimension, lda = m, ldu = m, ldvt = n, info, lwork;
  double wkopt;
  double* work;
  double *vt =(double *)malloc( n*n *sizeof(double));
  lwork = -1;
  dgesvd_(“All“,“All“,&m,&n,arr,&lda,s,u,&ldu,vt,&ldvt,&wkopt,&lwork,&info);
  lwork = (int)wkopt;
  work = (double*)malloc( lwork*sizeof(double));
  dgesvd_(“All“,“All“,&m,&n,arr,&lda,s,u,&ldu,vt,&ldvt,work,&lwork,&info);
  free( (void*)work );
  return 0;
}
```

Figura 4.15: Uso de la función externa dgesvd_ dentro de la función personalizada svd.

La función personalizada svd alojada en un DLL es invocada utilizando *DLLImport* en otro programa C# (Véase Fig. 4.16).

```
public partial class StoredProcedures {
      [DllImport("C:\svd.dll", EntryPoint = "svd")]
      unsafe public static extern double* svd(
     double arr[],int dimension,double u[], double s[]
   );
      [Microsoft.SqlServer.Server.SqlProcedure]
         unsafe public static void LAPACK_PRECOMPILED (
         SqlString table,
         [SqlFacet(MaxSize=-1)] SqlString columns,
         int dimension,
         NLQ_Object_AGGUDF NLQobj,
         SqlString table NLQcompType,
      )
      ⋮
      double[] u = new double[dimension*dimension];
      double[] s = new double[dimension];
      /*Llamada a función svd que llama al código FORTRAN*/
      double* u_s = svd(flatRho, dimension, u , s);
      ⋮
}
```

Figura 4.16: Uso de la función externa svd en SP.

4.3.2. Código Fuente en C#

Esta versión utiliza código fuente en C#. Es una traducción de la biblioteca original LAPACK escrita en FORTRAN. Esta versión es la más fácil de implementar entre las tres versiones ya que no hay necesidad de crear una mapeo con CLR .NET como en la versión anterior. Esta versión es fácil de implementar (Véase Fig. 4.17).

```
Matrix U;
Matrix S;
Matrix Vt;
SingularValueDecomposition svd = new SingularValueDecomposition();
svd.ComputeSVD(flatRho, out S, out U, out Vt);
```

Figura 4.17: Función SVD en DotNumerics.

4.3.3. Código Objeto de Intel MKL

Como se mencionó en la sección 2.10, la biblioteca MKL proporciona implementaciones de BLAS y LAPACK. La interfaz LAPACKE es utilizada directamente en esta biblioteca, esta interfaz permite utilizar matrices en ambos Orden Principal por Columnas u Orden Principal por Filas, el cual se define en el primer parámetro de la función.

Los pasos principales para llamar a LAPACK utilizando MKL son:

1. Descargar la biblioteca MKL de Intel© e Importarla (mkl_rt.dll).
2. Crear una clase, cuyo propósito es ser un intermediario y escribir una función en C# que importe el núcleo principal de MKL (mkl_rt.dll) utilizando *DllImport*.
3. Llamar la interfaz LAPACKE.
4. Llamar la función dentro de la clase intermediaria desde el SP escrito en C#.
5. Pasar como parámetros los valores requeridos por la función y realizar SVD.

El procesamiento multi-hilo está activado por omisión y por lo tanto no hay necesidad de hacer algún ajuste especial durante el uso de CPUs multi-core. La función de MKL debe ser declarada con el atributo *Import* y los parámetros de Aplanamiento (*Marshalling*) necesarios. Es importante hacer énfasis en los parámetros de Entrada y Salida de la función LAPACKE_dgesvd ya que se utilizan para devolver los valores en las mismas variables, además de que las variables se declaran como *Int32*, esto es sólo para hacer hincapié en que están siendo utilizados enteros de 32 bits. La función LAPACKE_dgesvd creada dentro de la importación se invoca desde el SP para calcular SVD [69]. El método para realizar lo anterior se describe en la Figura 4.18.

```
namespace mkl
internal sealed class MKLImports {
      [DllImport("C:\mkl_rt.dll", ExactSpelling=true,
            SetLastError=false,CallingConvention=CallingConvention.Cdecl)]
      internal static extern void LAPACKE_dgesvd(
        int matrix_order,char a1,char a2,
        int m,int n,
        [In,Out] double[] s,
        int lda,
        [In,Out] double[] u,
        int ldu,
        [In,Out] double[] vt,
        int ldvt,
        double[] superb,
     );
  }
using mkl;
namespac mklDirect {
public partial class StoredProcedures {
      [Microsoft.SqlServer.Server.SqlProcedure]
         unsafe public static void LAPACK_PRECOMPILED (
         SqlString table,
         [SqlFacet(MaxSize=-1)] SqlString columns,
         int dimension,
         NLQ_Object_AGGUDF NLQobj,
         SqlString table NLQcompType,
      )
      :
      Char jobu = 'A';
      Char jobvt = 'A';
      int mat_order = 101;
      Int32 lda = dimension, ldu = dimension, ldvt = dimension;
      Double[] u = new double[dimension * dimension];
      Double[] s = new double[dimension];
      Double[] vt = new double[dimension * dimension];
      Double[] superb = new Double[dimension -1];
      /*Llamada a función svd importada de MKL*/
      MKLImports.LAPACKE_dgesvd(
         mat_order, jobu, jobvt, dimension, dimension, flatRho,
         lda, s, u, ldu, vt, ldvt, superb
      );
     }
    }
  }
```

Figura 4.18: Código Objeto MKL.

En las figuras 4.19, 4.20 y 4.21 se muestran las matrices $\mathbf{U}$,$\mathbf{\Sigma}$ y $\mathbf{V}^\top$ respectivamente. Estas matrices son el resultado de calcular SVD a partir de la matriz de correlación ρ (Fig. 4.13) calculada utilizando las matrices de sumarización (Fig. 4.8) que resumen al conjunto de datos original $\mathbf{X}$ almacenado como tabla `TX` (Fig. 4.2). Observamos que los primeros cuatro (de un total de seis) valores singulares en orden descendente en $\mathbf{\Sigma}$ proporcionan mas del 85 % de la varianza. Así, con esta información es posible realizar una proyección utilizando los primeros cuatro nuevos ejes, definidos por los primeros cuatro vectores propios. De esta manera SVD resuelve PCA reduciendo la dimensión $d = 6$ a $d = 4$.

