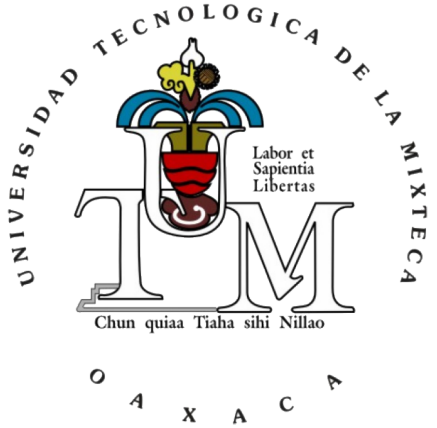

Universidad Tecnológica de la Mixteca

CÓMPUTO DE DISTANCIAS GEODÉSICAS PARA UN CONJUNTO DE DATOS GRANDE EN PROBLEMAS DE APRENDIZAJE AUTOMÁTICO

Tesis

para obtener el título de

Ingeniero en Computación

Presenta:

David de Jesús Bautista Villavicencio

Asesor:

Dr. Raúl Cruz Barbosa

Huajuapan de León, Oaxaca, México. Septiembre 2011

A mis padres

Agradecimientos

Primero, quiero agradecer a *Dios*, por todos y cada uno de los regalos que me ha brindado a lo largo de mi vida, especialmente por mis padres, mis hermanos, mis amigos, y por permitirme llegar a este día. *¡Gracias Dios!*

Agradezco a la Universidad Tecnológica de la Mixteca, por brindarme la oportunidad y el espacio para realizar mis estudios superiores.

Al proyecto de investigación PROMEP/103.5/10/5058, por el patrocinio parcial para la realización de esta investigación.

A mi director de tesis, el Dr. Raúl Cruz Barbosa, por su tiempo, atención, disposición, guía, consejos, paciencia, y por compartir conmigo su experiencia.

A los sinodales, la M.C. Verónica Rodríguez López, el Dr. Manuel Hernández Gutiérrez y el Dr. Aníbal Arias Aguilar, por su tiempo, consejos, interés y contribución al revisar este proyecto. Quiero agradecer especialmente a la Mtra. Vero por su disposición y verdadero compromiso en la revisión de esta tesis, a pesar de la distancia y las actividades que la ocupan.

A todos mis profesores, especialmente a aquellos que realizan su labor con verdadera entrega.

Gracias a todos aquellos que creyeron en mí, aquellos que me abrieron las puertas de su casa, los que me han ofrecido su amistad, aquellos con quienes he compartido momentos importantes de mi vida, aquellos que me han enseñado con su ejemplo, aquellos que me han dado palabras de aliento, aquellos que han estado pendientes de

mí, aquellos que han colaborado activamente para alcanzar este logro, a aquellos que me apoyaron durante este tiempo y a aquellos que me han apoyado desde siempre.

Gracias a mis padrinos Ma. Elena y Miguel, mis tíos Reyna y Miguel, mi hermana Betsabé y familia, mi hermano Israel, mi tío Rómulo, mi abuelo Isaías, mi tía Cristina, mi tía Adriana, mi tía Fátima y familia, mis tías Martina e Isabel, mis tías María y Gloria, mis primos: Miguel y David, Adriana y familia, Johnny y familia, Gabriel, Saúl y familia, Adela y familia, Rosario y Angélica; mi tío Salvador Maldonado y familia, Diego Maldonado, mi tía Victoria Velasco; mis amigos: Julio, Luis Noé, Carlos, Lissette, Rubi, Melina, Mayra, Daniel, Dámariz, Omar, Farisa, Gabriela, Mariana, Isaías, Andrea, Pamela, Abigail, Nidya, Montserrat, Magaly, Roberto, Abimael, Dalila, Eréndira, Noé, Lisset, Jessica y Miryam; la familia Paz Carreño, la familia Mendoza Ramírez, la familia Belmonte García; mis profesores y amigos: Dr. Víctor Escalante Jarero, L.F.M. Gustavo Jiménez Santana.

Finalmente, quiero agradecer con todo mi corazón y con toda mi alma, a mis padres.

Gracias mamá por la vida, gracias por tu *Amor*, gracias por creer en mí antes que nadie, gracias por tu sonrisa, gracias por ser el apoyo siempre fiel, gracias por escucharme, gracias por tus consejos y gracias por ser como eres. Gracias por brindarme tu hombro cuando he necesitado dónde llorar, y gracias por las risas y el tiempo que hemos compartido. Gracias por enseñarme mucho de lo que sé, gracias por sacar siempre lo mejor de mí. Gracias por preocuparte por mí, y claro, por ocuparte.

Gracias papá por tu entrega a nuestra familia, por el *Amor* con el que diariamente nos iluminas: a mi madre, a mis hermanos y a mí. Gracias por ser mi primer maestro, por mostrarme el mundo y saciar mi sed de respuestas. Gracias por enseñarme con tu gran ejemplo a ser un verdadero hombre. Gracias por apoyarme e impulsarme, gracias por acompañarme cuando tuve miedo, y gracias por enseñarme a ser valiente.

Gracias a ambos por mostrarme el camino, gracias por su gran ejemplo, por su tiempo, por su paciencia y por su *Amor*. Gracias por dar lo mejor de sí mismos día con día, y gracias por fortalecer las bases de nuestra familia. Gracias por su ejemplo como matrimonio y como padres. Gracias por darnos todo lo necesario, e incluso más. Gracias por nuestro tiempo juntos, y gracias por mis hermanos.

A todos ustedes, mi sincera y muy humilde gratitud. *¡Muchas gracias!*

David de Jesús Bautista Villavicencio
Julio, 2011

Índice general

Acrónimos

Aquí se listan los acrónimos empleados en esta tesis. Muchos de ellos se originan de palabras o frases en idioma inglés y se usaron del mismo modo debido a la simplicidad de su escritura.

APSP	Camino más corto entre todos los pares
CDG	Cómputo de distancias geodésicas
CDG^2	Cómputo de distancias geodésicas para conjuntos de datos grandes
COO	Lista coordenada
CSC	Columna dispersa comprimida
CSR	Renglón disperso comprimido
DOK	Diccionario de llaves
F-heaps	Montículos de Fibonacci
ISOMAP	*Isometric Feature Mapping*
KDD Archive	Archivo de descubrimiento de conocimiento en bases de datos
KNN	K-vecinos más cercanos
LDA	Análisis de Discriminantes Lineal
LIL	Lista de listas
MDS	Escalado Multidimensional
MMLL	*MMLL: Machine Learning Library*
MST	Árbol de expansión mínima
PCA	Análisis de Componentes Principales
RAM	Memoria de acceso aleatorio
RD	Reducción de la dimensionalidad
RDNL	Reducción de la dimensionalidad no lineal
RMN	Resonancia magnética nuclear
SSSP	Camino más corto para un único origen

Lista de publicaciones

Las publicaciones derivadas de esta tesis se listan a continuación:

1. Bautista-Villavicencio y Cruz-Barbosa (2011). On geodesic distance computation: An experimental study. 11^{th} *Mexican International Conference in Computer Science (ENC 2011), Advances in Computer Science and Applications, Research in Computer Science*, volumen 53, páginas 115−124, 2011.

2. Cruz-Barbosa, Bautista-Villavicencio, y Vellido (2011a). On the computation of the geodesic distance with an application to dimensionality reduction in a neuro-oncology problem. 16^{th} *Iberoamerican Congress on Pattern Recognition (CIARP 2011)*. Aceptado para publicación (Noviembre 2011).

3. Cruz-Barbosa, Bautista-Villavicencio, y Vellido (2011b). Comparative diagnostic accuracy of linear and nonlinear feature extraction methods in a neuro-oncology problem. *Pattern Recognition - 3^{rd} Mexican Conference in Pattern Recognition (MCPR 2011)*, volumen 6718 de *Lecture Notes in Computer Science*, páginas 34−41. Springer Berlin / Heidelberg, 2011.

La publicación 1 presenta los resultados experimentales del cómputo de las distancias geodésicas para diferentes conjuntos de datos, con el fin de encontrar la mejor combinación de representación de grafo y algoritmo de caminos más cortos. También se presenta una comparación de desempeño en tiempo entre dos implementaciones del mismo procedimiento, una escrita en C++ y otra usando el *software* Matlab.

En la publicación 2 se muestran los resultados obtenidos de la integración del procedimiento optimizado para el cómputo de las distancias geodésicas, en el método de reducción de la dimensionalidad no lineal ISOMAP. También, se incluye la comparación de algunos métodos de reducción de la dimensionalidad, en términos del

porcentaje de varianza explicada como función del número de características extraídas por cada método.

En la publicación 3 se presentan los resultados de exactitud de clasificación de espectros de tumores cerebrales a partir de las características extraídas por diferentes métodos de reducción de la dimensionalidad. Así mismo, se ofrece un panorama general de la inclusión del procedimiento optimizado para el cómputo de las distancias geodésicas en ISOMAP.

Capítulo 1

Introducción

El entendimiento del procesamiento de la información por parte del cerebro humano es uno de los grandes desafíos de la humanidad. Este obtiene información relevante a partir de grandes volúmenes de datos. Se sabe que el cerebro se comunica con aproximadamente 30 000 nervios auditivos, 10^6 nervios ópticos y otro número de orden mayor perteneciente a los demás nervios sensoriales, únicamente para percibir el mundo que nos rodea (Purves, 2007). A partir de los volúmenes de datos recibidos, el cerebro trabaja rápidamente en clasificarlos y discriminarlos, para tomar únicamente aquellos que le ofrecen conocimiento, mientras que los datos sin información relevante aparente, son ignorados. Todo esto es posible debido a diversas habilidades naturales, que pueden mejorar a través del aprendizaje. El cerebro debe realizar este procesamiento continuamente, puesto que se ve sometido a constantes ráfagas de datos provenientes de sus nervios sensoriales.

Cuando se pretende modelar y automatizar la inteligencia natural para crear entes inteligentes, nos enfrentamos a un problema realmente complejo, de interés para la inteligencia artificial. Así mismo, cuando el aprendizaje natural (propio de la inteligencia natural), se pretende llevar a las computadoras por medio del desarrollo de técnicas que permitan que las máquinas aprendan, se habla de aprendizaje automático (*machine learning*, en inglés).

Actualmente, dado el amplio uso de las computadoras en las diversas áreas del conocimiento, es natural que se quiera hacer uso de la tecnología disponible en la aplicación de métodos de aprendizaje automático.

En el aprendizaje automático, donde se estudian algoritmos que mejoran automáticamente con la experiencia (Russell y Norvig, 2003), el uso de la distancia geodésica

como métrica es fundamental.

La distancia geodésica es ampliamente usada como métrica en diversas aplicaciones. Algunos ejemplos son: gráficos por computadora, modelado de geometrías, segmentación de superficies, creación de mallas para figuras en 3D, clasificación de formas, imagenología médica, geofísica (Seong et al., 2008), visión computacional, emparejamiento de superficies 3D (Hua et al., 2008), parametrización y segmentación de mallas (Surazhsky et al., 2005), locomoción de robots y navegación por tierra (Mitchell et al., 1987). Asimismo es ampliamente usada en los campos de análisis de datos, minería de datos y aprendizaje automático (Lee y Verleysen, 2007).

En este proyecto se obtiene un procedimiento optimizado para el cómputo de las distancias geodésicas (aproximada por distancias de grafo), con lo que se inicia el desarrollo de una biblioteca de aprendizaje automático. El objetivo del procedimiento optimizado es servir de base para mejorar algoritmos de aprendizaje automático que empleen esta métrica. Esto satisface uno de los objetivos principales del proyecto de investigación intitulado "Mejoramiento de métodos de aprendizaje automático mediante distancias geodésicas" con identificador de proyecto PROMEP/103.5/10/5058, donde esta tesis es una de las metas de dicho proyecto.

1.1. Planteamiento del problema

El aprendizaje automático se puede categorizar principalmente en: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje semi-supervisado. En el aprendizaje supervisado se supone que existe un supervisor o maestro que indica si el aprendizaje de las características distintivas de un conjunto de elementos asociados a un tipo o clase es correcta. El objetivo final de este aprendizaje es que dado un elemento nuevo o desconocido (descrito por sus características respectivas) se le pueda asociar un tipo o la clase correspondiente. En contraste, en el aprendizaje no supervisado no se cuenta con un supervisor o maestro. Aquí, la información disponible son las características que describen a los elementos de un conjunto de datos, las cuales, si son numéricas, se pueden tratar como vectores en un espacio n-dimensional. El aprendizaje semi-supervisado es una combinación de los dos anteriores, donde se cuenta con pocos elementos con información de clase o tipo y muchos elementos sin dicha información.

Algunas de las tareas del aprendizaje no supervisado son el agrupamiento y

reducción de la dimensionalidad de los vectores correspondientes a un conjunto de datos. Aquí, las funciones de distancia tienen un papel muy importante. En tareas de agrupamiento, la distancia ayuda a definir los grupos, mediante la medición de la cercanía de los elementos del conjunto a elementos promedio o prototipo del grupo. En el caso de la reducción de la dimensionalidad, algunos métodos ocupan la distancia para encontrar la ruta más corta entre dos puntos contenidos en una variedad (*manifold*, en inglés).

La distancia Euclideana ha sido utilizada ampliamente en métodos de aprendizaje automático, debido a su fácil interpretación y entendimiento intuitivo en el mundo real. Además, la simplicidad de su cálculo la hace muy atractiva al momento de seleccionar una función de distancia. Sin embargo, se ha comprobado que para muchos conjuntos de datos con propiedades geométricas intrincadas (con muchos plegamientos) en el espacio de datos de entrada, la distancia Euclideana no es la adecuada, especialmente cuando se trabaja con elementos (o vectores) de un conjunto de observaciones que residen en un espacio de alta dimensionalidad, y no se cuenta con información adicional sobre la geometría de dicho conjunto de datos. Lo anterior motiva al modelamiento de dichos conjuntos de datos con una función de distancia alternativa.

Recientemente se ha demostrado que el uso de la distancia geodésica es más adecuado que la distancia Euclideana para calcular mediciones de (dis)similaridad en conjuntos de datos de alta dimensionalidad (Tenenbaum et al., 2000; Belkin y Niyogi, 2002; Silva y Tenenbaum, 2003). A diferencia de la distancia Euclideana, la distancia geodésica respeta la geometría de la variedad donde se encuentran los datos, es decir, la distancia geodésica mide la similaridad a lo largo de la variedad encajada (*embedded manifold*, en inglés), en lugar de hacerlo a través del espacio de encaje (*embedding space*, en inglés). De esta forma, se pueden evitar algunas de las distorsiones (tales como los rompimientos de preservación de topología) que el uso de una métrica estándar (como la distancia Euclideana), puede introducir cuando está aprendiendo la variedad, debido a su plegamiento excesivo (esto es, efectos de curvatura de la variedad indeseados) (Shi et al., 2006; Lee y Verleysen, 2007; Cruz-Barbosa y Vellido, 2008, 2010, 2011).

Es por esto que esta distancia es aprovechada y utilizada especialmente en métodos de reducción de la dimensionalidad, para el análisis de conjuntos de datos n-dimensionales (con superficies curvilíneas intrínsecas) como base para generar variedades de los mismos (Tenenbaum et al., 2000; Roweis y Saul, 2000; Lee et al., 2002; Lee y Verleysen, 2004, 2007; Cruz-Barbosa y Vellido, 2008; Yin, 2008), por

ejemplo, al trabajar con patrones climáticos globales, espectros estelares o el genoma humano, que son datos obtenidos del mundo real y con frecuencia difíciles de representar e interpretar debido a la alta dimensionalidad del espacio en que residen.

El cálculo de la distancia geodésica es intratable en términos computacionales, pero se ha demostrado que una buena aproximación se obtiene por medio de distancias de grafo (Bernstein et al., 2000). Así, en vez de encontrar el arco de longitud mínima entre dos puntos de la variedad, basta únicamente con encontrar el camino más corto entre los mismos puntos residentes en un grafo. Dicho grafo es construido al conectar los puntos sucesivos más cercanos. Los datos son los vértices, las conexiones permitidas son las aristas, y los pesos son las distancias Euclideanas entre dichos puntos. Como resultado se obtiene un grafo conexo, ponderado y no dirigido, al cual se le aplica repetidamente el algoritmo de Dijkstra para obtener los caminos más cortos entre todos los puntos.

Los principales problemas que se han encontrado al momento de realizar el cómputo de las distancias de grafo son problemas relacionados a limitaciones en tiempo y almacenamiento computacional. Por lo anterior, en el presente proyecto de tesis se pretende comparar distintas alternativas para el cálculo de dicha distancia y elegir la adecuada en función de los recursos computacionales con que se cuenta.

Posteriormente, se aplicará la alternativa escogida a un problema de reducción de la dimensionalidad.

1.2. Objetivos

La finalidad que se persigue en este trabajo de investigación se resume en los siguientes objetivos.