i	u_{i1}	u_{i2}	u_{i3}	u_{i4}	u_{i5}	u_{i6}
1	-0.568204264304213	0.191580010050053	0.344459905939842	0.057623813300714	-0.71186678745823	0.108229201374774
2	0.092203508694887	-0.529503080968478	0.114920527567713	0.783048200494938	-0.05359889705594	0.286148469345589
3	0.399921613786302	-0.136693414712789	0.669267980471650	-0.40473551946504	0.004693646229437	0.457848350968476
4	0.545763234721263	-0.103897122514854	-0.45002735934447	-0.13062437793060	-0.68558368084758	0.041659858939764
5	-0.440777193839364	-0.406362489876042	-0.42573002815102	-0.36086660658803	0.093306539446761	0.566046038136009
6	0.128697656758384	0.698794255931705	-0.19091970781153	0.269103577206925	0.107794763719302	0.612075746815743

Figura 4.19: Matriz $\mathbf{U}$ con los vectores propios de la matriz ρ.

i	σ_{i1}	σ_{i2}	σ_{i3}	σ_{i4}	σ_{i5}	σ_{i6}
1	1.757508326665900	0	0	0	0	0
2	0	1.561285064351230	0	0	0	0
3	0	0	1.108208836182120	0	0	0
4	0	0	0	0.829989535414321	0	0
5	0	0	0	0	0.469300557533083	0
6	0	0	0	0	0	0.273707679853350

Figura 4.20: Matriz $\mathbf{\Sigma}$ con los valores propios de la matriz ρ.

i	v_{i1}	v_{i2}	v_{i3}	v_{i4}	v_{i5}	v_{i6}
1	-0.568204264304213	0.092203508694887	0.399921613786302	0.545763234721263	-0.44077719383936	0.128697656758384
2	0.191580010050053	-0.529503080968478	-0.13669341471278	-0.10389712251485	-0.40636248987604	0.698794255931705
3	0.344459905939842	0.114920527567713	0.669267980471650	-0.45002735934447	-0.42573002815102	-0.19091970781153
4	0.057623813300714	0.783048200494938	-0.40473551946504	-0.13062437793060	-0.36086660658803	0.269103577206925
5	-0.711866787458234	-0.053598897055946	0.004693646229437	-0.68558368084758	0.093306539446761	0.107794763719302
6	0.108229201374774	0.286148469345589	0.457848350968476	0.041659858939764	0.566046038136009	0.612075746815743

Figura 4.21: Matriz $\mathbf{V}^\top$ con los vectores propios de la matriz ρ.

4.4. Análisis de Tiempo Polinomial

El número de Operaciones de Punto Flotante (*FLOPS Floating Point Operations*), el tiempo Polinomial $O()$ y los pasos de E/S para los pasos descritos se muestran en la Tabla 4.1. La primera columna muestra el número de FLOPS requeridos para el cálculo de n, L y Q. La segunda columna muestra el costo requerido para calcular la matriz de correlación ρ y a su vez convertirla en un bloque contiguo de memoria. La última columna muestra el número de FLOPS requeridos para realizar la descomposición SVD de ρ [9, 26].

Medida	Matrices de Sumarización. n, L, Q	Cálculos Matriz de Correlación ρ	Llamada a SVD LAPACK
FLOPS	$nd^2/2$	$10d^2$	$\frac{8}{3}d^3 + 12d^3$
Tiempo	$O(nd^2)$	$O(d^2)$	$O(d^3)$
Costo E/S	n	0	$d + d^2$

Tabla 4.1: Tiempo Polinomial para realizar PCA

Capítulo 5

Evaluación Experimental

En este capitulo se presenta la evaluación experimental que involucra los tres pasos principales para resolver PCA: determinar las matrices de sumarización, obtener la matriz de correlación y calcular SVD, explicados en los capítulos 2 y 4. Se comparan los tiempos de ejecución en tres configuraciones de Hardware: arquitecturas con dos núcleos, cuatro núcleos, y una arquitectura simulando dos núcleos. Se realizaron experimentos en las tres versiones del cálculo de las matrices de sumarización, la matriz de correlación ρ y SVD utilizando los dos servidores. Se registran siete tiempos de ejecución, eliminando el valor máximo y el valor mínimo para después obtener el promedio de los cinco valores restantes. Todos los resultados se expresan en segundos. La memoria del DBMS (*Buffer*) se reinicia antes de cada ejecución para asegurar que el conjunto de datos sea leído desde disco (*Almacenamiento secundario*).

5.1. Arquitectura y Software utilizados

Se realizaron experimentos en dos servidores con dos y cuatro núcleos respectivamente. Dado que es prácticamente imposible conseguir dos servidores con exactamente la misma configuración de Hardware y Software (con un número diferente de núcleos), no es factible realizar una comparación de la Aceleración (*Speedup*) en ambos servidores. Para resolver este situación se normalizaron los resultados de ambos servidores utilizando la velocidad de reloj (*Clock Speed*) como criterio principal, aunque diversos factores como la velocidad del Bus de datos y la memoria asignada por el sistema operativo complican la

tarea. Estos factores deben considerarse para la normalización. Además, también se realizaron experimentos desactivando dos de los cuatro núcleos del Servidor Quad, haciendo uso de la interfaz de SQL Server 2008. De esta manera el servidor funciona como un servidor de dos núcleos. Los detalles de ambos Servidores se muestran en la Tabla 5.1.

	Servidor 1 Dual Core	Servidor 2 Quad Core
Intel Xeon	E3110	X3210
Número de Núcleos	2	4
Número de Hilos	2	4
L2 Cache	6 MB	8 MB
Velocidad de reloj	3 GHz	2.13 GHz
Radio Bus/Core	9	8
Velocidad Bus Frontal	1333 MHz	1066 MHz
Intel Hyper Threading	No	No
Memoria RAM	4 GB	4 GB
Sistema Operativo	Win Server '03 32-bit	Win XP '02 32-bit
Memoria Asignada por el S.O.	4 GB	3.50 GB
DBMS	SQL SERVER 08	SQL SERVER 08

Tabla 5.1: Especificaciones del Sistema.

5.2. Descripción del Conjunto de Datos

Se utilizó el conjunto de datos **ISOLET** del Repositorio de Conjuntos de Datos de Aprendizaje Maquinal de la Universidad de California, Irving (*UCI University of California, Irvine Machine Learning repository*). El conjunto original tiene 617 atributos y 7797 puntos. Se crearon diversos conjuntos de datos de distintas dimensiones con el tamaño constante de $n = 1$ millón $(1M)$ mediante la replicación de puntos y funciones aleatorias para seleccionar dimensiones diferentes para cada conjunto de datos. El conjunto de datos $\mathbf{X}$ se almacena en una Tabla del DBMS con d columnas con el tipo de dato flotante para trabajar con la misma precisión que LAPACK.

5.3. Medición de Exactitud

Se realizó una medición de exactitud en los Conjuntos de Datos obtenidos de UCI (Véase la Tabla 5.2) comparando los resultados obtenidos con los resultados del paquete estadístico R. Se obtuvo 0.00 % como máximo error relativo al comparar las diferentes versiones de SVD contra R. La razón es que LAPACK es una biblioteca de métodos numéricos estándar utilizada como base por paquetes como R y MATLAB entre otros. La precisión numérica no es un problema porque se utilizó la biblioteca LAPACK. Tal aspecto sería crítico si los métodos numéricos fueran reprogramados. En general, el error relativo fue de $\epsilon = 1E^{-5}$.

Conjunto de Datos	d	n
Insurance Company Benchmark (COIL 2000) Data Set	86	5822
Letter Recognition Data Set	16	20000
MiniBooNE particle identification Data Set	50	130065
Waveform Database Generator (Version 2) Data Set	40	5000
Spambase Data Set	55	4601

Tabla 5.2: Conjunto de Datos de UCI para medir la precisión.

5.4. Comparación de tiempo de ejecución

Se ejecutaron experimentos en un servidor con cuatro núcleos. Para cada ejecución la memoria RAM se reinicia, asegurando que el Conjunto de Datos $\mathbf{X}$ es leído desde disco. En la Tabla 5.3 se comparan los tiempos en resolver PCA utilizando el paquete estadístico R, Consultas SQL, SP y UDFs utilizando MKL, la variante más rápida de LAPACK en un Conjunto de Datos con dimensionalidad $d = 100$.

n	Bulk Export	PCA R	PCA Consultas SQL	PCA SP	PCA UDA+ LAPACK
100k	43	143	2411	27	4
1M	188	1440	4030	149	41
10M	1890	14430	14173	1343	400

Tabla 5.3: Tiempo de ejecución para resolver PCA variando n y $d = 100$

Se observa que el tiempo de exportación del conjunto de datos $\mathbf{X}$ utilizando el mecanismo *Bulk Insert* es un cuello de botella. De hecho, el tiempo para exportar $\mathbf{X}$ es mayor que el tiempo en procesar los datos con SP o UDA. Se compara el tiempo de ejecución con el paquete estadístico R realizando los mismos pasos para realizar PCA. Dado que $\mathbf{X}$ no puede almacenarse en RAM, $\mathbf{X}$ debe ser procesada en bloques de $100K$ filas para obtener ρ. En este caso el algoritmo tiene un mejor desempeño que R con n grande. Sin embargo, considerando el tiempo de exportación, las consultas SQL mejoran el tiempo total en comparación a R. Se observa que los SP son más rápidos que las consultas SQL. La brecha entre UDA y SP se vuelve significativa cuando $n = 10M$. Además, UDA muestra una escalabilidad lineal. En resumen, la solución utilizando UDA es más eficiente que las demás [78].

5.4.1. Tiempo de ejecución por pasos para resolver PCA con n grande

La Tabla 5.4 presenta un desglose de PCA en dos pasos principales: Calcular las matrices de sumarización n, L, Q y resolver SVD a partir de la Matriz de Correlación ρ. La mayor parte del tiempo se consume en resumir $\mathbf{X}$, realizando la lectura de la Tabla con $O(d^2)$ FLOPS. Las UDA tienen un mejor desempeño en comparación con el SP para calcular n, L, Q, esto es debido a que las UDA procesan los datos en paralelo. Las consultas SQL se vuelven ineficientes para realizar cálculos con matrices con d grande. Por otra parte, los SP resuelven SVD, pero no son los más rápidos [78].

n	NLQ Consultas SQL	NLQ SP	NLQ UDA	SVD Consultas SQL	SVD SP	SVD LAPACK
100k	132	13	4	2279	14	0.01
1M	1127	132	41	2903	17	0.01
10M	11290	1326	400	2518	17	0.01

Tabla 5.4: Tiempo de ejecución por pasos para resolver PCA con n grande y $d = 100$.

5.4.2. Tiempo de ejecución variando d y $n = 1M$

Dado que las UDA tienen un mejor desempeño que el paquete R y que las consultas SQL, resulta innecesario analizar su escalabilidad variando d. La Tabla 5.5 compara los tiempos de ejecución variando d y manteniendo constante a n. El tiempo crece para las

matrices n, L, Q y SVD $O(n^2)$. Los tiempos para LAPACK se hacen evidentes para una d más grande (apenas por encima de un segundo), pero todavía son órdenes de magnitud menores que el tiempo que toma resumir a $\mathbf{X}$ con n, L y Q. Las UDA son mucho más rápidas que el SP, pero la diferencia en comparación al tiempo que toma en n es menos significativa, destacando $O(n^2)$ FLOPS [78].

d	NLQ SP	NLQ UDA	SVD SP	SVD LAPACK
10	8	7	0	0.00
50	17	42	1	0.00
100	41	132	17	0.01
200	121	460	197	0.04
400	432	1714	2683	0.28
800	1731	3646	12974	1.24

Tabla 5.5: Tiempo de ejecución variando d y $n = 1M$.

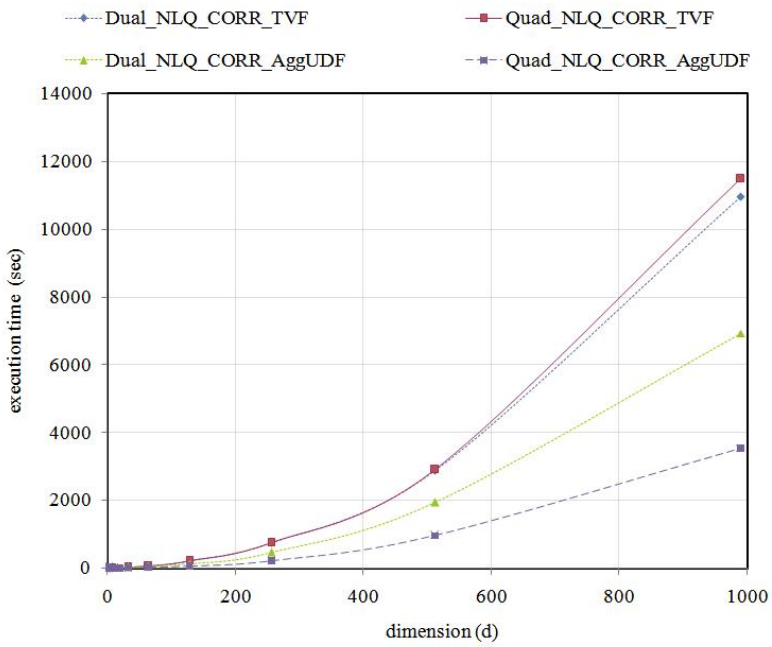

Figura 5.1: Cálculo de las matrices de sumarización con $n = 1M$ (Dual core vs Quad core).