- Objetivo general
  - Diseñar e implementar un procedimiento que realice el cómputo de distancias geodésicas para un conjunto de datos grande.
- Objetivos específicos
  - Implementar un método para representar matrices dispersas.
  - Implementar un método para representar un conjunto de datos en forma de un grafo ponderado, conexo y no dirigido.

- Implementar el algoritmo de Dijkstra con cola de prioridad.
- Aplicación de la distancia geodésica a un problema de reducción de la dimensionalidad.

1.3. Organización de la tesis

Para un buen entendimiento de la tesis, los capítulos de esta se han distribuido de la siguiente manera.

En el capítulo 2, se presentan las definiciones de las distancias Euclideana, geodésica, de grafo y de las funciones de distancia en general, así como la forma de calcularlas. También, se describe el proceso y las técnicas para construir grafos conexos, ponderados y no dirigidos, a partir de un conjunto de datos numérico. Se incluyen algunas alternativas para representar grafos; así mismo se presentan métodos para obtener los valores y vectores propios de una matriz cuadrada y simétrica. También se proporciona la teoría referente a la reducción de la dimensionalidad mediante PCA e ISOMAP.

El desarrollo del proyecto se describe en el capítulo 3. Aquí, se incluyen las especificaciones del ambiente donde se desarrolló el proyecto y se ejecutaron las pruebas experimentales, se describen los conjuntos de datos empleados, y se detalla cada módulo necesario para el cómputo de las distancias geodésicas, y los módulos referentes a la aplicación de un problema de aprendizaje automático. Así mismo, se presenta una clasificación para los conjuntos de datos en función de su número de elementos.

En el capítulo 4, se presentan los resultados obtenidos, en los experimentos propuestos. Parte de estos resultados fueron publicados en Bautista-Villavicencio y Cruz-Barbosa (2011); Cruz-Barbosa et al. (2011a,b). Finalmente, se presentan las conclusiones y trabajo a futuro en el capítulo 5.

Los anexos A y B contienen los pseudocódigos de algunos de los algoritmos empleados y una descripción de los montículos de Fibonacci, respectivamente. El manual de usuario del *software* correspondiente al programa desarrollado, se encuentra en el anexo C. El anexo D es un glosario.

Capítulo 2

Fundamento teórico

En este capítulo, se presentan las definiciones de las funciones de distancia en general, así como los casos particulares de las distancias Euclideana, geodésica y de grafo. También, se describe el proceso y las técnicas para construir grafos conexos, ponderados y no dirigidos, a partir de un conjunto de datos numérico. Se incluyen algunas alternativas para representar grafos; así mismo, se presentan métodos para obtener los valores y vectores propios de una matriz cuadrada y simétrica, y métodos de reducción de la dimensionalidad.

2.1. Funciones de distancia

Diariamente se presentan situaciones donde es necesario conocer la posición entre dos objetos, o cuán cerca o lejos se encuentra uno del otro. Una referencia cuantificable para esto es la distancia.

Dependiendo del contexto, el concepto de distancia puede variar. La distancia puede referirse a la longitud física que separa dos objetos, o bien, a una estimación basada en otros criterios, como estar n posiciones por encima (o debajo) de una entrada específica en algún listado. En la teoría de grafos, la distancia entre dos vértices es la longitud del camino más corto entre dichos vértices.

En el sentido matemático, en el cual estamos interesados, una métrica, función de distancia, o simplemente distancia, es una función que se comporta de acuerdo con un conjunto específico de reglas, y provee una manera concreta de describir los significados de cercanía o lejanía, mediante un valor numérico, para un conjunto de elementos en un espacio determinado.

A continuación se presenta la definición formal de distancia, así como los tipos particulares de ésta que son de interés en este trabajo.

Para un conjunto no vacío X, suponemos una función real, no negativa $d(x, y)$ $(x, y \in X)$ definida en $X \times X$. Si dicha función $d : X \times X \longrightarrow \mathbb{R}^+ \cup \{0\}$ satisface las propiedades siguientes para cualquier elemento $x, y, z, \in X$, entonces es llamada una métrica o distancia para el conjunto X, y el par (X, d) es llamado un espacio métrico (Bryant, 1994; Bronshtein et al., 2005). Los axiomas de los espacios métricos son:

(I) No negatividad: $d(x, y) \geq 0$ y $d(x, y) = 0$ si y sólo si $x = y$.

(II) Simetría: $d(x, y) = d(y, x)$.

(III) Desigualdad del triángulo: $d(x, y) \leq d(x, z) + d(z, y)$.

Normalmente, cuando se habla de distancia entre dos puntos, uno suele referirse a la distancia más corta entre esos dos puntos. En el espacio Euclideano, la distancia más corta entre dos puntos equivale a la longitud de la línea recta que los une, expresada numéricamente. Si en vez de una línea recta, consideramos un arco o segmento de curva que una dichos puntos, y tomamos la menor, esa longitud es la denominada distancia geodésica.

2.1.1. Distancia Euclideana

La distancia Euclideana entre los puntos u y v es la longitud del segmento de línea recta que los une.

Sean $u = (x_1, x_2, \ldots, x_n)$ y $v = (y_1, y_2, \ldots, y_n) \in \mathbb{R}^n$, se define la distancia de Minkowski de orden p como:

$$\left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p} \tag{2.1}$$

Para el caso particular en que $p = 2$, se obtiene la distancia Euclideana.

$$d(u, v) = \left(\sum_{i=1}^{n} |x_i - y_i|^2 \right)^{1/2} = \sqrt{(x_1 - y_1)^2 + \ldots + (x_n - y_n)^2} \tag{2.2}$$

Cabe destacar que la distancia es una magnitud y que es diferente del desplazamiento, el cual es un vector.

2.1.2. Distancia geodésica

El término "geodésico" proviene de la palabra "geodesia", que es la ciencia que se encarga de las mediciones y representaciones de la superficie de la Tierra (Torge, 2001). En este contexto, la distancia geodésica es la longitud de la línea más corta que une dos puntos sobre la superficie de la Tierra siguiendo su forma curva elipsoidal (Koch, 2002).

En el contexto matemático, en el cual estamos interesados, se puede decir que la línea geodésica es una generalización del concepto de línea recta Euclideana llevada a superficies curvilíneas (Veblen y Whitehead, 1932); y viceversa, en el plano R^2 (como caso particular), la distancia geodésica coincide con la distancia Euclideana (Bommes y Kobbelt, 2007).

Contraria a la distancia Euclideana, la cual sólo depende de las coordenadas de dos puntos en el espacio, la distancia geodésica también depende de la variedad[1] en la cual se encuentran posicionados dichos puntos (ver figura 2.1) (Lee y Verleysen, 2004). Para calcular la distancia geodésica es necesario encontrar la longitud de arco mínimo entre dos elementos que se encuentran contenidos en una variedad.

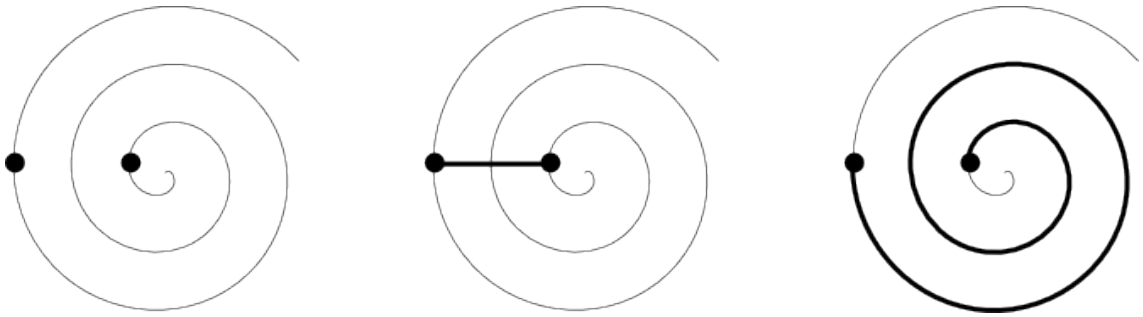

Figura 2.1: *Izq.* Dos puntos en una espiral. *Cen.* La distancia Euclideana entre dichos puntos. *Der.* La distancia geodésica. (Imagen tomada de Lee et al. (2002))

Formalmente, la distancia geodésica es bastante complicada de calcularse a partir de la expresión analítica de variedad (Lee y Verleysen, 2007). Consideremos dos puntos $\mathbf{x}_i$ y $\mathbf{x}_j$ en la variedad multidimensional $\mathcal{M}$, la cual depende de una variable latente t de baja dimensionalidad. La variedad $\mathcal{M}$ es parametrizada como sigue:

$$\mathbf{m} : \mathbb{R}^p \rightarrow \mathcal{M} \subset \mathbb{R}^d : \mathbf{t} \mapsto x = \mathbf{m}(t) \tag{2.3}$$

[1]Una variedad (*manifold*, en inglés), es un espacio matemático que en una escala suficientemente pequeña asemeja el espacio Euclideano de cierta dimensión. La relatividad general describe el espacio-tiempo como una variedad de cuatro dimensiones (Lee, 2000). Para más detalles véase el anexo D.

donde d es la dimensión del espacio de encaje y p ($\leq d$) es la dimensión de $\mathcal{M}$. Existen diferentes rutas que van del punto $\mathbf{x}_i$ al $\mathbf{x}_j$. Cada una de ellas es descrita por una sub-variedad unidimensional $\mathcal{P}_{i,j}$ de $\mathcal{M}$ con ecuaciones paramétricas:

$$\mathbf{p} : \mathbb{R} \to \mathcal{P}_{i,j} \subset \mathbb{R}^p : z \mapsto \mathbf{t} = \mathbf{p}(z) \tag{2.4}$$

La distancia geodésica entre $\mathbf{x}_i$ y $\mathbf{x}_j$ es definida como la longitud del arco mínimo que conecta ambos puntos:

$$l(\mathbf{x}_i, \mathbf{x}_j) = min_{\mathbf{p}(z)} \int_{z(i)}^{z(j)} \|\mathbf{J}_z \mathbf{m}(\mathbf{p}(z))\| \, dz \tag{2.5}$$

donde $\mathbf{J}_z\mathbf{m}(.)$ denota la matriz Jacobiana de $\mathbf{m}$ con respecto a z.

2.2. Distancia de grafo

Formalmente, la distancia geodésica es bastante complicada de calcularse a partir de la expresión analítica de variedad, además de que en la práctica, es intratable. Como consecuencia, el problema se reformula y la distancia geodésica es aproximada al computar la distancia de grafo (Bernstein et al., 2000).

En lugar de minimizar la longitud de un arco entre dos puntos de la variedad, ahora es necesario minimizar la longitud de una ruta o camino, del punto $\mathbf{y}_i$ al punto $\mathbf{y}_j$ que pase a través de cierto número de puntos $\mathbf{y}_k$, $\mathbf{y}_l, \ldots$ que residen en la variedad $\mathcal{M}$. La solución trivial a este problema es la distancia Euclideana, sin embargo, la idea es que el camino debería seguir la forma de la variedad, o al menos aproximarse (ver figura 2.2).

En la teoría de grafos, la longitud de los caminos más cortos es denominada distancia de grafo (Lee y Verleysen, 2007). En la medida que la distancia de grafo es una métrica, ésta debe cumplir los axiomas de los espacios métricos (véase §2.1), y por tanto es un requisito necesario que el grafo sea conexo, ponderado y no dirigido.

Dijkstra (1959) diseñó un algoritmo que resuelve el problema del camino más corto para un único origen (*SSSP*). En otras palabras, se calculan las longitudes de los caminos más cortos entre un nodo determinado (el origen) y todos los demás. El problema del camino más corto entre todos los pares (*APSP*), se resuelve fácilmente al aplicar repetidamente el algoritmo de Dijkstra, tomando cada nodo como nodo origen.

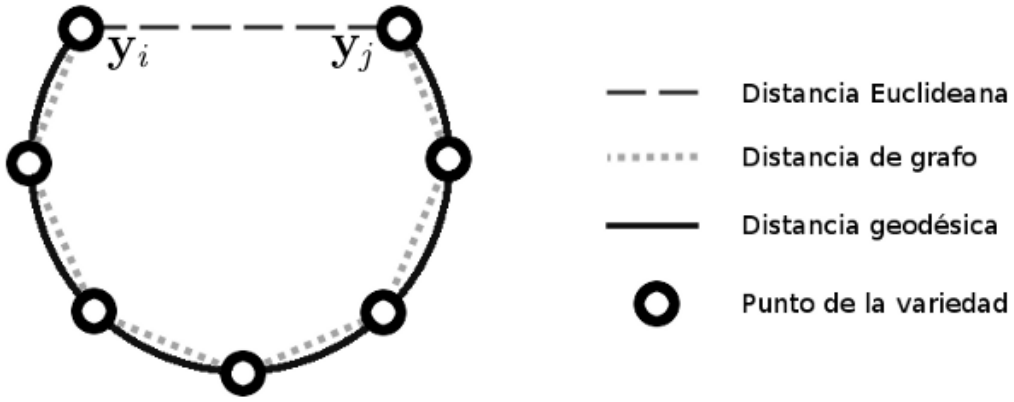

Figura 2.2: Con el fin de aproximar la distancia geodésica, se asocian los puntos de la variedad a los vértices del grafo, y se calcula la longitud del camino más corto entre cada par de vértices.

Los algoritmos que encuentran los caminos más cortos se basan en la propiedad de que el camino más corto entre un par de nodos, contiene otros subcaminos más cortos en él. Esta propiedad es utilizada tanto por los algoritmos basados en programación dinámica, como el de Floyd-Warshall, así como por los métodos ávidos, como el algoritmo de Dijkstra (Cormen et al., 1990).

2.2.1. Caminos más cortos

Existen problemas donde se desea encontrar el camino más corto entre dos puntos conectados directa o indirectamente, en un grafo. Formalmente, un grafo es la dupla $G = (V_n, E)$, donde V_n es un conjunto finito de n vértices o nodos v_i (los puntos de la variedad) y E el conjunto de aristas (las conexiones permitidas). Con el fin de calcular la longitud del camino, se deben asignar valores a las aristas. Como el valor dado es un atributo numérico, se dice que el grafo es ponderado. Si la longitud de la arista entre dos puntos, es la distancia Euclideana entre los mismos, entonces se dice que el grafo es Euclideano.

Un camino π en un grafo G es un subconjunto ordenado de vértices $[v_1, v_2, v_3, \ldots]$ tal que las aristas $(v_1, v_2), (v_2, v_3), \ldots$ pertenecen a E. Así, la longitud de π es definida

como la suma de las longitudes (o pesos) de las aristas que lo constituyen.

$$w(\pi) = \sum_{i=1}^{k} w(v_{i-1}, v_i) \tag{2.6}$$

Se define entonces la longitud del camino más corto de v_1 a v_n como:

$$\delta(v_1, v_n) = \begin{cases} min\{w(\pi) : v_1 \xrightarrow{\pi} v_n\} & \text{si hay un camino de } v_1 \text{ a } v_n, \\ \infty & \text{en otro caso} \end{cases} \tag{2.7}$$

Entonces, el camino más corto del vértice v_1 al vértice v_n está definido como cualquier camino π con longitud $w(\pi) = \delta(v_1, v_n)$.

Algoritmo de Dijkstra

Dijkstra (1959) diseñó un algoritmo que resuelve el problema del camino más corto para un único origen (*SSSP*), donde se desea conocer los caminos más cortos del nodo origen s, a todos los demás nodos del grafo. El algoritmo de Dijkstra mantiene un conjunto S de nodos (o vértices) para los cuales, el peso del camino más corto desde el nodo origen s, ha sido determinado. El algoritmo selecciona repetidamente el nodo $u \in V - S$ con el peso más pequeño (camino más corto), añade u a S y actualiza el peso de todas las aristas que se alcanzan desde u.

Como el algoritmo de Dijkstra siempre elige el nodo más cercano en $V - S$ para añadirlo a S, se dice que sigue una estrategia ávida (Cormen et al., 1990).

Para un grafo G, con vértices V y aristas E, la complejidad del algoritmo de Dijkstra es $O(|V^2| + |E|)$. La clave para implementar el algoritmo de manera eficiente, es hacer fácil la selección del nodo u con la distancia mínima $d[u]$, que será agregado al conjunto S[2]. Para realizar esta tarea, se puede hacer uso de una cola de prioridad. El algoritmo de Dijkstra tiene una complejidad de $O(|V| \log |V| + |E|)$ utilizando montículos de Fibonacci como cola de prioridad (Fredman y Tarjan, 1987; Cormen et al., 1990).

Los montículos de Fibonacci, o *F-heaps*, diseñados por Fredman y Tarjan (1987), son contenedores que permiten la inserción de elementos y la extracción del último elemento. Existen implementaciones que permiten decrementar algún elemento que se encuentre en el montículo, y algunas otras operaciones más. Las colas de prioridad

[2]Véase paso 5.*a* del algoritmo de Dijkstra en el anexo A.1

son usadas para mantener una lista dinámica de tareas con diferentes prioridades. La operación Insertar() agrega un nueva tarea a la cola. La operación Extraer-Menor(), extrae de la cola la tarea con la prioridad más alta. Si una tarea requiere un cambio súbito a una prioridad mayor, se emplea la operación Decrementar-Llave().