5.5. Cálculo de las matrices de sumarización

Las matrices de sumarización y la matriz de correlación calculadas usando la función Lectora (*Reader*), disponible en el SP, tomaron más tiempo en los servidores de dos y cuatro núcleos en comparación con el otro enfoque utilizando UDA. Los resultados de n, L y Q se muestran en la Tabla 5.6. Las funciones SP se ejecutan de manera secuencial en un solo hilo, por lo tanto no hay ninguna forma de paralelizar los cálculos. La lectura de la Tabla $\mathbf{X}$ se lleva a cabo de forma secuencial, por lo tanto no hay ninguna ventaja al usar más núcleos. En el caso de las UDA los cuatro núcleos tienen un mejor desempeño que dos núcleos. Se explota la paralelización de los núcleos al momento de realizar la lectura de la Tabla $\mathbf{X}$ utilizando múltiples hilos en la fase de acumulación, calculando los valores correspondientes de n, L y Q en cada hilo. El resultado combinado de cada uno de los hilos se mezcla en la fase **MERGE** y el resultado se devuelve en la fase **TERMINATE**. La matriz de correlación también se calcula en la fase **TERMINATE** de manera secuencial. Es evidente que las UDA en el servidor de cuatro núcleos realizan casi dos veces más rápido el cálculo de n, L y Q a comparación con el servidor de dos núcleos (Véase Tablas 5.7 y 5.8). Obtener las matrices de sumarización utilizando UDA es la manera más eficiente cuando el DBMS ofrece varios núcleos [78].

d	Dual SP	Quad SP	Dual UDA	Quad UDA
2	4	4	23	23
4	5	4	13	8
8	7	6	15	9
16	10	10	18	10
20	13	13	14	9
32	23	23	27	16
64	64	65	53	32
128	208	209	139	65
256	751	749	456	217
512	2874	2902	1943	957
800	7195	6888	4613	2312
990	10958	11496	6934	3549

Tabla 5.6: Cálculo de las matrices de sumarización $n = 1M$ (Dual core vs Quad core).

d	Dual	Quad	Radio
2	23.13	23.18	0.99
4	13.06	8.28	1.57
8	14.86	8.96	1.68
16	17.82	9.86	1.80
20	13.70	8.97	1.52
32	26.61	16.11	1.64
64	53.41	31.73	1.68
128	139.26	65.01	2.14
256	456.02	216.93	2.10
512	1943.21	957.41	2.02
990	6934.03	3549.17	1.95

Tabla 5.7: Aceleración de las matrices de sumarización $n = 1M$ (Dual core vs Quad core).

d	Dual SP	Quad SP	Speedup Velocidad del reloj	Speedup Velocidad del Bus
2	17	23	1.05	0.93
4	10	8	1.70	1.51
8	11	9	1.73	1.53
16	12	10	1.66	1.48
20	8	9	1.29	1.14
32	17	16	1.49	1.32
64	34	32	1.50	1.33
128	91	65	1.96	1.74
256	299	217	1.94	1.72
512	1033	957	1.52	1.34
990	3874	3549	1.53	1.36

Tabla 5.8: Aceleración de las matrices de sumarización $n = 1M$ (Dual core vs Quad core).

5.6. Matriz de correlación

La implementación del Cálculo de ρ es la misma tanto en UDA como en SP, debido a esto, el tiempo de ejecución es el mismo en ambas implementaciones. El tiempo que toma calcular ρ es un tiempo despreciable comparado al tiempo que toma calcular n, L y Q y SVD (Véase Tabla 5.9). Más aún el tiempo que toma convertir ρ a un arreglo unidimensional también es despreciable en tanto el conjunto de datos tiene una dimensión máxima $d = 990$. Este tamaño puede ser contenido en memoria principal, por tanto los cálculos son rápidos aún siendo secuenciales.

d	Cálculo ρ	Conversión ρ a vector
10	0	0
50	0	0
100	0	0
200	0.002	0
400	0.0098	0.002
800	0.0366	0.0088

Tabla 5.9: Tiempo requerido para el cálculo de ρ y su conversión a un arreglo unidimensional.

5.7. Llamada al método SVD

Las dos versiones, FORTRAN LAPACK y el código Fuente C# son más lentas que la versión MKL como se muestra en la Tabla 5.10. Una razón es que inicialmente la biblioteca LAPACK se ejecuta en un solo hilo (LAPACK es *Single Threaded*), entonces no importa cuantos núcleos ofrezca el DBMS, la ejecución siempre se realizará en un solo hilo. La biblioteca MKL está optimizada para arquitecturas Intel©, además la inclusión de la biblioteca ScaLAPACK en MKL explota la paralización con múltiples núcleos, MKL es una biblioteca de álgebra lineal para computadoras con memoria distribuida en paralelo. ScaLAPACK resuelve sistemas lineales densos, problemas de valores propios, y los problemas de valores singulares.

d	Fortran Dual	C# Code Dual	MKL Dual	Fortran Quad	C# Code Quad	MKL Quad
2 - 20	0.00	0.00	0.00	0.00	0.00	0.00
32	0.00	0.01	0.00	0.00	0.01	0.00
64	0.02	0.04	0.00	0.02	0.04	0.00
128	0.11	0.20	0.02	0.11	0.20	0.02
256	0.92	1.44	0.16	0.91	1.44	0.11
512	7.60	10.53	1.16	7.64	10.54	0.53
990	35.32	46.81	4.40	35.33	46.67	1.77

Tabla 5.10: Cálculo de SVD con $n = 1M$ (Dual core vs Quad core.)

Basándose en los experimentos, es evidente que MKL realiza los cálculos mucho mejor, sin tomar en consideración los núcleos, entre las otras versiones. MKL está optimizada para la arquitectura Intel©.

La Tabla 5.11 muestra la importancia del cálculo de cada paso. Evidentemente la lectura del Conjunto de Datos $\mathbf{X}$ es el paso más tardado. Dado que el cálculo de las matrices de sumarización es en paralelo, parece difícil acelerar más su cómputo. Se observa que determinar las matrices de sumarización toma más del 90 % del tiempo global comparado con ρ y SVD.

d = 256	Dual	% Uso	Quad	% Uso
nLQ	456	99.96	216	99.95
Rho y FlatRho	0.005	0.00	0.005	0.00
SVD	0.162	0.04	0.113	0.05
Global	456.17	100.00	216.12	100.00

Tabla 5.11: Perfil de todos los cálculos con $n = 1M$ para calcular SVD con MKL y nLQ con UDA.

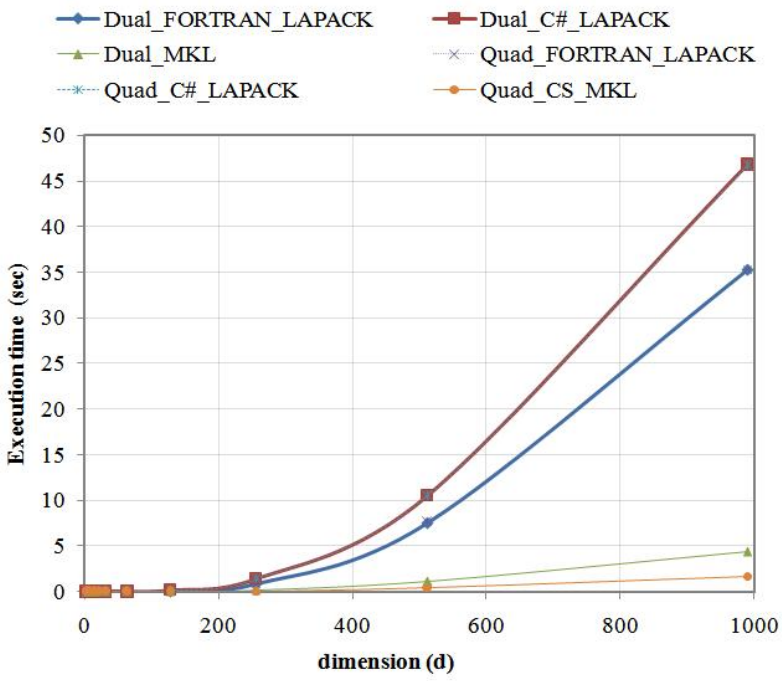

Figura 5.2: SVD con $n = 1M$ (Dual core vs Quad core).

A continuación se resume todos nuestros hallazgos, contribuciones y trabajo futuro en el Capítulo 6.

Capítulo 6

Conclusiones

Se mostró la factibilidad de utilizar la biblioteca de métodos numéricos LAPACK dentro de un SP. Se argumentó y mostró que llamar a la biblioteca LAPACK dentro de un DBMS provee un mejor desempeño comparado a la investigación previa de implementar el algoritmo PCA con consultas SQL y SP. Se mostró que la mejor manera de procesar SVD a partir de un conjunto de datos se realiza en tres fases: Calcular las matrices de sumarización del conjunto de datos, construir y reorganizar la matriz de correlación y llamar al método numérico que resuelve SVD disponible en la biblioteca LAPACK que recibe a la matriz ρ como parámetro de entrada. LAPACK posee un buen desempeño debido al uso de algoritmos basados en particiones por bloques. Estos algoritmos requieren que los datos sean manipulados como bloques, en lugar de vectores o escalares, aunque la cantidad de datos manipulada permanece sin cambios, la latencia (costo de Startup) asociado a esta manipulación por bloques es enormemente reducida, ya que es requerida una cantidad menor de mensajes para mover los datos . Esta ventaja en cálculos de Álgebra Lineal han existido desde antes, basándose en el estado del arte, no ha sido explotado por un DBMS. En general se obtuvieron buenos resultados al llamar a la biblioteca de métodos numéricos LAPACK dentro del DBMS. El procesamiento de matrices y la llamada a SVD son eficientes debido a que los cálculos se efectúan en memoria principal con algoritmos basados en particiones. ScaLAPACK provee de algoritmos basados en particiones, con la ventaja de trabajar en paralelo en CPS multi-núcleo. Gracias a este acoplamiento en arquitecturas Intel© multi-núcleo fue posible mostrar que ScaLAPACK provee de una Acelertación significativa en CPUs multi-núcleo debido al procesamiento en paralelo. El código pre-

compilado FORTRAN, diseñado para ejecutarse en un hilo, tiene un buen desempeño en CPUs con un solo núcleo. La solución de MKL con ScaLAPACK resuelve la descomposición SVD para una matriz de correlación con dimensión 800 en segundos. Se comparó tal desempeño con otras alternativas y se mostró que las consultas SQL son dos ordenes de magnitud mas lentas.

En resumen, se presentó una solución para integrar a la biblioteca LAPACK en un DBMS, utilizando UDFs, logrando una escalabilidad lineal en el tamaño del conjunto de datos $\mathbf{X}$, resolviendo PCA en matrices notablemente grandes, alcanzando casi una aceleración lineal en CPUs multi-núcleo aprovechando al máximo la velocidad del CPU.

6.1. Trabajo futuro

Los temas de investigación incluyen los siguientes. Desarrollar un sistema que permita llamar cualquier función de LAPACK utilizando UDFs (i.e. Mínimos Cuadrados, Factorización LU). Resolver el problema de Mínimos Cuadrados y resolver Ecuaciones Lineales con métodos de factorización es la investigación a seguir después de resolver SVD, ya que estos tres problemas son la base fundamental del Álgebra Lineal. Existe la hipótesis de que resolver estos tres problemas dentro del DBMS de manera eficiente mejorará el análisis de grandes volúmenes de datos. Otro punto a estudiar es identificar los requisitos del DBMS para manejar matrices enormes en LAPACK, especialmente cuando no caben en RAM.

La solución que utiliza código precompilado FORTRAN utiliza un enfoque de crear código C para llamar a la función escrita en FORTRAN, para esto es necesario crear un DLL especializado e importarlo en otra rutina en otra capa. Este enfoque es factible en el DBMS Microsoft SQL Server como se demostró en el capítulo 5. Aunque las UDFs están disponibles en los DBMS modernos, debe estudiarse la factibilidad, ventajas y desventajas de aplicar este enfoque a otros DBMS. Finalmente, es necesario investigar mecanismos para llamar LAPACK durante la lectura del conjunto de datos $\mathbf{X}$ a través de las UDFs o UDA para complementar o incluso hacer más eficiente la solución actual.