A continuación se muestra la complejidad de las operaciones básicas para los montículos binarios y de Fibonacci, según un análisis amortizado[3].

Operación	**Montículo binario**	**Montículo de Fibonacci**
Insertar()	$\Theta(\log n)$	$\Theta(1)$
Extraer-Menor()	$\Theta(\log n)$	$\Theta(\log n)$
Decrementar-Llave()	$\Theta(\log n)$	$\Theta(1)$

Cuadro 2.1: Comparativa entre las operaciones básicas para los montículos binarios y los montículos de Fibonacci. Θ representa la cota asintótica ajustada.

El algoritmo de Dijkstra, asigna la distancia de cada nodo a infinito (excepto para el nodo origen que es cero), y los inserta al montículo (o cola de prioridad). A continuación, se extrae del montículo el nodo menor u[4], y se examina cada nodo v que es adyacente a u. Si la distancia de u, $d[u]$, más la distancia de u a v, $w(u, v)$, es menor que la distancia actual de v, $d[v]$, entonces se actualiza la distancia de v, es decir, se decrementa el valor de la distancia de v, $d[v] \leftarrow d[u] + w(u, v)$[5]. La capacidad de decrementar un valor (o llave) almacenado en la cola de prioridad, es una propiedad esencial que permite que las distancias que son desconocidas, según vaya avanzando el algoritmo, se van actualizando, y al mismo tiempo se van descubriendo los caminos más cortos. El algoritmo finaliza cuando la cola está vacía, lo que significa que cada nodo fue visitado (Fredman y Tarjan, 1987; Cormen et al., 1990). Para más detalles sobre los F-heaps, véase el anexo B.1.

Cabe resaltar que el algoritmo de Dijkstra realiza una búsqueda en amplitud, y que al finalizar su ejecución, si consideramos el camino más corto como un árbol con raíz en el nodo origen, tendremos un árbol de expansión, y no cualquiera, sino el mínimo. Esto es debido a las similitudes entre los algoritmos de Prim (1957) y Dijkstra, ambos de naturaleza ávida.

[3]El cálculo de la eficiencia en tiempo, es realizado mediante un análisis amortizado. Para los montículos binarios, el tiempo refleja el tiempo total de la operación. Para los F-heaps, una llamada particular a alguna operación puede tomar un tiempo mayor que en otras ocasiones, esto es debido a que realiza trabajo de otras operaciones previas. Con el análisis amortizado, este trabajo adicional se adjudica a la operación que lo causa.

[4]Paso 5.*a*, en A.1

[5]Paso 5.*c*, en A.1

Para más detalles del algoritmo de Dijkstra, véase el anexo A.1.

Algoritmo de Floyd-Warshall

Se desea conocer los caminos más cortos entre todos los pares de nodos (vértices) de un grafo $G = (V, E)$ ponderado con una función $w : E \to \mathbb{R}$ que relacione los arcos con valores reales (pesos). Esto es, encontrar para cada par de vértices $u, v \in V$ un camino más corto (de menor peso), desde u hasta v, donde el peso del camino, es la suma de los pesos de los arcos que lo conforman. Para resolver este problema ($APSP$), tenemos dos opciones principales. La primera consiste en ejecutar un algoritmo SSSP en $|V|$ ocasiones, una con cada vértice como nodo origen. Y la segunda, que lo resuelve de una manera más directa, es aplicar el algoritmo de Floyd-Warshall. Este algoritmo tiene una complejidad de $\Theta(V^3)$, y puede trabajar incluso en presencia de arcos con pesos negativos, siempre y cuando éstos no formen ciclos. El algoritmo considera los vértices intermedios del camino más corto, donde un vértice intermedio de un camino simple $p = \langle v_1, v_2, \ldots, v_l \rangle$ es cualquier vértice de p distinto de v_1 o v_l, esto es, cualquier vértice del subconjunto $p = \langle v_2, v_3, \ldots, v_{l-1} \rangle$.

Bajo la suposición de que los vértices del grafo G son $V = \{1, 2, \ldots, n\}$, se considera un subconjunto $\{1, 2, \ldots, k\}$ para algún k. Para cada par de vértices $i, j \in V$, consideramos todos los caminos de i a j cuyos vértices intermedios estén contenidos en $\{1, 2, \ldots, k\}$, y sea p de entre todos los caminos, el de menor peso. El algoritmo de Floyd-Warshall explota la relación entre el camino p y los caminos más cortos entre i y j con todos sus vértices intermedios contenidos en el conjunto $\{1, 2, \ldots, k-1\}$. La relación depende de si k es un vértice intermedio del camino p, o no. Si no lo es, entonces todos los vértices intermedios del camino p están en el conjunto $\{1, 2, \ldots, k-1\}$. Así, el camino más corto del vértice i al j con todos los vértices intermedios en $\{1, 2, \ldots, k-1\}$, es también el camino más corto de i a j con todos los vértices intermedios en $\{1, 2, \ldots, k\}$. Pero si k es un vértice intermedio del camino p, entonces se debe descomponer p en sus subcaminos p_1 y p_2: $i \xrightarrow{p_1} k \xrightarrow{p_2} j$.

Basado en lo anterior, se define una formulación recursiva que calcula el camino más corto. Sea $d_{ij}^{(k)}$ el peso del camino más corto de i a j para los cuales todos los vértices intermedios están en $\{1, 2, \ldots, k\}$. Cuando $k = 0$, el camino del vértice i al j no incluye vértices intermedios. Tal camino tiene entonces, sólo un arco, siendo $d_{ij}^{(0)} = w_{ij}$. Una definición recursiva que contempla lo expuesto anteriormente, está

dada por:

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{si } k = 0 \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{si } k \geq 1 \end{cases} \tag{2.8}$$

Para más detalles del algoritmo, véase el anexo A.2.

2.3. Construcción de grafos

La construcción de grafos es una etapa esencial en este trabajo, ya que debe capturar las relaciones de vecindad entre los puntos del conjunto de datos original, de modo que se garantice la conservación de su estructura intrínseca. Así, el grafo se construye al conectar los puntos sucesivos más cercanos. Los datos son los vértices, las conexiones permitidas son las aristas, y los pesos son las distancias Euclideanas entre dichos puntos. Para determinar las conexiones permitidas, se hace uso de las reglas de construcción.

2.3.1. Reglas de construcción

Existen numerosas reglas inspiradas en ideas simples e intuitivas que permiten la construcción de grafos a partir de un conjunto de datos numérico visto como un conjunto de vectores en el espacio $\mathbb{R}^n$. Cada regla posee sus propias características y emplea diferentes tipos de información, dependiendo principalmente de los datos, o bien, de las preferencias o necesidades del usuario.

Regla-k

La idea principal de los modelos de vecinos más cercanos, es que las propiedades de cualquier punto de entrada x son similares a las de los puntos en la vecindad de x. Esta idea parece simple, hasta que es necesario especificar exactamente a qué se refiere uno con "vecindad". Si la vecindad es demasiado pequeña, no contendrá ningún punto; si es demasiado grande, podría incluir todos los puntos. Una solución consiste en definir la vecindad para ser lo suficientemente grande para incluir k puntos, donde k es lo suficientemente grande para asegurar un estimado significativo. Para un valor de k fijo, el tamaño de la vecindad varía (cuando los datos son dispersos, la vecindad es grande; cuando los datos son densos, la vecindad es pequeña) (Russell y Norvig, 2003). El algoritmo de los k-vecinos más cercanos (KNN) se basa en lo mencionado

anteriormente y conecta cada punto del conjunto de entrada con sus k vecinos más cercanos. El criterio de cercanía se basa en distancias Euclideanas, cuyo valor es asignado como el peso de cada conexión (o arista) entre un par de puntos (o nodos). Cada punto elige exactamente k vecinos, y puede ser elegido por otros puntos que no pertenezcan a su propio conjunto de vecinos (Lee y Verleysen, 2007); sin embargo, en todos los casos, la unión es simétrica. Es decir, la arista creada entre los puntos x e y es no dirigida, de manera equivalente, tenemos que $d(x, y) = d(y, x)$.

El éxito del procedimiento depende directamente del conjunto de datos de entrada y del parámetro k. El conjunto de datos (puntos) puede ser disperso o denso, y elegir el valor apropiado para k no es una tarea fácil.

Para valores muy grandes de k, la vecindad se torna demasiado grande y la estructura intrínseca del conjunto de datos se ve afectada, y puede incluso perderse (Russell y Norvig, 2003). Por el contrario, para valores muy pequeños de k, se podría obtener un conjunto de subgrafos, que habría que conectar apropiadamente de forma que no se distorsione la estructura original del conjunto, para obtener un único grafo que lo represente. Algunos trabajos de las disciplinas de reconocimiento de patrones, compresión vectorial, estadística computacional y minería de datos, han dedicado parte de sus esfuerzos en investigación para la elección "adecuada" del parámetro k (Shakhnarovich et al., 2006). Un resultado común ha sido el descubrimiento de que el valor "adecuado" de k, se determina experimentalmente usando los conjuntos de datos con que se realizarán las pruebas finales (Baoli et al., 2003). En contraste, en Chun-Guang et al. (2007) se propone una técnica para estabilizar este parámetro, la cual involucra resolver un problema de programación cuadrática. Los requerimientos computacionales para resolver dicho problema, deben ser tomados en cuenta, ya que agregarían una carga de tiempo adicional al módulo a desarrollar. Esta alternativa no es contemplada ya que la resolución de un problema de programación cuadrática queda fuera del ámbito de esta tesis.

Entre las ventajas de este algoritmo destacan, que es efectivo incluso si el conjunto de datos es grande y que es robusto a datos con ruido. Por el contrario, la principal desventaja es su costo computacional, que es bastante elevado debido a la necesidad de calcular la distancia de cada instancia a todas las restantes pertenecientes al conjunto de datos y su posterior ordenamiento.

Para más detalles del algoritmo, véase el anexo A.3.

Regla-ϵ

En comparación con la regla k, la regla ϵ trabaja casi de modo contrario. Cada punto x es conectado con todos los puntos que se encuentran contenidos dentro de una bola de radio ϵ y centrada en x. De este modo, todos los nodos (vecinos) cuya distancia es menor que ϵ, se conectan con el nodo de referencia que se encuentra en el centro, x. La vecindad de puntos de x en una bola de radio ϵ, $N_\epsilon(x)$, se define de la siguiente manera:

Sea $M = (X, d)$ un espacio métrico, $x \in X$, y $\epsilon \in \mathbb{R} : \epsilon > 0$, se tiene que: $N_\epsilon(x) = \{y \in X : d(x, y) < \epsilon\}$.

Determinar el valor de ϵ es incluso más difícil que elegir un valor de k adecuado para no distorsionar el conjunto de datos. Un valor muy pequeño de ϵ podría dejar aislados algunos nodos que se encuentren alejados del cúmulo principal. En contraparte, un valor muy grande podría contener a todo el conjunto (McCarty, 1988; Lee y Verleysen, 2007).

Existen reglas de construcción más sofisticadas (Lee y Verleysen, 2007), las cuales quedan fuera del ámbito de esta tesis.

2.3.2. Conexión de componentes

Existen diversos problemas en los que se desea realizar la interconexión de varios puntos. Para interconectar una serie de n puntos, podemos crear un arreglo de $n - 1$ enlaces, cada uno conectando dos puntos. De todos los arreglos posibles, uno es el que tiene el menor peso total, el cual es, en la mayoría de los casos, el más deseable.

Este tipo de problemas se pueden modelar mediante un grafo $G = (V, E)$ conexo y no dirigido, donde V es el conjunto de nodos o vértices, E es el conjunto de arcos o interconexiones entre pares de nodos, y para cada arco $(u, v) \in E$, tenemos un peso $w(u, v)$ que especifica el costo de conectar u y v. Ahora lo que queremos es encontrar un subconjunto no cíclico $T \subseteq E$, que conecte todos los vértices y cuyo peso total, denotado por la ecuación 2.9, sea minimizado.

$$w(T) = \sum_{(u,v)\in T} w(u, v) \tag{2.9}$$

Dado que T es acíclico y conecta todos los vértices, éste debe formar un árbol, el cual es llamado árbol de expansión, puesto que "expande" al grafo G.

Para hallar el árbol de expansión mínima[6] (MST), existen diversos algoritmos entre los que se encuentran los algoritmos de Prim y Kruskal.

Prim

El algoritmo de Prim (1957) encuentra el árbol de expansión mínima para un grafo conexo y ponderado.

El algoritmo de Prim funciona de manera similar al algoritmo de Dijkstra para encontrar los caminos más cortos en un grafo. Este algoritmo comienza desde un vértice arbitrario raíz r y crece hasta que el árbol expande a todos los vértices en V. En cada paso, el arco más "ligero" o de menor peso, es agregado al árbol A (que es un subconjunto de algún árbol de expansión mínima), que conecta a A con un vértice aislado de $G_A = (V, A)$. Cuando el algoritmo termina, los arcos de A forman un árbol de expansión mínima (Cormen et al., 1990).

La clave para implementar el algoritmo de Prim de manera eficiente es hacer fácil la selección del arco que será agregado para formar el árbol A. Durante la ejecución del algoritmo, todos los vértices que no pertenecen al árbol permanecen en una cola de prioridad Q basada en un campo llave k. Para cada vértice v, $key[v]$ es el peso mínimo de cualquier arco que conecta a v con un vértice del árbol; por convención $key[v] = \infty$ si no existe tal arco. Cada vértice conoce a su padre en el árbol por el atributo $\pi[v]$, que nombra al padre de v en el árbol. Si $v = r$ o v no tiene padre, entonces $\pi[v] = \text{NULL}$.

Mientras el algoritmo avanza, $A = \{(v, \pi[v]) : v \in V - \{r\} - Q\}$; y cuando éste llega a su fin, la cola de prioridad está vacía, $V_A = V \Rightarrow Q = \emptyset$, por lo que el árbol de expansión mínima A de G es entonces:

$$A = \{(v, \pi[v]) : v \in V - \{r\}\}. \tag{2.10}$$

Para más detalles del algoritmo, véase el anexo A.4.

[6] La frase "árbol de expansión mínima" (*minimum spanning tree*, en inglés) es una forma abreviada de la expresión "árbol de expansión con el mínimo peso", que refiere al hecho de que no se está minimizando el número de arcos en T, sino únicamente el peso total del mismo. El algoritmo devuelve de todos los árboles de expansión, aquel con la expansión mínima.

Kruskal

El algoritmo de Kruskal resuelve el problema del árbol de expansión mínima, por medio de una estrategia ávida. El algoritmo pretende hallar el arco más seguro para agregar al bosque en crecimiento, buscándolo entre todos los arcos que conecten cualesquiera dos árboles en el bosque, el arco (u, v) de mínimo peso. Emplea una estructura de datos para mantener numerosos elementos pertenecientes a conjuntos disjuntos. Cada conjunto contiene los vértices de un árbol del bosque actual. Un método con parámetro de entrada u, nos devuelve un elemento representativo del conjunto al que pertenece el mismo u, de este modo, podemos determinar si dos vértices u y v pertenecen al mismo árbol al invocar dicho método para u, luego para v y comparar los nodos resultantes. Finalmente se realiza la unión de árboles.

2.4. Representación de grafos

Existen dos maneras principales de representar un grafo $G = (V, E)$, como una colección de listas de adyacencia o como una matriz de adyacencia. Ambas representaciones se pueden aplicar tanto a grafos dirigidos, como a grafos no dirigidos. Sin embargo, la elección del formato para representar el grafo, depende directamente del grafo en cuestión. Generalmente, se opta por representar el grafo mediante una lista de adyacencia, si éste es disperso (aquel donde $|E|$ es mucho menor que $|V|^2$), ya que provee una manera compacta de representarlo. Por el contrario, si el grafo es denso ($|E|$ es cercano a $|V|^2$) o es de gran importancia la rapidez con que se pueda determinar si existe un arco entre un par de nodos dado, se elige representarlo haciendo uso de una matriz de adyacencia.

2.4.1. Lista de adyacencia

La representación mediante lista de adyacencia de un grafo $G = (V, E)$ consiste en un *arreglo* $\mathsf{Adj}[\,]$ de $|V|$ listas, una por cada nodo en V. Para cada $u \in V$, la lista de adyacencia $\mathsf{Adj}[u]$ contiene todos los vértices v para los cuales existe el arco $(u, v) \in E$. Esto es, $\mathsf{Adj}[u]$ contiene todos los nodos que son adyacentes a u en G. Comúnmente, los vértices en cada lista de adyacencia están ordenados arbitrariamente.

Una desventaja potencial de la representación de grafos mediante listas de adyacencia, es que no hay otra forma más rápida de determinar si un arco (u, v)

existe en el grafo, que buscar a v en la lista de adyacencia $\mathsf{Adj}[u]$ (Cormen et al., 1990).