Apéndice A

Lista de Acrónimos

BLAS Basic Linear Algebra Subprograms 2

CLR Common Language Runtime 24

DBMS Data Base Management System 1

DBMSs Data Base Management Systems 19

DDL Data Definition Language 21

DLL Dynamic-link library 40

DML Data Manipulation Language 21

DR Dimensionality Reduction 6

DWH Data Warehouse 4

DM Data Mining 1

FORTRAN Formula Translating System 26

FLOPS Floating Point Operations 46

JIT Just-In-Time 25

KDD Knowledge Discovery Data in Databases 4

LAPACK Linear Algebra Package 2

MATLAB Matrix Laboratory 26

MKL Math Kernel Library 26

ML Machine Learning 4

OLAP On-Line Analytical Processing 28

PCA Principal Component Analysis 2

PPS Procedural Programming Systems 18

RDBMS Relational Data Base Management System 21

ScaLAPACK Scalable LAPACK 39

SP Stored Procedures 2

SQL Structured Query Language 1

SVD Singular Value Decomposition 1

TVF Table Valued Function 21

UCI University of California, Irvine Machine Learning repository 48

UDA User-defined Aggregates 2

UDTs User Defined Types 24

UDFs User Defined Functions 2

VDL View Definition language 21

Bibliografía

[1] S. Abiteboul, R. Hull, y V. Vianu. *Foundations of Databases : The Logical Level*. Pearson Education POD, facsimile ed$^{\underline{\acute{o}n}}$., 1994.

[2] Serge Abiteboul, Richard Hull, y Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. ISBN 0-201-53771-0.

[3] A. Acheson, M. Bendixen, J. A. Blakeley, P. Carlin, E. Ersan, J. Fang, X. Jiang, C. Kleinerman, B. Rathakrishnan, G. Schaller, B. Sezgin, R. Venkatesh, y H. Zhang. Hosting the .NET runtime in microsoft SQL server. págs. 860–865, 2004. doi: http://doi.acm.org/10.1145/1007568.1007669.

[4] J. Adibi, T. Barrett, S. Bhatt, H. Chalupsky, J. Chame, y M.W. Hall. Processing-in-memory technology for knowledge discovery algorithms. En *Proc. ACM DaMoN Workshop*, pág. Article 2. 2006.

[5] E. Agichtein y V. Ganti. Mining reference tables for automatic text segmentation. En *ACM KDD*, págs. 20–29. 2004.

[6] R. Agrawal, J. Gehrke, D. Gunopolos, y P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. En *ACM SIGMOD Conference*, págs. 94–105. 1998.

[7] R. Agrawal, T. Imielinski, y A. Swami. Mining association rules between sets of items in large databases. En *ACM SIGMOD Conference*, págs. 207–216. 1993.

[8] J. Albrecht, W. Hummer, W. Lehner, y L. Schlesinger. Query optimization by using derivability in a data warehouse environment. En *Proc. ACM DOLAP Workshop*. 2000.

[9] E. Anderson, Z. Bai, C. Bischof, S. Blackford J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, McKenney, et al. Lapack users'guide. society for industrial and applied mathematics, philadelphia, pa. Inf. téc., ISBN 0-89871-447-8 (paperback), 1999.

[10] D. Basu. On statistics independent of a complete sufficient statistic. *Selected Works of Debabrata Basu*, págs. 61–64, 2011.

[11] R. Bayardo y R. Agrawal. Mining the most interesting rules. En *ACM KDD Conference*, págs. 145–154. 1999.

[12] J.A. Blakeley, V. Rao, I. Kunen, A. Prout, M. Henaire, y C. Kleinerman. .NET database programmability and extensibility in Microsoft SQL Server. En *Proc. ACM SIGMOD Conference*, págs. 1087–1098. 2008.

[13] K. Bouil, S. Bimonte, H. Mahboubi, y F. Pinet. Towards the definition of spatial data warehouses integrity constraints with OCL. En *Proc. ACM DOLAP Workshop*. 2010.

[14] C. Boutsidis, W.M. Mahoney, y P. Drineas. Unsupervised feature selection for principal components analysis. En *Proc. SIGKDD*, págs. 61–69. ACM, New York, NY, USA, 2008. ISBN 978-1-60558-193-4. doi:http://doi.acm.org/10.1145/1401890.1401903.

[15] A. Buttari, J. Langou, J. Kurzak, y J. Dongarra. Parallel tiled QR factorization for multicore architectures. *Concurrency and Computation: Practice and Experience*, 20(13):1573–1590, 2008.

[16] A.M. Castaldo y R.C. Whaley. Scaling LAPACK panel operations using parallel cache assignment. En *ACM Sigplan Notices*, tomo 45, págs. 223–232. ACM, 2010.

[17] O.U. Celepcikay, C.F. Eick, y C. Ordonez. Regional Pattern Discovery in Georeferenced Datasets Using PCA. En *MLDM*, págs. 719–733. 2009.

[18] Z. Chen y C. Ordonez. Efficient OLAP with UDFs. En *Proc. ACM DOLAP Workshop*, págs. 41–48. 2008.

[19] S. Chitroub, A. Houacine, y B. Sansal. A new PCA-based method for data compression and enhancement of multi-frequency polarimetric SAR imagery. *Intelligent Data Analysis*, 6(2):187–207, 2002. ISSN 1088-467X.

[20] J. Cieslewicz, J. Berry, B. Hendrickson, y K.A. Ross. Realizing parallelism in database operations: insights from a massively multithreaded architecture. En *Proc. of ACM Workshop on Data Management on New Hardware (DaMoN)*, pág. 4. 2006.

[21] E.F. Codd. *The Relational Model for Database Management-Version 2*. Addison-Wesley, 1990.

[22] A. daspremont, F. Bach, y L. Ghaoui. Optimal solutions for sparse principal component analysis. *J. Mach. Learn. Res.*, 9:1269–1294, 2008. ISSN 1533-7928.

[23] C.J. Date. *An Introduction to Database Systems*. Addison Wesley, 2003.

[24] J. Demmel y K. Stanley. *The performance of finding eigenvalues and eigenvectors of dense symmetric matrices on distributed memory computers*. Computer Science Dep., 1994.

[25] J.W. Demmel. *Applied Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, 1997.

[26] J.W. Demmel, L. Grigori, M.F. Hoemmen, y J. Langou. Communication-optimal parallel and sequential qr and lu factorizations. Inf. Téc. 204, LAPACK Working Note, 2008. doi:http://www.netlib.org/lapack/lawnspdf/lawn204.pdf.

[27] C. Ding y X. He. K-means clustering via principal component analysis. En *Proc. ICML Conference*, pág. 29. 2004.

[28] L. Dobos, A. Szalay, J. Blakeley, T. Budavári, I. Csabai, D. Tomic, M. Milovanovic, M. Tintor, y A. Jovanovic. Array requirements for scientific applications and an implementation for microsoft SQL server. En *Proc. of ACM Workshop on Array Databases EDBT/ICDT*, págs. 13–19. 2011.

[29] D. Eddelbuettel. Benchmarking single-and multi-core BLAS implementations and GPUs for use with R. *Mathematica*, 2010.

[30] R. Elmasri y S.B. Navathe. *Fundamentals of Database Systems*. Addison-Wesley, 4th ed$^{\underline{\text{ón}}}$., 2003.

[31] U. Fayyad y G. Piateski-Shapiro. *From Data Mining to Knowledge Discovery*. MIT Press, 1995.

[32] U. Fayyad, G. Piatetsky-Shapiro, y P. Smyth. The KDD process for extracting useful knowledge from volumes of data. *Communications of the ACM*, 39(11):27–34, 1996.

[33] Usama Fayyad, Georges G. Grinstein, y Andreas Wierse. *Information Visualization in Data Mining and Knowledge Discovery*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001. ISBN 1558606890.

[34] K. Fernando y H. Nicholson. Karhunen-loeve expansion with reference to singular-value decomposition and separation of variables. En *Control Theory and Applications, IEE Proceedings D*, tomo 127, págs. 204–206. IET, 1980.

[35] Y. Fofanov, Y. Luo, C. Katili, J. Wang, Y. Belosludtsev, T. Powdrill, C. Belapurkar, V. Fofanov, T.B. Li, S. Chumakov, y B.M. Pettitt. How independent are the appearances of n-mers in different genomes? *Bioinformatics*, 20(15):2421–2428,

2004. doi:10.1093/bioinformatics/bth266. URL http://bioinformatics.oxfordjournals.org/cgi/content/abstract/20/15/2421.

[36] W.J. Frawley, G. Piatetsky-Shapiro, y C.J. Matheus. Knowledge discovery in databases: An overview. *AI Magazine*, (3):57, 1992.

[37] J.H. Friedman. Exploratory projection pursuit. *Journal of the American Statistical Association*, 82(397):249–266, 1987.

[38] H. Garcia-Molina, J.D. Ullman, y J. Widom. *Database Systems: The Complete Book*. Prentice Hall, 2001.

[39] J. Gehrke, Venkatesh Ganti, y R. Ramakrishnan. BOAT-optimistic decision tree construction. En *Proc. ACM SIGMOD Conference*, págs. 169–180. 1999.

[40] J.J. Gerbrands. On the relationships between SVD, KLT and PCA. *Pattern Recognition*, 14(1-6):375–381, 1981. doi:10.1016/0031-3203(81)90082-0. URL http://dx.doi.org/10.1016/0031-3203(81)90082-0.

[41] R. Ghani y C. Soares. Data mining for business applications: KDD-2006 workshop. *SIGKDD Explor. Newsl.*, 8(2):79–81, 2006. ISSN 1931-0145. doi:http://doi.acm.org/10.1145/1233321.1233332.

[42] G.H. Golub y C.F. Van Loan. *Matrix computations*, tomo 3. Johns Hopkins Univ. Press, 1996.

[43] G. Graefe, U. Fayyad, y S. Chaudhuri. On the efficient gathering of sufficient statistics for classification from large SQL databases. En *Proc. ACM KDD Conference*, págs. 204–208. 1998.

[44] J. Gray, A. Bosworth, A. Layman, y H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab and sub-total. En *ICDE Conference*, págs. 152–159. 1996.

[45] R.G Grimes y H.D. Simon. Solution of large, dense symmetric generalized eigenvalue problems using secondary storage. *ACM Transactions on Mathematical Software (TOMS)*, 14(3):241–256, 1988.

[46] M. Gu, J. Demmely, y I. Dhillonz. Efficient computation of the singular value decomposition with applications to least squares problems. Inf. téc., Technical Report CS-94-257, Department of Computer Science, University of Tennessee, Knoxville, TN, USA, 1994.

[47] J. Han y M. Kamber. *Data Mining: Concepts and Techniques (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, 2000. ISBN 1558604898. URL http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/1558604898.

[48] J. Han y M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, San Francisco, 1st ed$^{\underline{\text{ón}}}$., 2001.

[49] T. Hastie, R. Tibshirani, y J.H. Friedman. *The Elements of Statistical Learning*. Springer, New York, 1st ed$^{\underline{\text{ón}}}$., 2001.

[50] A. Hinneburg, D. Habich, y W. Lehner. Combi-operator-database support for data mining applications. En *Proc. VLDB Conference*, págs. 429–439. 2003.

[51] M. Hubert y S. Engelen. Robust pca and classification in biosciences. *Bioinformatics*, 20(11):1728–1736, 2004. ISSN 1367-4803. doi:http://dx.doi.org/10.1093/bioinformatics/bth158.

[52] Intel. Intel©Math Kernel Library Documentation. http://software.intel.com/en-us/articles/intel-math-kernel-library-documentation/, 2012.

[53] ISO-ANSI. *Database Language SQL-Part2: SQL/Foundation*. ANSI, 1999.

[54] M. Jaedicke y B. Mitschang. On parallel processing of aggregate and scalar functions in object-relational DBMS. En *ACM SIGMOD Conference*, págs. 379–389. 1998.

[55] I.T. Jolliffe. *Principal component analysis*, tomo 2. Wiley Online Library, 2002.

[56] ISO/ANSI JTC1/SC21. *Information Technology Database Languages SQL2*. ANSI, 1992.

[57] J. Kleinberg y T. Tomkins. Applications of linear algebra in information retrieval and hypertext analysis. En *ACM PODS*. 1999.

[58] V. Klema y A. Laub. The singular value decomposition: Its computation and some applications. *Automatic Control, IEEE Transactions on*, 25(2):164–176, 1980.

[59] A. J. Knobbe, A. Siebes, y B. Marseille. Involving aggregate functions in multi-relational search. En *PKDD02*, págs. 145–168. 2002.

[60] M. Kryszkiewicz. Mining with cover and extension operators. En *PKDD*, págs. 476–482. 2000.

[61] S. Kullback y R.A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[62] S. Lahabar y P.J. Narayanan. Singular value decomposition on GPU using CUDA. En *Proc. of IEEE International Symposium on Parallel & Distributed Processing*, págs. 1–10. 2009.

[63] N. Lawrence. Probabilistic non-linear principal component analysis with gaussian process latent variable models. *J. Mach. Learn. Res.*, 6:1783–1816, 2005. ISSN 1533-7928.

[64] C. Luo, H. Thakkar, H. Wang, y C. Zaniolo. A native extension of SQL for mining data streams. En *Proc. ACM SIGMOD Conference*, págs. 873–875. 2005. ISBN 1-59593-060-4. doi:http://doi.acm.org/10.1145/1066157.1066271.