2.4.2. Matriz de adyacencia

Para la representación de un grafo $G = (V, E)$ en una matriz de adyacencia, se asume que los vértices se encuentran enumerados $1, 2, \ldots, |V|$ en un orden arbitrario. Así, la representación del grafo por medio de una matriz de adyacencia, consiste en una matriz $\mathsf{A} = (a_{ij})$, tal que:

$$a_{ij} = \begin{cases} 1 & \text{si } (i,j) \in E, \\ 0 & \text{en otro caso.} \end{cases} \tag{2.11}$$

La matriz de adyacencia de un grafo requiere $\Theta(V^2)$ de almacenamiento, independientemente del número de arcos en el grafo (Cormen et al., 1990).

2.4.3. Matriz dispersa

Una matriz que tiene sólo un pequeño porcentaje de entradas diferentes de cero, se dice que es dispersa. Las matrices dispersas se emplean con frecuencia en aplicaciones como: programación lineal, análisis estructural, teoría de redes, ecuaciones diferenciales, métodos numéricos, teoría de grafos, teoría genética, ciencias sociales y programación de computadoras.

Es común que se presenten problemas interesantes e importantes que no pueden ser resueltos porque emplean grandes matrices, las cuales pueden ser difíciles, o incluso imposibles, de almacenar en la memoria de una computadora. Ya que muchas de esas matrices son dispersas, es útil conocer las técnicas disponibles para manejar dichas matrices, con el fin de elegir la que mejor se adapte para el tipo de matriz en cuestión (Tewarson, 1973).

Generalmente, las matrices dispersas grandes son almacenadas en forma "empaquetada", es decir, sólo se almacenan los valores distintos a cero, así como la información necesaria para indexarlos. Diversas son las razones para utilizar este esquema de almacenamiento. Primero, matrices grandes pueden ser almacenadas y manipuladas en la memoria de la computadora. Segundo, hay casos en los que aún cuando la matriz se encuentra en forma empaquetada no puede ser almacenada en la memoria interna y se puede emplear memoria externa como auxiliar. Generalmente, obtener información de

la memoria externa es mucho más lento que obtenerla del almacenamiento interno, por lo cual, es una ventaja que los datos que deban viajar sean los menos posibles. Tercero, hay un ahorro sustancial de tiempo si se evita realizar cálculos sobre las entradas con valor cero, esto se hace al procesar únicamente las operaciones no triviales.

Existen varios esquemas de almacenamiento empaquetado disponibles, dichos formatos se pueden dividir convenientemente en dos grupos: aquellos que soportan modificaciones de forma eficiente, y aquellos que soportan las operaciones matriciales de forma eficiente. El grupo de modificación eficiente lo conforman los diccionarios de llaves, listas de listas y listas coordinadas, los cuales son usados normalmente para construir la matriz. Una vez que la matriz se ha construido, generalmente se convierten a un formato que soporte modificaciones de forma eficiente. A continuación se listan y describen brevemente dichos formatos.

- Diccionario de llaves (DOK, del inglés *dictionary of keys*), representa los valores distintos de cero como un diccionario que relaciona las duplas (renglón, columna), con sus valores.

- Lista de listas (LIL, del inglés *list of lists*), almacena una lista por renglón, donde cada entrada almacena un índice columna y un valor.

- Lista coordenada (COO, del inglés *coordinate list*), almacena una lista de tuplas (renglón, columna, valor).

Estos formatos son buenos para construir de forma incremental una matriz dispersa, pero su rendimiento es pobre para acceder al contenido. Comúnmente se crea la matriz en alguno de estos formatos y luego se convierte a otro para operar. Éstos pueden ser:

- Formato Yale, almacena la matriz dispersa inicial M ($m \times n$), en forma de renglón usando tres arreglos. Sea NNZ el número de entradas no cero en M. El primer arreglo es A[ ], el cual tiene una longitud NNZ, y almacena todas las entradas no cero de M, en orden izquierda-derecha y arriba-abajo. El segundo arreglo es IA[ ], de longitud $m + 1$. IA[i] contiene el índice en A del primer elemento no cero en el renglón i. El renglón i de la matriz original se deriva de A[IA[i]] a A[IA[i + 1] - 1]. El tercer arreglo, JA[ ], contiene el índice columna de cada elemento de A[ ], y es de longitud NNZ.

- Renglón disperso comprimido (CSR, del inglés *compressed sparse row*), es idéntico al formato de matriz dispersa Yale, excepto que el arreglo columna

se almacena normalmente por delante del arreglo renglón. Es decir, CSR es (val, col_idx, row_ptr), donde val es un arreglo con los valores no cero de la matriz, en orden izquierda-derecha y arriba-abajo, col_idx son los índices columna correspondientes a los valores, y row_ptr es la lista con los índices donde comienza cada renglón.

- Columna dispersa comprimida (CSC, del inglés *compressed sparse column*), es similar a CSR, salvo que los valores son leídos primero por columna, un renglón índice es almacenado por cada valor, y se almacenan apuntadores columna. CSC es (val, row_idx, col_ptr), donde val es un arreglo con los valores no cero de la matriz, en orden arriba-abajo y enseguida izquierda-derecha, row_idx son los índices renglón correspondientes a los valores, y col_ptr es la lista de índices donde comienza cada columna.

2.5. Valores y vectores propios

En álgebra lineal, la descomposición o factorización de una matriz en formas canónicas, conlleva a representar dicha matriz en términos de sus valores y vectores propios. Los vectores propios (*eigenvectors*) de una matriz cuadrada, son los vectores distintos de cero que después de ser multiplicados por la matriz, se mantienen proporcionales al vector original, es decir, cambian únicamente de magnitud, pero no de dirección. A cada vector propio, le corresponde un valor propio (*eigenvalue*), denotado por la letra griega λ, que es el factor por el cual el vector propio cambia cuando es multiplicado por la matriz.

De manera formal, sea $T : V \to V$ una transformación lineal. Se desea encontrar un vector $\vec{\mathbf{v}} \in V$, tal que $T\vec{\mathbf{v}}$ y $\vec{\mathbf{v}}$ sean paralelos. Es decir, se busca un vector $\vec{\mathbf{v}}$ y un escalar λ, tal que:

$$T\vec{\mathbf{v}} = \lambda\vec{\mathbf{v}} \tag{2.12}$$

Si $\vec{\mathbf{v}} \neq 0$ y λ satisface la ecuación 2.12, entonces λ se llama un valor propio de T, y $\vec{\mathbf{v}}$ se llama un vector propio de T correspondiente al valor propio λ (Grossman, 1996).

Para hallar los valores y vectores propios de una matriz cuadrada (A de $n \times n$), se tienen varias opciones. Aquí se presentan dos, especiales para el caso en el que la matriz además de ser cuadrada, es simétrica ($A = A^T$). Uno destaca por su simplicidad, y

el otro por su eficiencia. El principio de funcionamiento de ambos métodos, es llevar la matriz a una forma más simple, y a partir de ésta, obtener los valores y vectores deseados.

2.5.1. Método de Jacobi

El método de Jacobi consiste en realizar una serie de transformaciones sucesivas, de tal forma que con cada transformación (que es una rotación del plano), se anule uno de los elementos de la matriz fuera de la diagonal. Después de sucesivas iteraciones, la matriz se vuelve diagonal. Acumular el producto de las transformaciones, permite obtener la matriz de vectores propios, mientras que los elementos que conforman la diagonal final, son los valores propios.

Éste método realiza una secuencia de transformaciones $A \leftarrow Q^T AQ$, con la propiedad de que cada nueva A, aunque aún es una matriz completa, es más diagonal que su predecesora. Eventualmente, las entradas que no se encuentran en la diagonal, serán lo suficientemente pequeñas para ser consideradas cero.

La idea detrás del método de Jacobi es reducir sistemáticamente la cantidad:

$$off(A) = \sqrt{\sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij}^2} \tag{2.13}$$

es decir, la "norma" de los elementos fuera de la diagonal. Las herramientas para lograr esto son las rotaciones de la forma:

$$J(p,q,\theta) = \begin{bmatrix} 1 & \dots & 0 & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \dots & c & \dots & s & \dots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \dots & -s & \dots & c & \dots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \dots & 0 & \dots & 1 \end{bmatrix} \begin{matrix} \\ \\ p \\ \\ q \\ \\ \\ \end{matrix} \tag{2.14}$$
$$\qquad\qquad\qquad\quad p \qquad\quad q$$

que son denominadas rotaciones de Jacobi.

El paso básico en el procedimiento de Jacobi para obtener los valores propios

involucra, primero, elegir un par de índices (p, q) que satisfagan $1 \leq p < q \leq n$, segundo, calcular un par coseno-seno (c, s) tal que:

$$\begin{bmatrix} b_{pp} & b_{pq} \\ b_{qp} & b_{qq} \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} a_{pp} & a_{pq} \\ a_{qp} & a_{qq} \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \tag{2.15}$$

es diagonal, y tercero, sobrescribir A con $B = J^T A J$, donde $J = J(p, q, \theta)$. Así, la matriz B coincide con A, excepto en los renglones y columnas p y q, por ello, A se acerca cada vez más a la forma diagonal, con cada iteración del paso básico de Jacobi.

Eventualmente, se obtiene una matriz D que es diagonal en términos de precisión máquina (*machine precision*, en inglés). Los elementos de la diagonal principal son los valores propios de la matriz original A, ya que $D = V^T A V$, donde $V = P_1 P_2 P_3 \ldots$, y P_i son las matrices sucesivas de rotación de Jacobi. Las columnas de V son los vectores propios, puesto que $AV = VD$.

Finalmente, la estrategia a seguir para seleccionar los elementos que serán anulados, es hacerlo en estricto orden. Por ejemplo, hacia abajo por renglones, y no buscando en toda la matriz inferior al elemento mayor, como sugiere el algoritmo original propuesto en 1846 (Golub y Loan, 1996; Grossman, 1996; Press et al., 2007).

2.5.2. Algoritmo QR

Una estrategia eficiente en tiempo, para encontrar los valores y vectores propios, consiste en, primero, reducir la matriz a una forma simple, y en seguida comenzar un procedimiento iterativo. Para matrices simétricas, la forma simple es la tridiagonal.

El algoritmo QR emplea para ello, el método de Householder y enseguida, la descomposición QR. Ambos métodos se describen a continuación.

Método de Householder

El algoritmo de Householder, reduce una matriz simétrica A_1 $(n \times n)$ a la forma tridiagonal A_{n-1} en $n - 2$ transformaciones ortogonales. Cada transformación anula la parte requerida de una columna completa y su correspondiente renglón completo. Las matrices A_i, se definen con las relaciones:

$$A_{i+1} = P_i \, A_i \, P_i \qquad (i = 1, 2, \ldots, n-2) \tag{2.16}$$

donde P_i son las matrices ortogonales definidas por:

$$\begin{gathered} P_i = I - 2w_i w_i^T \\ w_i^T = [w_{i,1}; w_{i,2}; \ldots; w_{i,n-i}; 0; 0; \ldots; 0] \\ w_i^T w_i = 1 \end{gathered} \tag{2.17}$$

Se asume que A_i es de forma tridiagonal en sus últimos $(i-1)$ renglones y columnas, y se eligen los elementos de w_i para que A_{i+1} sea de forma tridiagonal en sus últimos i renglones y columnas. Es evidente, a partir de las formas asumidas de A_i y de w_i, que la transformación con la matriz P_i, deja los últimos $(i-1)$ renglones y columnas sin ser alterados, y así, w_i tenga que ser elegido para que $(l,1),(l,2),\ldots,(l,l-2)$ elementos de A_{i+1} sean cero $(l = n - i + 1)$. Es claro que todas las matrices A_i, son simétricas.

Supóngase que las matrices de Householder $P_1, \ldots, P_{k-1}$ han sido determinadas de tal forma que si $A_{k-1} = (P_1, \ldots, P_{k-1})^T A (P_1, \ldots, P_{k-1})$, entonces:

$$A_{k-1} = \begin{matrix} \\ \\ \\ \end{matrix} \begin{matrix} \begin{bmatrix} B_{11} & B_{12} & 0 \\ B_{21} & B_{22} & B_{23} \\ 0 & B_{32} & B_{33} \end{bmatrix} \\ {\scriptstyle k-1 \quad 1 \quad n-k} \end{matrix} \begin{matrix} {\scriptstyle k-1} \\ {\scriptstyle 1} \\ {\scriptstyle n-k} \end{matrix} \tag{2.18}$$

es tridiagonal en sus primeras $(k-1)$ columnas. Es importante explotar la simetría en el cálculo de las nuevas matrices (Wilkinson, 1962; Golub y Loan, 1996; Press et al., 2007).

Algoritmo QL con desplazamientos implícitos

El algoritmo QL con desplazamientos implícitos (*QL Algorithm with implicit shifts*, en inglés), determina los valores propios, y opcionalmente los correspondientes vectores propios, de una matriz tridiagonal simétrica y real. Así mismo, es matemáticamente equivalente al algoritmo QL, aunque no realiza explícitamente algunas operaciones que el último sí hace, lo que le permite tener un desempeño más eficiente (Golub y Loan, 1996).

El algoritmo se basa en lo siguiente: si $T = QR$, es la factorización QR de una matriz simétrica tridiagonal $T \in \mathbb{R}^{n \times n}$, entonces:

$$T_+ = R\,Q = Q^T (Q\,R)\,Q = Q^T\,T\,Q \tag{2.19}$$

también es simétrica y tridiagonal. Por otro lado, si $s \in \mathbb{R}$ y $T - sI = QR$, es la factorización QR, entonces:

$$T_+ = RQ + sI = Q^T T Q \tag{2.20}$$

también es tridiagonal. Esto es denominado el paso QR desplazado. Así mismo, si T es irreducible, entonces las primeras $n-1$ columnas de $T - sI$ son independientes sin considerar s. Así, si $s \in \lambda(T)$ y

$$Q\,R = T - sI \tag{2.21}$$

es una factorización, entonces $r_{nn} = 0$, y la última columna de $T_+ = RQ + sI = sI_n(:, n)$.

2.6. Reducción de la dimensionalidad

Los métodos de reducción de la dimensionalidad (RD) son herramientas innovadoras e importantes en diversos campos de estudio, como el análisis de datos, minería de datos y aprendizaje automático. Estos métodos proveen una manera de visualizar en un espacio de baja dimensionalidad, la estructura de conjuntos de datos de alta dimensionalidad. Existen diversas formas de categorizar los métodos de RD dependiendo de factores como: propósito del método, modelo subyacente o criterio matemático a ser optimizado. Una forma en la cual se realiza la RD es a través de métodos de extracción de características, los cuales se pueden clasificar como lineales y no lineales. Uno de los métodos lineales mejor conocido y ampliamente utilizado, es el Análisis de Componentes Principales (PCA, del inglés *Principal Component Analysis*). Su objetivo principal es reducir la dimensionalidad de un conjunto de alta dimensionalidad, por medio de una transformación ortogonal, mientras retiene la varianza de los datos (tanto como sea posible) a lo largo de las dimensiones (componentes) principales extraídas (Jolliffe, 2002). Sin embargo, los métodos tradicionales como PCA y Escalado Multidimensional (MDS, del inglés *Multidimensional Scaling*), presentan desventajas frente a ciertos conjuntos de datos, por estar basados en modelos lineales; mientras que los métodos nuevos como *Isometric Feature Mapping* (*ISOMAP*) y *Locally Linear Embedding* (*LLE*) son capaces de reducir la dimensionalidad en un modo no lineal (RDNL).

ISOMAP es un método reciente para la RDNL. Este es una variante de MDS (Kruskal, 1964a,b), cuyo propósito es encajar (*embed*, en inglés) puntos de alta dimensionalidad en un espacio de baja dimensionalidad, procurando preservar la distancia entre puntos tan cercana como sea posible (Tenenbaum et al., 2000).

Muchos de los métodos nuevos se desarrollaron hacia el año 2000, convirtiendo la reducción de la dimensionalidad no lineal en un tópico de suma importancia. Algunos avances que permitieron su rápido crecimiento y desarrollo fueron por ejemplo, el uso de grafos para representar la topología de la variedad y el uso de nuevas métricas como la distancia geodésica. El desarrollo y mejoramiento de métodos de RDNL se ha vuelto relevante, debido a que intentan descubrir la geometría intrínseca de los datos. El principio general de estos métodos, particularmente los que utilizan la distancia geodésica, es el de considerar al conjunto de datos original como muestras discretas de una variedad. A partir de ello, se construye una solución aproximada de la métrica de la variedad que permite extraer las dimensiones más significativas del conjunto. Con este procedimiento, se obtienen las estructuras de baja dimensión existentes en los espacios de alta dimensionalidad.

2.6.1. PCA

El análisis de componentes principales o PCA, identifica patrones en un conjunto de datos y expresa dichos datos de tal manera que se resalten sus diferencias y similitudes.

PCA es un procedimiento matemático que mediante una transformación lineal, convierte un conjunto de observaciones de variables posiblemente correlacionadas, en un conjunto de valores de variables no correlacionadas, denominadas componentes principales. El número de componentes principales es menor o igual que el número de variables originales. Esta transformación se define de tal modo que la primer componente principal tenga el porcentaje de varianza más grande posible (esto es, acumular la mayor variabilidad en los datos tanto como sea posible), y cada componente sucesiva en turno, debe poseer la mayor varianza posible bajo la condición de que debe ser ortogonal (no correlacionada) a las componentes precedentes (Jolliffe, 2002).

PCA es uno de los métodos más antiguos y mejor conocidos que se emplea para reducir la dimensionalidad de un conjunto de datos. De forma intuitiva, se puede decir que esta técnica permite encontrar las causas de la variabilidad de un conjunto

de datos y las ordena por importancia.

Para construir la transformación lineal, primero se debe construir la matriz de covarianza, y enseguida se debe descomponer en sus valores y vectores propios, cuya existencia está asegurada debido a la simetría de la matriz. Esta transformación lineal lleva las antiguas coordenadas a las coordenadas de la nueva base, para reducir la dimensionalidad de los datos. Otra forma de realizar PCA, es la descomposición en valores singulares de la matriz de datos, después de un preprocesamiento.

La ventaja más destacada de PCA es que retiene aquellas características del conjunto de datos que contribuyen más a su varianza, al mismo tiempo que mantiene un orden de bajo nivel de los componentes principales e ignora los de alto nivel, todo esto sin mucha pérdida de información (Smith, 2002; Shlens, 2005; Lee y Verleysen, 2007).

2.6.2. ISOMAP

Isometric feature mapping o ISOMAP, es un algoritmo para reducir la dimensionalidad en un modo no lineal, que emplea la distancia de grafo como aproximación a la distancia geodésica. Combina las mejores características de los métodos tradicionales, como PCA y MDS: eficiencia computacional, optimalidad global y garantía de convergencia asintótica, además de que cuenta con la flexibilidad de aprender una amplia gama de variedades no lineales. Se basa principalmente en el funcionamiento de MDS pero busca preservar la geometría intrínseca de los datos, capturada por la distancia de grafo entre todos los puntos de la variedad. Esto es, mientras MDS emplea la distancia Euclideana como métrica, ISOMAP usa la distancia de grafo en el procedimiento algebraico del algoritmo. El uso de la distancia de grafo hace que MDS, que es puramente lineal, se convierta un método no lineal en ISOMAP. Sin embargo, es preciso destacar que las capacidades no lineales de ISOMAP son exclusivamente provistas por la distancia de grafo.

ISOMAP computa eficientemente una solución óptima global; así mismo, tiene la ventaja de reducir la dimensionalidad con una manipulación algebraica simple, rápida y directa. El algoritmo consta de tres pasos. El primero determina los puntos que son vecinos en una variedad $\mathcal{M}$, basado en las distancias $d_x(i,j)$ entre pares de puntos i,j en el espacio de entrada X. Esto se puede realizar haciendo uso de la regla-k o de la regla-ϵ (ver §2.3.1). El segundo paso consiste en estimar las distancias geodésicas $d_{\mathcal{M}}(i,j)$ entre todos los pares de puntos de la variedad $\mathcal{M}$. Esto se lleva a cabo al

calcular las distancias de grafo $d_G(i, j)$ en el grafo G, que se obtuvo mediante la regla de construcción correspondiente. En el último paso se aplica el MDS clásico a la matriz de distancias de grafo $D_G = \{d_G(i, j)\}$ (Tenenbaum et al., 2000).

Del mismo modo que PCA y MDS garantizan que dados los datos suficientes, es posible recuperar la estructura verdadera de variedades lineales, ISOMAP garantiza asintóticamente, recuperar la dimensión verdadera y la estructura geométrica de una amplia gama de variedades no lineales. Por otro lado, ISOMAP hereda una de las mayores limitantes de MDS, posee un modelo muy rígido, ya que se encuentra restringido a la proyección en el hiperplano. Sin embargo, en comparación con PCA y MDS, ISOMAP es mucho más poderoso (Lee y Verleysen, 2007).

Capítulo 3

Desarrollo del proyecto

En este capítulo, se presenta una clasificación para los conjuntos de datos en función del número de elementos que posee. Enseguida, se provee una breve descripción acerca del entorno (*hardware* y *software*) en el cual se desarrolló el proyecto, así como de los conjuntos de datos empleados en los experimentos.

También, se proporcionan detalles sobre la implementación del procedimiento para el cómputo de las distancias geodésicas, así como algunas de las consideraciones que se hicieron en cada módulo que lo conforma. Finalmente, se incluye una breve explicación de su posterior integración en ISOMAP. Esta integración se hizo con el fin de mejorar el desempeño en tiempo de este método de reducción de la dimensionalidad no lineal.

3.1. Conjunto de datos grande

Actualmente, la nueva generación de computadoras incorpora grandes prestaciones de recursos, lo que ha permitido la creación y procesamiento de conjuntos de datos masivos (Unwin et al., 2006). Esto ha sido aprovechado en áreas como: bioinformática, biomedicina o minería de datos. Es por ello que, también en el área de aprendizaje automático se ha vuelto necesario, cada vez más, trabajar con grandes cantidades de datos.

Pero, ¿a qué nos referimos cuando decimos que un conjunto de datos es "grande"? Encontrar una definición general que pueda aplicarse en todos los casos, es difícil, ya que "grande" es un término impreciso, incluso ambiguo. Un elefante es grande si lo comparamos con una hormiga. Sin embargo, si el mismo elefante lo comparamos con el planeta Tierra, es una nimiedad, casi tanto como la Tierra lo es frente al Sol. No

sólo en cuanto a tamaño se refiere, existen estas imprecisiones, las hay también en forma, posición, tiempo, color, textura, e incluso la semántica que describe lo que son.

Por ello, para definir a qué nos referimos con "grande", debemos contextualizar el concepto en el campo donde lo vamos a emplear, considerando los aspectos más importantes.

Un conjunto de datos que contenga miles de millones de observaciones (por ejemplo, el genoma humano), puede ser mediano o grande para una supercomputadora o un clúster de computadoras. Sin embargo, ese tipo de sistemas de cómputo no está al alcance de un investigador promedio, y en caso de estarlo, el acceso a este tipo de recursos es limitado. Tomando en cuenta que el lector al que está orientada esta tesis, tiene a su alcance únicamente una computadora personal, y por ende, recursos bastante más limitados a los de una supercomputadora o un clúster de computadoras, el tamaño de los conjuntos de datos que se manejen aquí, será condicionado por dichos recursos disponibles.

Partiendo del hecho de que hablaremos de una computadora personal como campo de pruebas, otro elemento a considerar es la capacidad de la memoria RAM. Dicha capacidad viene dada por dos factores principales: uno físico y otro lógico. Del lado físico, se tiene que las tarjetas de memoria, vienen diseñadas para poder almacenar una cantidad de datos determinada por sus componentes. En contraparte, el lado lógico se ve determinado por el manejo que haga el sistema operativo del recurso. De manera experimental, se descubrió que el sistema operativo dedica aproximadamente 700MB a cada proceso (Bautista-Villavicencio y Cruz-Barbosa, 2011).

Otro factor igualmente importante es que dentro del grupo de lectores a los que se orienta esta tesis, se encuentra la comunidad del aprendizaje automático, cuya fuente primaria de conjuntos de datos para experimentación (entrenamiento y pruebas), es el repositorio de aprendizaje automático de la Universidad de California (UCI) (Frank y Asuncion, 2010), que a partir del año 2009 alberga el contenido del archivo de descubrimiento de conocimiento en bases de datos (*KDD Archive*, del inglés *Knowledge Discovery in Databases Archive*), que a su decir, contiene conjuntos de datos "grandes". El repositorio UCI cuenta con aproximadamente 200 conjuntos de datos, de los cuales clasifica a 176 en tres grupos. El primero agrupa los conjuntos con menos de 100 elementos (11), mientras que el segundo contiene aquellos conjuntos que tienen de 100 a 1 000 instancias (84), y el tercer grupo incluye a los conjuntos de datos que tienen más de 1 000 elementos (81). Analizando el tercer grupo, se destaca que 45 de los 81 conjuntos pueden ser almacenados y manejados en una computadora personal,

esto es, aquellos conjuntos con máximo 11 000 elementos.

Con base en lo anterior, definimos el tamaño de un conjunto de datos para los intereses de este trabajo, de la siguiente manera. Un conjunto de datos es:

- pequeño, si cuenta con un máximo de 100 elementos.
- mediano, si no es pequeño y cuenta con un máximo de 1 000 elementos.
- grande, si no es mediano y cuenta con un máximo de 11 000 elementos, que pueden ser almacenados en la memoria RAM de una computadora personal.
- muy grande, si no es grande y no puede ser almacenado en la memoria RAM de una computadora personal, esto es, si lo conforman más de 11 000 elementos.

Además del tamaño del conjunto de datos, también nos interesa su naturaleza. Es común que en aprendizaje automático, los conjuntos de datos sean un grupo de valores que representan observaciones, variables o clases. Cada observación puede ser un renglón de la matriz, mientras que las variables (o dimensiones) y las clases (si las hay) conformen las columnas, o viceversa. En esta tesis se asume que los conjuntos de datos son numéricos.

3.2. Especificaciones de hardware y software

Los experimentos se realizaron en una computadora de escritorio con un microprocesador AMD Athlon 64 X2 BE-2400 doble núcleo a 2300MHz, y 3GB de memoria RAM.

El entorno principal para desarrollo y experimentación, fue el sistema operativo GNU/Linux Ubuntu 9.10 Karmic Koala x86 (32 bits), con núcleo Linux 2.6.31-22-generic. Las pruebas con Matlab se realizaron en el sistema operativo Microsoft Windows 7 Ultimate x86 (32 bits) y el *software* Matlab R2010b.

El lenguaje de programación utilizado para el desarrollo del *software*, fue C++ (utilizando el compilador GCC (Ubuntu 4.4.1-4ubuntu9) 4.4.1). Se eligió este lenguaje por la rapidez de ejecución de los programas que genera, puesto que se quiere entregar un procedimiento optimizado en términos del manejo de recursos, principalmente el tiempo y el almacenamiento, en cada etapa que lo conforma. El *software* licenciado Matlab, fue utilizado para el desarrollo de prototipos iniciales y para la realización de comparaciones con métodos tradicionales en las áreas de aprendizaje automático.

3.3. Conjuntos de datos utilizados

A continuación se describen brevemente los conjuntos de datos utilizados en los experimentos. Se tomaron cinco conjuntos de datos del repositorio de aprendizaje automático de la Universidad de California (UCI) (Frank y Asuncion, 2010), y otros más de Julià-Sapé et al. (2006) que son datos de espectros de resonancia magnética nuclear (RMN) de tumores cerebrales humanos.

3.3.1. UCI

Cinco son los conjuntos de datos tomados del repositorio de aprendizaje automático UCI.

El primero de ellos es *Ecoli*, el cual consiste en 336 elementos de 7 dimensiones, pertenecientes a 8 clases que representan la localización de proteínas.

El segundo conjunto, denominado aquí *German*, proviene de la versión numérica del conjunto *German-credit-data*. Consiste en 1 000 elementos de 24 dimensiones, que pertenecen a buenos o malos riesgos de crédito.

El tercer conjunto, *Segmentation*, está formado por 2 310 elementos de 19 dimensiones, que representan medidas realizadas a algunas características de imágenes segmentadas. Los elementos pertenecen a siete clases diferentes.

El cuarto conjunto de datos, llamado *Pageblocks*, consiste en mediciones a distintos documentos, pertenecientes a cinco clases. Se encuentra formado por 5 473 elementos descritos por 10 atributos.

El quinto conjunto, denominado aquí *Pendigits*, se conforma por 10 992 elementos de 16 dimensiones, correspondientes a medidas de coordenadas (x, y), pertenecientes a 10 dígitos numéricos (clases).

3.3.2. Datos de espectros de tumores cerebrales RMN

Los datos RMN fueron adquiridos a diferentes tiempos de eco, cortos (STE), largos (LTE) y una combinación de ambos por una concatenación directa. Los datos pertenecen a la base de datos internacional multicentro (Julià-Sapé et al., 2006), y consiste de: (1) 217 espectros STE, incluyendo 58 meningiomas (mm), 86 glioblastomas (gl), 38 metastases (me), 22 astrocytomas grado II (a2), 6 oligoastrocytomas grado II (oa), y 7 oligodendrogliomas grado II (od); (2) 195 espectros LTE, que incluye 55 mm, 78 gl, 31 me, 20 a2, 6 oa, y 5 od. (3) 195 elementos resultantes de combinar, a través de

una concatenación directa, los espectros STE y LTE de los mismos pacientes. Sólo se analizan las regiones de los espectros que se consideran clínicamente relevantes. Éstas consisten de 195 valores de intensidad de frecuencia (medidos en partes por millón (ppm), una unidad de posición relativa de frecuencia adimensional en el vector de datos), comenzando a 4.25 ppm. Estas frecuencias son las características de los datos observados.

3.4. Módulos del proyecto

En esta sección se describe el desarrollo del procedimiento para realizar el cómputo de las distancias geodésicas para un conjunto de datos grande (CDG^2).

Es preciso decir que el diseño y desarrollo del procedimiento se hizo siguiendo estándares, para beneficiar su calidad, compatibilidad multiplataforma y claridad de código, con el fin de ser una alternativa real para la comunidad interesada. Con esto en mente, se desarrolló una biblioteca de aprendizaje automático (denominada MMLL[1]), cuya primera versión incluye los módulos necesarios para realizar el cómputo de la distancia geodésica.

El diagrama de clases que se muestra en la figura 3.1, incluye únicamente las clases principales de MMLL y sólo se detallan las más importantes. Algunos de los métodos mostrados en el diagrama de clases, se describen en las secciones subsecuentes.

Esta biblioteca fue codificada en el lenguaje de programación C++, siguiendo los lineamientos de la programación orientada a objetos, y sin dependencia de bibliotecas de terceros, con el fin de maximizar la compatibilidad multiplataforma. MMLL se liberó bajo la licencia GPLv3[2], para no comprometer su uso y desarrollo futuro.

A continuación se describen los detalles de la implementación del procedimiento, etapa por etapa, según lo mostrado en la figura 3.2.

El cómputo de la distancia geodésica es intratable en términos computacionales, sin embargo, es ampliamente conocido que la distancia geodésica puede ser aproximada por distancias de grafo. Así, en vez de encontrar el arco de longitud mínima entre dos puntos que se encuentran contenidos en una variedad, se debe hallar el camino más corto entre los nodos (que representan a dichos puntos) que pertenecen al grafo (que representa a la variedad), como se vio en §2.2. Cabe recordar de la §1.1 que para construir un grafo a partir de un conjunto de datos, cada elemento de este es

[1] MMLL, del acrónimo recursivo (en inglés): *MMLL: Machine Learning Library*.
[2] Para más detalles, consúltese: http://www.gnu.org/licenses/gpl-3.0.html

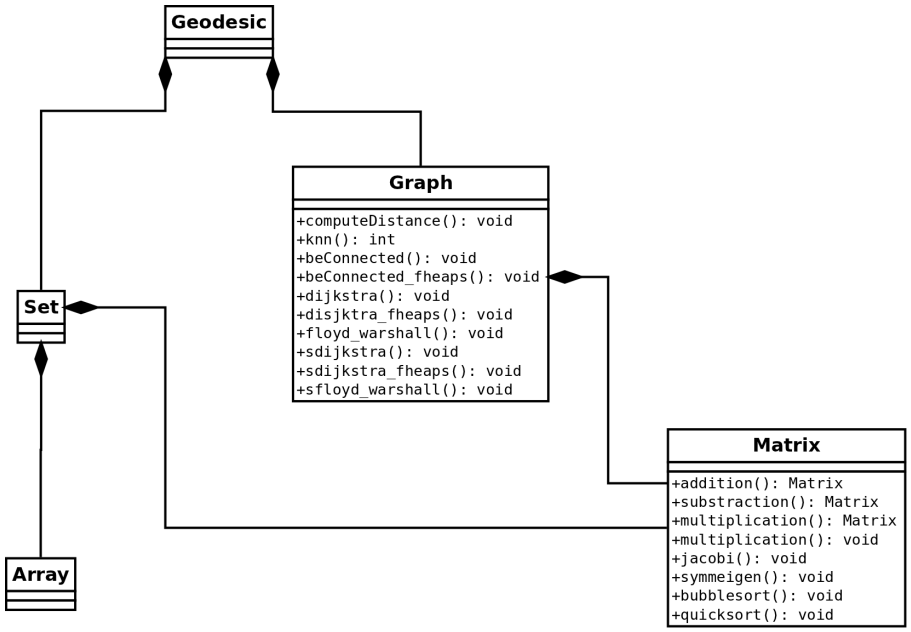

Figura 3.1: Diagrama de clases simplificado de MMLL.

considerado como un vértice, las conexiones permitidas (utilizando la regla-k ó -ϵ, ver §2.3.1) son las aristas, y los pesos respectivos son las distancias Euclideanas entre los puntos relacionados.