[65] E. Malinowski y E. Zimányi. *Advanced Data Warehouse Design: From Conventional to Spatial and Temporal Applications*. Springer, 1st ed$^{\underline{\text{ón}}}$., 2008.

[66] S. Manegold, P.A. Boncz, y M.L. Kersten. Optimizing main-memory join on modern hardware. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 14(4):709–730, 2002.

[67] S. Mosci, L. Rosasco, y A. Verri. Dimensionality reduction and generalization. En *Proc. ICML*, págs. 657–664. ACM, New York, NY, USA, 2007. ISBN 978-1-59593-793-3. doi:http://doi.acm.org/10.1145/1273496.1273579.

[68] M. Navas y C. Ordonez. Efficient computation of PCA with SVD in SQL. En *KDD Workshop on Data Mining using Tensors and Matrices*, pág. Article 5. 2009.

[69] Microsoft Developer Network. CLR Hosted Environment. `http://msdn.microsoft.com/en-us/library/ms131047.aspx/`, 2012.

[70] A. Netz, S. Chaudhuri, J. Berhardt, y U. Fayyad. Integration of data mining with database technology. En *VLDB Conference*. 2000.

[71] A. Netz, S. Chaudhuri, U. Fayyad, y J. Berhardt. Integrating data mining with SQL databases: OLE DB for data mining. En *Proc. IEEE ICDE Conference*, págs. 379–387. 2001.

[72] C. Ordonez. Vertical and horizontal percentage aggregations. En *Proc. ACM SIGMOD Conference*, págs. 866–871. 2004.

[73] C. Ordonez. Building statistical models and scoring with UDFs. En *Proc. ACM SIGMOD Conference*, págs. 1005–1016. 2007.

[74] C. Ordonez. Models for association rules based on clustering and correlation. *Intelligent Data Analysis*, 13(2):337–358, 2009.

[75] C. Ordonez. Statistical model computation with UDFs. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 22(12):1752–1765, 2010.

[76] C. Ordonez y P. Cereghini. SQLEM: Fast clustering in SQL using the EM algorithm. En *Proc. ACM SIGMOD Conference*, págs. 559–570. 2000.

[77] C. Ordonez y J. García-García. Vector and matrix operations programmed with UDFs in a relational DBMS. En *Proc. ACM CIKM Conference*, págs. 503–512. 2006.

[78] C. Ordonez, N. Mohanam, C. Garcia-Alvarado, T. Predrag, y E. Martinez. Fast PCA computation in a DBMS with aggregate UDFs and LAPACK. En *Proceedings of the 21st ACM international conference on Information and knowledge management*, CIKM '12, págs. 2219–2223. ACM, New York, NY, USA, 2012. URL http://doi.acm.org/10.1145/2396761.2398605.

[79] C. Ordonez, M. Navas, y C. Garcia-Alvarado. Parallel multithreaded processing for data set summarization on multicore cpus. *Journal of Computing Science and Engineering*, 5(2):111–120, 2011.

[80] C. Ordonez y S. Pitchaimalai. Bayesian classifiers programmed in SQL. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 22(1):139–144, 2010.

[81] S.K. Pitchaimalai, C. Ordonez, y C. Garcia-Alvarado. Efficient distance computation using sql queries and udfs. En *Data Mining Workshops, 2008. ICDMW'08. IEEE International Conference on*, págs. 533–542. IEEE, 2008.

[82] W.H. Press, S.A. Teukolsky, W.T. Vetterling, y B.P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 2007. ISBN 0521880688, 9780521880688.

[83] C.R. Rao, E.J. Wegman, y J.L. Solka. *Handbook of Statistics, Volume 24: Data Mining and Data Visualization (Handbook of Statistics)*. North-Holland Publishing Co., 2005.

[84] W. Rinsurongkawong y C. Ordonez. Microarray data analysis with PCA in a DBMS. En *ACM DTMBIO*, págs. 13–20. 2008.

[85] K. Sattler y O. Dunemann. SQL database primitives for decision tree classifiers. En *Proc. ACM CIKM Conference*, págs. 379–386. 2001.

[86] A. Silberschatz, H.F. Korth, y S. Sudarshan. *Database System Concepts*. Mcgraw-Hill College, 1998.

[87] Z.H. Tang y J. MacLennan. *Data Mining with SQL Server 2005*. John Wiley & Sons, 2005. ISBN 0471462616.

[88] J. Verbeek. Mixture models for clustering and dimension reduction. 2004.

[89] Michael E. Wall, Andreas Rechtsteiner, y Luis M. Rocha. *Singular Value Decomposition and Principal Component Analysis*, cap. 5, págs. 91–109. Kluwel, Norwell, MA, 2003. URL http://adsabs.harvard.edu/cgi-bin/nph-bib_query?bibcode=2002physics...8101W.

[90] H. Wang y C. Zaniolo. User defined aggregates in object-relational systems. En *Proc. ICDE Conference*, págs. 135–144. 2000.

[91] H. Wang y C. Zaniolo. A native extension of SQL for data mining. En *Proc. SIAM SDM Conference*, págs. 130–144. 2003.

[92] H. Wang, C. Zaniolo, y C.R. Luo. ATLaS: A small but complete SQL extension for data mining and data streams. En *Proc. VLDB Conference*, págs. 1113–1116. 2003.

[93] I.H. Witten y E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kauffman, 2005.

[94] Y. Ye, K.A. Ross, y N. Vesdapunt. Scalable aggregation on multicore processors. En *Proc. of ACM Workshop on Data Management on New Hardware (DaMoN)*, págs. 1–9. 2011.

[95] A.L. Yuille, P. Stolorz, y J. Utans. Statistical physics, mixtures of distributions and the EM algorithm. *Neural Computation*, 6(1):334–340, 1994.

[96] Polyxeni Zacharouli, Michalis Titsias, y Michalis Vazirgiannis. Web page rank prediction with pca and em clustering. En *Proc WAW*, págs. 104–115. Springer-Verlag, Berlin, Heidelberg, 2009. ISBN 978-3-540-95994-6. doi:http://dx.doi.org/10.1007/978-3-540-95995-3_9.

[97] T. Zhang, R. Ramakrishnan, y M. Livny. BIRCH: An efficient data clustering method for very large databases. En *Proc. ACM SIGMOD Conference*, págs. 103–114. 1996.

[98] X.S. Zhuang y D.Q. Dai. Improved discriminate analysis for high-dimensional data and its application to face recognition. *Pattern Recogn.*, 40(5):1570–1578, 2007. ISSN 0031-3203. doi:http://dx.doi.org/10.1016/j.patcog.2006.11.015.