3.4.1. Conjunto de datos de entrada

Un conjunto de datos típico (en aprendizaje automático), es una colección de observaciones de características, de un fenómeno determinado o de uno o varios objetos de interés, y es común que el conjunto de datos se escriba en forma de lista. Por lo regular, cada entrada de la lista se conforma por valores de las características observadas, y la etiqueta que la identifica. A cada valor observado se conoce como atributo o dimensión, mientras que a las etiquetas se les conoce como clases, las cuales son útiles para realizar tareas de clasificación.

De ahí que, de forma intuitiva, se opte por almacenar el conjunto de datos en una matriz, A de $(m \times n)$, donde m es el número de observaciones (o instancias), y n es el número de dimensiones (o atributos), y un arreglo $\mathsf{B}[\,]$ de $(m \times 1)$, que almacene las clases asociadas.

Sin embargo, esta forma de almacenar al conjunto de datos, tiene una seria desventaja. Los conjuntos de datos tienen, en su inmensa mayoría, más observaciones

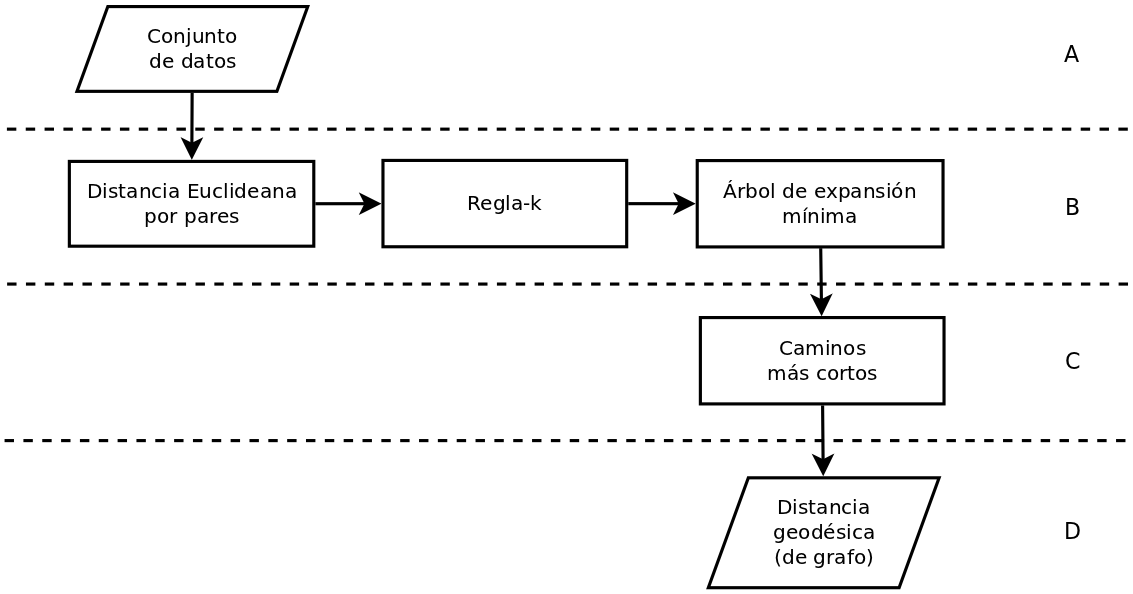

Figura 3.2: Esquema del procedimiento para el cómputo de las distancias de grafo. La etapa (A) representa el conjunto de datos de entrada. La etapa (B) construye el grafo ponderado, conexo y no dirigido. La etapa (C) realiza el cómputo de la distancia de grafo, la cual es presentada como salida en la etapa (D).

que dimensiones, es decir, $m > n$ (incluso es posible que $m \gg n$). Dado que el modelado de una matriz en el lenguaje C++ es un arreglo de arreglos, y tomando en cuenta el esquema planteado anteriormente, para almacenar el conjunto de entrada sería necesario crear un arreglo de $m \times 1$ que a su vez almacene en cada entrada, un arreglo de $1 \times n$. Es decir, se necesitan $m+1$ apuntadores para almacenar el conjunto de datos (ver figura 3.3).

Por otro lado, si se opta por transponer el conjunto de datos para su almacenamiento en una matriz, se realiza un ahorro. Esto es, almacenar dicho conjunto en una matriz A1 de $(n \times m)$, donde m es el número de observaciones, y n es el número de dimensiones.

En este segundo esquema, se necesitan únicamente $n + 1$ apuntadores para almacenar el conjunto de datos, como lo ilustra la figura 3.4. La implementación que se incluye en MMLL, para la manipulación del conjunto de datos de entrada, se realizó según lo recién descrito, y todo el manejo es totalmente transparente para el usuario.

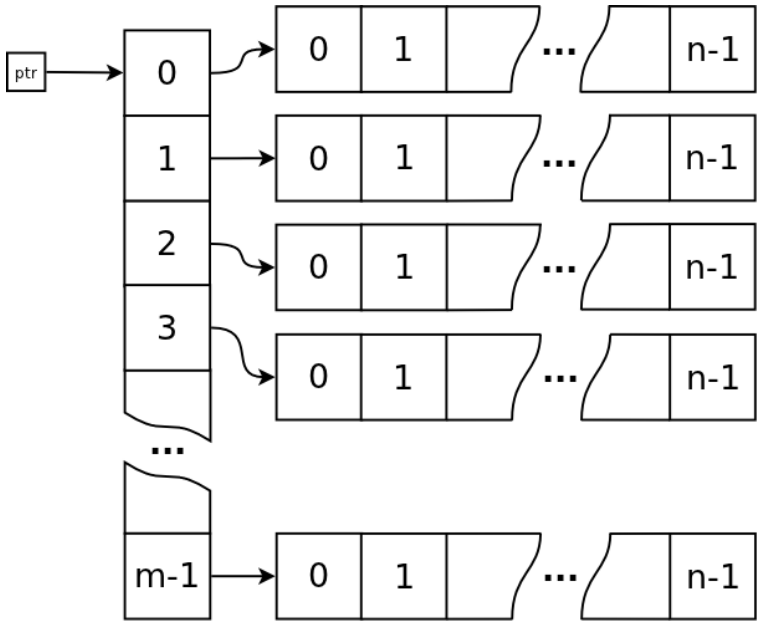

Figura 3.3: Almacenamiento del conjunto de datos en la matriz $\mathbf{A}(m \times n)$.

3.4.2. Construcción del grafo

A partir del conjunto de datos de entrada se construye el grafo, para poder realizar el cómputo de las distancias geodésicas. Dicho grafo es construido al conectar los puntos sucesivos más cercanos. Los datos son los vértices, las conexiones permitidas son las aristas, y los pesos son las distancias Euclideanas entre dichos puntos.

La construcción de grafos es una etapa esencial ya que debe capturar las relaciones de vecindad entre los puntos del conjunto de datos original, de modo que se garantice la conservación de su estructura intrínseca. Esta etapa consta de tres módulos. Se inicia la construcción del grafo haciendo uso de un modelo de vecinos más cercanos, y enseguida se comprueba que ningún nodo quede desconectado. Al final, se asegura un grafo conexo, ponderado y no dirigido.

Distancia Euclideana por pares

A partir del conjunto de entrada, se realiza el cálculo de la distancia Euclideana por pares, que es la longitud denotada por la ecuación 2.2, que existe entre cada par de instancias pertenecientes al conjunto. Estas distancias se almacenan en una matriz de $m \times m$, donde la entrada (i, j) con $i \neq j$ es la distancia Euclideana de i a j, mientras que la entrada (i, j) con $i = j$ es 0 (axioma de no negatividad, §2.1).

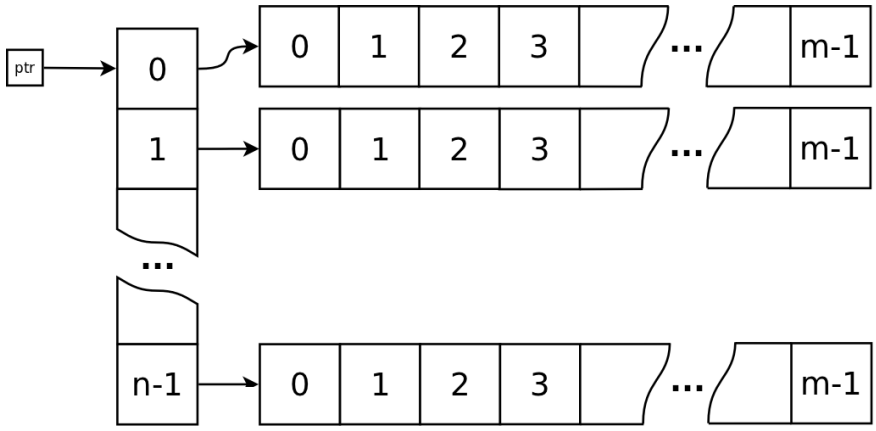

Figura 3.4: Almacenamiento del conjunto de datos en la matriz $A1(n \times m)$.

El cómputo de esta matriz es indispensable para los siguientes módulos de esta etapa. El inmediato siguiente consulta las distancias Euclideanas para saber cuán cerca se encuentra un nodo de otro, y determinar si deben unirse. Para el último módulo de esta etapa, será necesario consultar esta misma matriz y determinar los enlaces que se deben agregar entre algunos nodos.

Sin embargo, con este esquema de almacenamiento, hay 50 % de redundancia en la matriz de distancias Euclideanas (matriz simétrica), puesto que la entrada (i, j) es igual a la entrada (j, i), como consecuencia natural del axioma de simetría de las funciones de distancia (§2.1). Tomando esto en cuenta, y considerando también que en la siguiente etapa (ver figura 3.2-C) será necesario almacenar las distancias geodésicas en otra matriz de idénticas dimensiones y con el mismo factor de redundancia, aunado al deseo expreso de optimizar los recursos, se procedió a diseñar una alternativa que mitigara estos problemas.

Así, en MMLL se implementó una matriz llamada distancesMatrix, que es el resultado de consolidar la matriz de distancias Euclideanas y la matriz de distancias geodésicas, en una sola matriz, con el fin de eliminar la redundancia innecesaria de cada una de ellas. distancesMatrix es una matriz de $m \times m$, cuya matriz triangular superior contiene las distancias Euclideanas, mientras la triangular inferior almacena las distancias geodésicas (ec. 3.1).

$$\mathsf{distancesMatrix} = \begin{bmatrix} 0 & E_{12} & E_{13} & \dots & E_{1i} & \dots & E_{1m} \\ g_{21} & 0 & E_{23} & \dots & E_{2i} & \dots & E_{2m} \\ g_{31} & g_{32} & 0 & \dots & E_{3i} & \dots & E_{3m} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ g_{j1} & g_{j2} & g_{j3} & \dots & 0 & \dots & E_{jm} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ g_{m1} & g_{m2} & g_{m3} & \dots & g_{mi} & \dots & 0 \end{bmatrix} \tag{3.1}$$

Como consecuencia de este nuevo esquema de almacenamiento, se tuvo que modificar el modo de acceder a los datos contenidos en la matriz distancesMatrix, a través de sus métodos setValue() y getValue() asociados, con el fin de acceder correctamente a los datos. Los métodos setValue() y getValue() hacen que el manejo especial requerido sea completamente transparente para el usuario.

Regla-k

Este módulo implementa el algoritmo de los k-vecinos más cercanos (KNN) descrito en §2.3.1. Se toma cada instancia del conjunto como nodo actual (uno a la vez), y se consulta distancesMatrix para determinar los k nodos más cercanos a él, para luego agregar un arco entre ellos e ir formando el grafo.

Es en este módulo de la segunda etapa (ver figura 3.2-B), que comienza propiamente la construcción del grafo. El grafo recién construido necesita ser representado de modo que no sólo permita realizar consultas y actualizaciones, sino que además las simplifique y facilite. Partiendo de lo anterior, y como se puede notar a partir de la figura 3.2, existen diversas alternativas para el desarrollo de ciertos módulos. En lo que concierne a la forma de representar el grafo, se tienen como opciones las listas y las matrices de adyacencia, como se vio en §2.4.

En MMLL se implementaron la lista de adyacencia (§2.4.1) y la matriz de adyacencia (§2.4.2). La matriz de adyacencia permite acceder rápidamente a los datos que almacena, pero consume grandes cantidades de memoria. Cuando el valor de k es pequeño, se puede llegar a tener un grafo disperso, lo que implicaría que su matriz de adyacencia también sea dispersa. Así, la lista de adyacencia puede ser una mejor opción, ya que sus demandas de memoria son menores. Como alternativa de representación, se incluyen dos implementaciones correspondientes a dos tipos de matrices dispersas, la lista coordenada y la lista de listas (§2.4.3). Si el grafo

creado es muy grande, consecuencia de trabajar con un conjunto de datos grande, será indispensable utilizar una representación adecuada para evitar los problemas de espacio en memoria.

Dependiendo de la elección del parámetro k, al finalizar el algoritmo KNN nos encontraremos con seguridad en una de las siguientes situaciones:

- Se tiene un grafo conexo, en cuyo caso se puede proceder a realizar el cómputo de las distancias geodésicas.

- Se tiene un conjunto de subgrafos, que habrá que conectar apropiadamente de forma que no se distorsione la estructura original del conjunto.

Para identificar ante qué situación nos encontramos, y en caso de ser la segunda, para proceder a la obtención del grafo conexo, se hace uso de una versión modificada de un algoritmo para hallar el árbol de expansión mínima.

Árbol de expansión mínima

Dado que es posible que al finalizar KNN se obtenga un conjunto de subgrafos, éstos se deben identificar y conectar de manera que se capture la estructura intrínseca del conjunto de datos, minimizándose así la pérdida de información. Para ello, se utilizó un método derivado del algoritmo de Prim (§2.3.2) propuesto en Cruz-Barbosa (2009).

El método sugiere seguir el algoritmo de Prim, y que cuando se encuentre un subgrafo, se proceda a conectarlo inmediatamente. Para interconectarlo, el esquema más simple consiste en unir el subgrafo recién hallado, con el nodo más cercano del resto de los nodos. Así, se añade una arista entre el par de nodos, correspondiente a dos puntos que se encuentran en dos subgrafos no conectados, cuya distancia Euclideana es la menor de entre todos los pares de puntos que aún no se encuentran conectados. Esta modificación permite que se continúe con el algoritmo de Prim de forma normal.

En MMLL se implementó `beConnected()`, que es el nombre del método con la modificación recién descrita. Del mismo modo, se incorporó al método el uso de *F-heaps* (§2.2.1) para hacer fácil la selección del arco que será agregado al árbol, y como consecuencia, contar con una implementación que funcione de manera eficiente.

Hay que notar que en caso de que el grafo sea conexo desde el principio del procedimiento, no sufrirá modificaciones. Sin embargo, si la entrada es un conjunto

de subgrafos, al finalizar la versión modificada del algoritmo de Prim, se tendrá con seguridad un grafo conexo que conserva la estructura real del conjunto de datos.

3.4.3. Caminos más cortos

Una vez que se tiene el grafo conexo, ponderado y no dirigido, éste puede ser usado para obtener una buena aproximación a la distancia geodésica con la distancia de grafo.

Como se vio en §3.4.2, existen diversas alternativas para el desarrollo de ciertos módulos (ver figura 3.2). Para encontrar los caminos más cortos, existe un par de alternativas principales, por un lado el algoritmo de Dijkstra (§2.2.1) que es un algoritmo ávido y el método más y mejor conocido para resolver este tipo de problemas; y por otro, el algoritmo de Floyd-Warshall (§2.2.1) que es un ejemplo de programación dinámica. Ambos algoritmos para encontrar los caminos más cortos, fueron implementados e incluidos en MMLL. Del algoritmo de Dijkstra se hicieron dos versiones: la original, y una que incorpora F-heaps para mejorar su rendimiento.

A partir del grafo recién creado, se obtiene la matriz que contiene las distancias geodésicas calculadas, como producto de aplicar repetidamente el algoritmo de Dijkstra, o bien, aplicar únicamente una vez el algoritmo de Floyd-Warshall. Como se explicó en §3.4.2, distancesMatrix almacena en su triangular inferior la matriz de distancias geodésicas, que es la misma que se entrega como salida del procedimiento (ver figura 3.2-D).

3.4.4. Métodos y estructuras de datos auxiliares

Se implementaron e incluyeron en MMLL, algunos métodos y estructuras de datos auxiliares necesarios para la realización adecuada y eficiente de algunos módulos a lo largo de las diferentes etapas del procedimiento para el CDG^2 (ver figura 3.2), mismos que se listan y describen brevemente a continuación:

- Manejo de archivos. Los métodos permiten cargar conjuntos de datos desde archivo, a una estructura de datos interna, y viceversa, la descarga de la estructura a un archivo.

- Soporte para información de clase. Los métodos son capaces de realizar el manejo de la etiqueta de clase a la que pertenecen las instancias del conjunto de datos

de entrada. Así mismo, soportan diferentes configuraciones en la posición de la clase respecto a los atributos (al inicio del renglón, al final o sin clase).

- Tipos de datos para matrices y arreglos. Todos los arreglos y matrices empleados en los métodos pertenecientes a MMLL, se desarrollaron con total soporte para los tipos de datos: *double*, *float*, e *int*.

- Representación de grafo. Todos los métodos que trabajan con grafos, son capaces de usar cualquier versión de representación para grafo: matriz de adyacencia, lista de adyacencia, lista de listas y lista coordenada.

- Ordenamiento. Se incluyeron dos métodos para ordenamiento, burbuja y rápido (*quicksort*), para su uso dentro de otros métodos. Para *quicksort*, que es un algoritmo naturalmente recursivo, se empleó simulación de recursión en su implementación, para disminuir los efectos negativos de la recursión, especialmente aquellos que repercuten en la memoria.

- Estructuras de datos. Como parte de MMLL, se emplearon además de las matrices y los arreglos, los F-heaps y las pilas. Los F-heaps son empleados como cola de prioridad en el algoritmo de Dijkstra y el método beConnected() (derivado del algoritmo de Prim). La pila se empleó en *quicksort* para realizar la recursión simulada.

3.5. Una aplicación para Reducción de Dimensionalidad

Como una prueba de que el procedimiento desarrollado, ayuda a mejorar los métodos de aprendizaje automático actuales, se integró dicho módulo al método para la reducción de la dimensionalidad no lineal ISOMAP, que es un algoritmo perteneciente al grupo de algoritmos no supervisados.

A continuación, se explican algunos detalles acerca de la implementación de ISOMAP.

Como se vio en §2.6.2 y como ilustra la figura 3.5, el algoritmo ISOMAP se compone por tres pasos. Los primeros dos pasos, pertenecientes a la primera etapa, comprenden el cálculo de la distancia geodésica, como se explicó en §3.4. El último paso, realiza propiamente la reducción de la dimensionalidad.

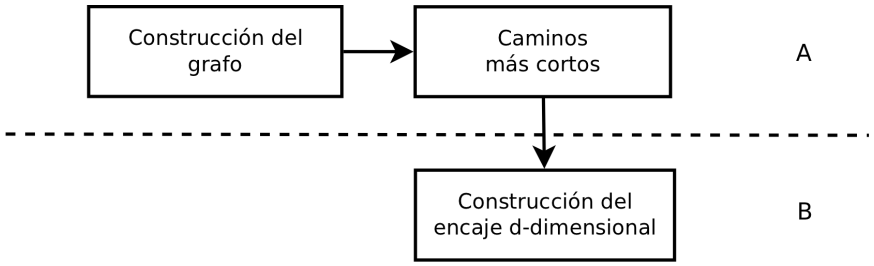

Figura 3.5: Algoritmo ISOMAP. La etapa (A) calcula las distancias geodésicas. La etapa (B) construye el encaje d-dimensional, para realizar la reducción de la dimensionalidad.

El primer paso para la construcción del encaje d-dimensional (ver figura 3.5-B), consiste en obtener la matriz τ a partir de la matriz de distancias geodésicas D, donde $\tau(D) = -HSH/2$, $S_j = D_{ij}^2$ y H la matriz centrada. Enseguida, se obtienen los valores y vectores propios de la matriz τ, mediante el algoritmo QR (§2.5). Finalmente, se reduce a la dimensión deseada, usando los valores λ_p y vectores $\mathbf{v}_p$ propios recién obtenidos. El vector reducido, $\mathbf{y}$, se construye como sigue:

$$\mathbf{y}_i = \sqrt{\lambda_p \mathbf{v}_p^i} \tag{3.2}$$

donde $\mathbf{v}_p^i$ es el i-ésimo componente del p-ésimo vector propio.

Puesto que ISOMAP se deriva de MDS (Kruskal, 1964a,b), y la única diferencia entre ambos es la métrica que utilizan (MDS emplea la distancia Euclideana, mientras que ISOMAP la geodésica), si el conjunto de entrada no presenta pliegues en su geometría, los resultados de ambos algoritmos podrían ser altamente parecidos.

3.5.1. Métodos auxiliares

Los métodos auxiliares que se implementaron para ISOMAP, se listan y describen brevemente a continuación.

- Operaciones matriciales. Se incluyen métodos para realizar operaciones matriciales, entre las que destacan: la suma, resta y multiplicación de matrices; transpuesta y cuadrado de una matriz; creación de matrices identidad y centrada; así como multiplicación de una matriz por un escalar.

- Optimización de operaciones. Todas las operaciones matriciales se optimizaron para los casos en donde la entrada es una matriz cuadrada.

- Valores y vectores propios. Se utilizaron dos métodos para hallar los valores y vectores propios (*eigensystem*) de una matriz cuadrada, el método de Jacobi y el algoritmo QR.

Capítulo 4

Resultados

Se realizaron diferentes tipos de experimentos con el fin de encontrar respuestas a las principales interrogantes que surgen cuando se desea realizar el cómputo de las distancias geodésicas.

La primer parte de los experimentos está relacionada al tiempo y almacenamiento computacionales consumidos al realizar el CDG^2, usando para ello, diferentes configuraciones en el proceso. Aquí se evalúan diferentes representaciones de grafo y algoritmos para encontrar los caminos más cortos. La segunda parte, muestra el impacto de emplear el módulo para el CDG^2, como base para ISOMAP. Cabe mencionar que la mayor parte de los resultados fueron publicados en Bautista-Villavicencio y Cruz-Barbosa (2011); Cruz-Barbosa et al. (2011a,b).

A continuación, se describen los experimentos y se presentan y analizan los resultados obtenidos.

4.1. Matrices completas y dispersas

Cuando se trabaja con conjuntos de datos grandes es crucial optimizar los recursos. En este caso, es muy importante encontrar una representación que permita el manejo del conjunto, y al mismo tiempo, permita realizar operaciones con rapidez.

El objetivo principal de este experimento es comparar y evaluar el desempeño de dos implementaciones de matriz de adyacencia, matriz completa y matriz dispersa, como representación de grafo, al realizar el cómputo de las distancias geodésicas. El proceso para realizar el cómputo de las distancias geodésicas se muestra en la figura 3.2, y se detalla en §§3.4.1-3.4.3.

Aquí, se empleó el algoritmo (básico) de Dijkstra, e inicialmente, al parámetro k se le asignó el valor 10, con el fin de obtener un grafo conexo después de aplicar la regla-k. En seguida, se asignó a k el valor de 1, para mostrar el desempeño del procedimiento con un grafo no conexo (después de la regla-k) y disperso. Estos valores de k se determinaron experimentalmente como se sugiere en §2.3.1.

Para llevar a cabo el experimento, se hizo uso de la configuración *hardware/software* descrita en §3.2, y los conjuntos de datos obtenidos de la UCI, descritos en §3.3.1.

Los resultados de desempeño en tiempo, para el cómputo de las distancias geodésicas, usando $k = 10$ y el algoritmo de Dijkstra, se muestran en el cuadro 4.1.

Conjunto de datos (# ítems)	**Representación** (Matriz de adyacencia)	**Tiempo (s)**
Ecoli	Completa	0.43
(336)	Dispersa	7.92
German	Completa	12.43
(1 000)	Dispersa	646.23
Segmentation	Completa	185.57
(2 310)	Dispersa	16 385.91
Pageblocks	Completa	3 621.90
(5 473)	Dispersa	707 227.99
Pendigits	Completa	−−
(10 992)	Dispersa	−−

Cuadro 4.1: Resultados de desempeño en tiempo para el cómputo de las distancias geodésicas (asumiendo un grafo conexo al establecer $k = 10$) para varios conjuntos de datos UCI y dos representaciones de grafo, matrices de adyacencia completa y dispersa. El símbolo '−−' indica que el límite de memoria se excedió.

Se puede ver que el desempeño de la matriz completa rebasa al correspondiente a la matriz dispersa para todos los conjuntos de datos, excepto para *Pendigits*, donde la memoria se excedió en ambas representaciones. Así mismo, es notorio cómo el tiempo aumentó cuando se hizo uso de la matriz dispersa, al incrementar el tamaño del conjunto de datos.

Ahora, usando $k = 1$ y nuevamente el algoritmo de Dijkstra, se tiene un grafo no conexo y disperso. Los resultados de desempeño en tiempo, se muestran en el cuadro 4.2.

En general, se observa que el procedimiento de conexión de componentes influye en

Conjunto de datos (# ítems)	**Representación** (Matriz de adyacencia)	**Tiempo (s)**
Ecoli	Completa	0.47
(336)	Dispersa	8.21
German	Completa	12.85
(1 000)	Dispersa	685.23
Segmentation	Completa	186.55
(2 310)	Dispersa	16 886.08
Pageblocks	Completa	3 483.08
(5 473)	Dispersa	719 319.49
Pendigits	Completa	−−
(10 992)	Dispersa	−−

Cuadro 4.2: Resultados de desempeño en tiempo para el cómputo de las distancias geodésicas (asumiendo un grafo no conexo al establecer $k = 1$) para varios conjuntos de datos UCI y dos representaciones de grafo, matrices de adyacencia completa y dispersa. El símbolo '−−' indica que el límite de memoria se excedió.

los tiempos obtenidos. Sin embargo, la tendencia observada a partir de los resultados mostrados en el cuadro 4.1, se mantienen. Es importante destacar que la matriz dispersa tampoco pudo manejar al conjunto de datos *Pendigits*, ni siquiera cuando el grafo era disperso.

4.2. Resultados de tiempo y almacenamiento computacional

La mayoría de los métodos basados en grafos, realizan el cómputo de las distancias de grafo usando el algoritmo (básico) de Dijkstra y una representación de matriz (completa) de adyacencia. Los principales problemas de esta solución están relacionados a restricciones de tiempo y espacio computacional.

El presente experimento tiene dos objetivos principales. Primero, determinar experimentalmente qué combinación de representación de grafo y algoritmo de caminos más cortos produce el mejor tiempo al realizar el cómputo de las distancias geodésicas, cuando el número de elementos del conjunto de entrada se incrementa. Segundo, comparar y evaluar el desempeño en tiempo de la mejor combinación (de representación de grafo y algoritmo de caminos más cortos) encontrada en el

experimento anterior, usando la implementación en C++ que provee MMLL, y una implementación realizada en Matlab.

El proceso para realizar el cómputo de las distancias geodésicas se muestra en la figura 3.2, y se detalla en §3.4. Para realizar los experimentos, se estableció el parámetro $k = 10$, con el fin de obtener un grafo conexo después de aplicar la regla-k. A continuación, se asignó al parámetro k el valor de 1, para mostrar el desempeño del procedimiento con un grafo no conexo y disperso. Para estos experimentos se emplearon los conjuntos de datos obtenidos de la UCI, descritos en §3.3.1, y el arreglo *hardware/software* descrito en §3.2.

Los resultados de desempeño en tiempo, para el cómputo de las distancias geodésicas, usando $k = 10$, se muestran en el cuadro 4.3.

Aquí, la combinación de matriz de adyacencia como representación de grafo, y el algoritmo básico de Dijkstra como algoritmo para hallar los caminos más cortos, superó a las demás combinaciones, excepto a *Pageblocks*. Este resultado se debe al rápido acceso a los elementos almacenados en la matriz, cuando el algoritmo básico de Dijkstra los necesita. Es interesante notar cómo el desempeño en tiempo de la lista de adyacencia y el algoritmo de Dijkstra, es mejor para conjuntos de datos grandes. Usando el algoritmo de Dijkstra, la proporción del tiempo entre la lista de adyacencia y la matriz, se decrementa a medida que el número de elementos en el conjunto, se incrementa. Esto implica que la representación de lista de adyacencia es crucial para conjuntos de datos grandes. Este efecto se pronuncia en *Pendigits*, donde la representación de matriz de adyacencia no puede manejar al conjunto, debido a las restricciones de memoria del sistema operativo (se comprobó experimentalmente que éste dedica aproximadamente 700 MB por cada proceso). Para conjuntos de datos grandes, como *Pendigits*, se puede observar que la mejor combinación es la lista de adyacencia y Dijkstra con montículos de Fibonacci (F-heaps). Ahora, usando la representación de matriz, si se comparan los tiempos para Dijkstra y Dijkstra con F-heaps, se puede apreciar que la proporción de tiempo se decrementa cuando el número de elementos del conjunto se incrementa, y esta diferencia es más pronunciada para el algoritmo de Dijkstra con montículos de Fibonacci. Esta tendencia no se cumple para la representación de lista usando conjuntos de datos pequeños y medianos, pero es notablemente baja para conjuntos grandes como *Pendigits*. Por lo tanto, se puede inferir que para conjuntos grandes y muy grandes, el mejor desempeño para el cómputo de la distancia geodésica, es emplear la lista de adyacencia (o la matriz, cuando las restricciones de espacio no se toman en cuenta), y el algoritmo

Conjunto de datos (# ítems)	**Caminos más cortos**	**Representación**	**Tiempo (s)**
Ecoli (336)	Dijkstra	Matriz de adyacencia	0.43
	Dijkstra + Fheaps	Matriz de adyacencia	1.19
	Floyd-Warshall	Matriz de adyacencia	0.53
	Dijkstra	Lista de adyacencia	0.67
	Dijkstra + Fheaps	Lista de adyacencia	1.59
	Floyd-Warshall	Lista de adyacencia	*0.42
German (1 000)	Dijkstra	Matriz de adyacencia	*12.43
	Dijkstra + Fheaps	Matriz de adyacencia	25.03
	Floyd-Warshall	Matriz de adyacencia	23.67
	Dijkstra	Lista de adyacencia	16.18
	Dijkstra + Fheaps	Lista de adyacencia	38.39
	Floyd-Warshall	Lista de adyacencia	18.71
Segmentation (2 310)	Dijkstra	Matriz de adyacencia	*185.57
	Dijkstra + Fheaps	Matriz de adyacencia	297.31
	Floyd-Warshall	Matriz de adyacencia	347.16
	Dijkstra	Lista de adyacencia	229.83
	Dijkstra + Fheaps	Lista de adyacencia	511.59
	Floyd-Warshall	Lista de adyacencia	292.89
Pageblocks (5 473)	Dijkstra	Matriz de adyacencia	3 621.90
	Dijkstra + Fheaps	Matriz de adyacencia	4 031.93
	Floyd-Warshall	Matriz de adyacencia	18 369.84
	Dijkstra	Lista de adyacencia	*3 585.92
	Dijkstra + Fheaps	Lista de adyacencia	8 039.92
	Floyd-Warshall	Lista de adyacencia	10 409.90
Pendigits (10 992)	Dijkstra	Matriz de adyacencia	−−
	Dijkstra + Fheaps	Matriz de adyacencia	−−
	Floyd-Warshall	Matriz de adyacencia	−−
	Dijkstra	Lista de adyacencia	124 363.18
	Dijkstra + Fheaps	Lista de adyacencia	*66 105.34
	Floyd-Warshall	Lista de adyacencia	204 604.99

Cuadro 4.3: Resultados de desempeño en tiempo para el cómputo de las distancias geodésicas (asumiendo un grafo conexo al establecer $k = 10$) para varios conjuntos de datos UCI y diferentes configuraciones. El mejor tiempo para cada conjunto está señalado con un asterisco. El símbolo '−−' indica que el límite de memoria se excedió.

de Dijkstra con F-heaps. Lo opuesto ocurre con el algoritmo de Floyd-Warshall, independientemente de la representación de grafo, su desempeño sólo es notorio para conjuntos pequeños.

Ahora, el parámetro k de la regla-k, toma el valor de 1 para mostrar el desempeño en tiempo, cuando el procedimiento debe tratar con un grafo disperso y no conexo. Los resultados correspondientes se muestran en el cuadro 4.4.

En general, es notorio cómo el procedimiento de conexión de componentes, beConnected(), influye en los tiempos obtenidos. Sin embargo, la tendencia observada en los resultados mostrados por el cuadro 4.3 se mantiene. Además, se puede inferir a partir de los cuadros 4.3 y 4.4, que mientras más grande sea el conjunto de datos, menos se ve afectado el algoritmo de Dijkstra con montículos de Fibonacci por beConnected().

Finalmente, se compara el desempeño en tiempo de nuestro procedimiento para el CDG^2 en MMLL (desarrollado en C++), usando la matriz de adyacencia como representación del grafo, y el algoritmo de Dijkstra básico para encontrar los caminos más cortos, con una implementación en Matlab del mismo procedimiento usando la misma configuración. El parámetro k se establece en 10. Los resultados se muestran en el cuadro 4.5.

En todos los casos, los resultados de desempeño en tiempo para la implementación en C++ son mejores que la versión de Matlab. De hecho, cualquier resultado de los cuadros 4.3 y 4.4, es mejor que su contraparte obtenida con Matlab. Así mismo, el desempeño en tiempo para la versión desarrollada en Matlab, se incrementa a medida que crece el número de elementos en el conjunto. También se puede notar que Matlab no puede manejar conjuntos tan grandes como *Pageblocks*, o mayores.

4.3. Resultados de aplicación de reducción de dimensionalidad

El procedimiento optimizado para el CDG^2, descrito en §3.4, se inserta en el algoritmo para la reducción de la dimensionalidad no lineal, ISOMAP (§3.5), el cual es aplicado a espectros de resonancia magnética nuclear (RMN) de tumores cerebrales (conjuntos: LTE, STE, SLTE; §3.3.2), utilizando el arreglo *hardware/software* descrito en §3.2.

Nuestra implementación de ISOMAP, llamada aquí ISOMAP gMod, fue compara-

Conjunto de datos (# ítems)	**Caminos más cortos**	**Representación**	**Tiempo (s)**
Ecoli *(336)*	Dijkstra	Matriz de adyacencia	0.47
	Dijkstra + Fheaps	Matriz de adyacencia	1.21
	Floyd-Warshall	Matriz de adyacencia	0.60
	Dijkstra	Lista de adyacencia	0.67
	Dijkstra + Fheaps	Lista de adyacencia	1.57
	Floyd-Warshall	Lista de adyacencia	*0.44
German *(1 000)*	Dijkstra	Matriz de adyacencia	*12.85
	Dijkstra + Fheaps	Matriz de adyacencia	25.72
	Floyd-Warshall	Matriz de adyacencia	23.32
	Dijkstra	Lista de adyacencia	16.18
	Dijkstra + Fheaps	Lista de adyacencia	37.89
	Floyd-Warshall	Lista de adyacencia	19.27
Segmentation *(2 310)*	Dijkstra	Matriz de adyacencia	*186.55
	Dijkstra + Fheaps	Matriz de adyacencia	294.22
	Floyd-Warshall	Matriz de adyacencia	345.38
	Dijkstra	Lista de adyacencia	228.47
	Dijkstra + Fheaps	Lista de adyacencia	507.53
	Floyd-Warshall	Lista de adyacencia	192.38
Pageblocks *(5 473)*	Dijkstra	Matriz de adyacencia	*3 483.08
	Dijkstra + Fheaps	Matriz de adyacencia	3 955.05
	Floyd-Warshall	Matriz de adyacencia	10 867.04
	Dijkstra	Lista de adyacencia	5 549.91
	Dijkstra + Fheaps	Lista de adyacencia	7 678.91
	Floyd-Warshall	Lista de adyacencia	10 179.90
Pendigits *(10 992)*	Dijkstra	Matriz de adyacencia	−−
	Dijkstra + Fheaps	Matriz de adyacencia	−−
	Floyd-Warshall	Matriz de adyacencia	−−
	Dijkstra	Lista de adyacencia	131 085.17
	Dijkstra + Fheaps	Lista de adyacencia	*67 312.69
	Floyd-Warshall	Lista de adyacencia	193 720.78

Cuadro 4.4: Resultados de desempeño en tiempo para el cómputo de las distancias geodésicas (asumiendo un grafo no conexo al establecer $k = 1$) para varios conjuntos de datos UCI y diferentes configuraciones. Anotaciones como en la figura 4.3.

Conjunto de datos (# ítems)	**Lenguaje**	**Tiempo (s)**
Ecoli	C++	0.43
(336)	Matlab	8.60
German	C++	12.43
(1 000)	Matlab	220.98
Segmentation	C++	185.57
(2 310)	Matlab	2 479.50
Pageblocks	C++	3 621.90
(5 473)	Matlab	−−
Pendigits	C++	−−
(10 992)	Matlab	−−

Cuadro 4.5: Resultados de desempeño en tiempo para C++ vs Matlab para el cómputo de las distancias geodésicas usando el algoritmo de Dijkstra y una matriz de adyacencia.

da con las dos implementaciones de Tenenbaum et al. (2000), básica y por marcadores (*landmarks*), así como con el método PCA (§2.6.1), en términos del porcentaje de varianza explicada como función del número de características nuevas extraídas. Los resultados correspondientes se muestran en el cuadro 4.6.

Conjunto de datos (ítem × dim)	Método para reducción de dimensionalidad	% de varianza explicada para el número de características extraídas										# Var > 80 %	% (#Var)
		1	2	3	4	5	6	7	8	9	10		
LTE (195 x 195)	PCA	57.82	9.89	8.32	5.36	4.97	3.54	3.25	2.61	2.16	2.09	4	81.39
	ISOMAP	58.31	12.08	9.88	4.52	3.96	2.72	2.45	2.18	2.05	1.85	3	80.28
	ISOMAP Land	58.82	10.49	7.35	4.46	4.11	3.61	3.21	3.00	2.62	2.33	4	81.11
	ISOMAP gMod	80.50	9.06	3.50	2.25	1.19	1.02	0.76	0.66	0.59	0.46	1	80.50
STE (217 x 195)	PCA	66.88	7.68	6.58	5.74	3.71	2.64	2.18	1.80	1.41	1.38	3	81.14
	ISOMAP	67.05	8.38	7.86	4.70	3.12	2.30	2.00	1.65	1.55	1.39	3	83.29
	ISOMAP Land	66.42	7.42	6.58	4.26	3.16	2.92	2.70	2.45	2.18	1.92	3	80.42
	ISOMAP gMod	78.15	8.10	3.75	3.06	2.14	1.35	1.04	0.90	0.81	0.70	2	86.24
SLTE (195 x 390)	PCA	61.61	8.28	7.10	6.02	4.16	3.40	2.77	2.58	2.14	1.94	4	83.01
	ISOMAP	65.26	9.73	7.01	3.97	3.00	2.83	2.55	2.09	1.88	1.67	3	82.00
	ISOMAP Land	66.27	9.48	4.48	4.26	3.51	3.22	2.57	2.40	1.98	1.85	3	80.23
	ISOMAP gMod	75.28	13.22	4.53	1.76	1.32	1.00	0.88	0.77	0.68	0.55	2	88.50

Cuadro 4.6: Varianza explicada como función del número de características extraídas. Variantes de ISOMAP: Básico, Landmark (Land) y con el procedimiento propuesto (gMod).

Se puede observar que usar ISOMAP gMod ayuda a explicar un mayor porcentaje de la varianza de los datos, con muy pocas características extraídas con respecto a las implementaciones alternativas. Para el conjunto LTE (195 características correspondientes a frecuencias espectrales obtenidas en tiempo de eco largo), únicamente la primer característica extraída explica 80 % del total de la varianza. Así mismo,

para el conjunto de alta dimensionalidad SLTE (con 390 características), bastan dos características extraídas para explicar casi el 90 % de la varianza. A partir del cuadro 4.6, se puede notar que en todos los casos, ISOMAP gMod supera a las demás implementaciones usando este parámetro de evaluación.

Se realizó un experimento adicional, empleando versiones reducidas de los conjuntos a 20 características previamente seleccionadas, cuyos resultados se reportan en el cuadro 4.7, y que son consistentes con los resultados mostrados en el cuadro 4.6.

Conjunto de datos (ítem × dim)	Método para reducción de dimensionalidad	# Var > 80 %	% (#Var)
LTE (195 x 195)	PCA	6	80.85
	ISOMAP	6	80.84
	ISOMAP Land	8	81.73
	ISOMAP gMod	2	87.23
STE (217 x 195)	PCA	4	81.52
	ISOMAP	4	80.20
	ISOMAP Land	6	80.78
	ISOMAP gMod	2	83.19
SLTE (195 x 390)	PCA	6	80.64
	ISOMAP	6	82.17
	ISOMAP Land	6	80.27
	ISOMAP gMod	2	85.71

Cuadro 4.7: Condensado de la varianza explicada como función de las primeras 20 características extraídas. Anotaciones como en el cuadro 4.6.

4.4. Exactitud de clasificación a partir de datos reducidos

La clasificación diagnóstica de tumores cerebrales con base en espectros de resonancia magnética, es un problema no trivial donde la reducción de la dimensionalidad puede ser incluso obligatoria.

Aquí se compara el desempeño de algunas alternativas para la extracción de características, mediante la reducción de la dimensionalidad, en función de su exactitud para la clasificación de tumores cerebrales.

Primero, se realiza la reducción de la dimensionalidad, y una vez que los datos

son reducidos, se puede proceder a la clasificación. Un clasificador básico pero eficiente, y ampliamente utilizado en el área de análisis de datos médicos, es el Análisis de Discriminantes Lineal (LDA, del inglés *Linear Discriminant Analysis*). Su funcionamiento consiste en encontrar una combinación lineal de características que separe óptimamente diferentes tipos de clases de datos. Los métodos de reducción de la dimensionalidad utilizados son, el método lineal PCA (§2.6.1), y el no lineal ISOMAP (§2.6.2), aplicados a espectros de RMN de tumores cerebrales (§3.3.2). ISOMAP es utilizado en las tres variantes anteriormente presentadas, básica y por marcadores de Tenenbaum et al. (2000), y gMod desarrollada usando MMLL y descrita en §3.5.

Estos experimentos tienen dos objetivos, primero, mostrar cómo la forma en que se construye la variedad en ISOMAP, afecta la exactitud de la clasificación. Segundo, evaluar los resultados de la clasificación en términos del número de características extraídas y su correspondiente exactitud. En todos los experimentos se empleó $k = 10$ en la regla-k. Los experimentos de clasificación realizados con los tipos de tumores, agrupan dichos tumores en tres grandes grupos, G1: gliomas de bajo grado (a2, oa y od); G2: tumores malignos de alto grado (me y dl); y G3: meningiomas (mm). Este tipo de agrupamiento está justificado en Tate et al. (2003) y Vellido et al. (2009), por la bien conocida dificultad de distinguir entre metastases y glioblastomas, debido a la similitud de sus patrones espectrales.

Los resultados de exactitud de la clasificación promedio fueron validados por medio de una validación cruzada de diez ciclos. Los resultados de clasificación de LDA para G1, G2 y G3 usando los espectros STE, LTE y STE+LTE, se resumen en las figuras 4.1, 4.2 y 4.3.

Los resultados de exactitud obtenidos mediante las diferentes versiones de ISOMAP, difieren claramente. Estas diferencias son el resultado de las diversas formas de construir la variedad subyacente (representación del grafo), correspondientes a las versiones del algoritmo. El algoritmo ISOMAP básico, toma el componente (subgrafo) más grande cuando el grafo resultante, construido mediante la regla-k, está desconectado. En cambio, ISOMAP gMod siempre asegura un grafo conexo, usando un procedimiento modificado para hallar el árbol de expansión mínima. Finalmente, ISOMAP landmark selecciona aleatoriamente l marcadores del conjunto original y construye el grafo a partir de ellos. Por tanto, la matriz de distancias de grafo resultante depende completamente de cómo se haya realizado la construcción del grafo. Puesto que ISOMAP emplea la matriz de distancias de grafo como entrada para el método MDS, el cual calcula las coordenadas de los puntos en el espacio de

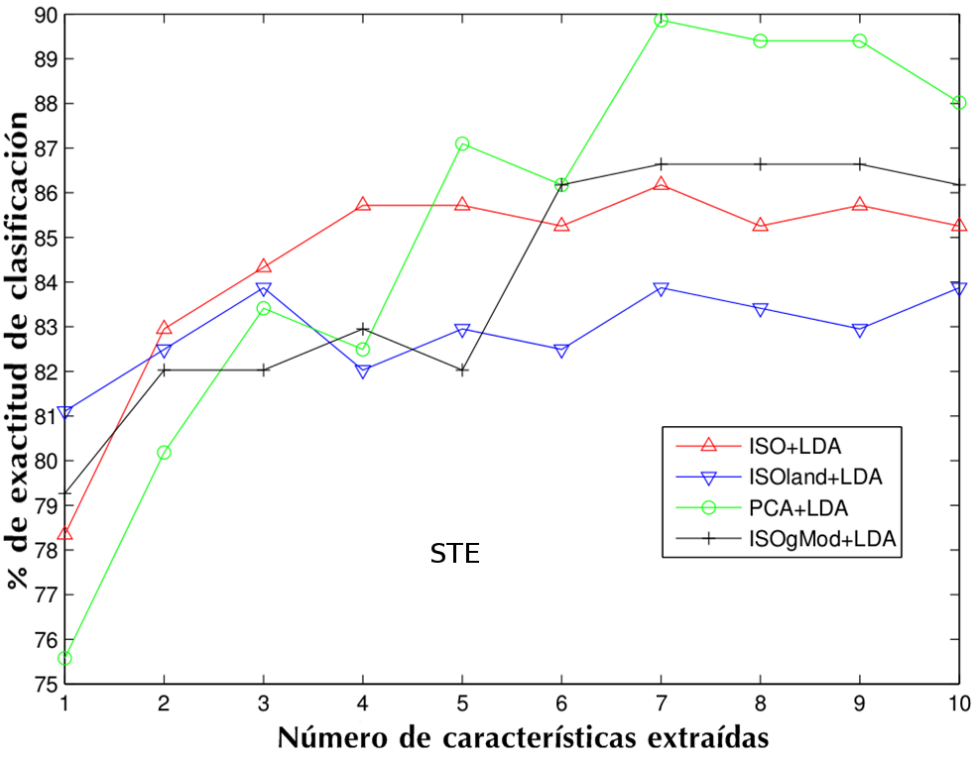

Figura 4.1: Resultados de clasificación LDA para G1, G2 y G3 usando características extraídas de los espectros STE. Variantes de ISOMAP: básico (ISO), landmark (ISOland) y con el módulo optimizado (ISOgMod).

proyección de baja dimensión, las características extraídas difieren notablemente, y por ello, los resultados de exactitud de clasificación difieren entre sí.

Aunque los resultados previos de la §4.3 muestran que ISOMAP gMod puede preservar un gran porcentaje de varianza de los datos con pocas características extraídas, es claro, a partir de los resultados mostrados en las figuras 4.2 y 4.3, que esta preservación de varianza no garantiza un correspondiente buen nivel de exactitud en una clasificación posterior. Así, cuando el número de características extraídas es pequeño, las implementaciones de ISOMAP no proveen una mejora significativa en exactitud de LDA, respecto a PCA. Para cinco o más características, PCA supera claramente a ISOMAP. Las variantes de ISOMAP sólo superan a PCA cuando se usan los datos de dos tiempos de eco diferentes combinados (ver figura 4.3). Una representación extremadamente condensada formada por sólo 3 características es suficiente para obtener un promedio apenas abajo del 90 %. En todos los casos, estos resultados de exactitud de clasificación son consistentes con los reportados en

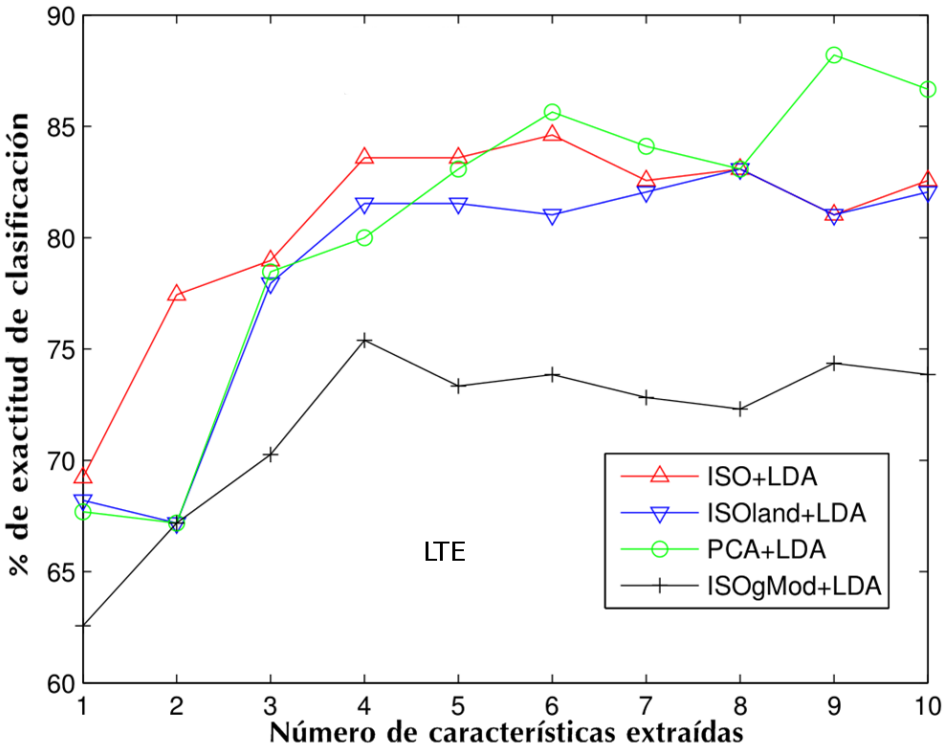

Figura 4.2: Resultados de clasificación LDA para G1, G2 y G3 usando características extraídas de los espectros LTE. Las anotaciones son como en la figura 4.1.

la literatura. Por ejemplo, en García-Gómez et al. (2008), el uso de los espectros STE+LTE con una combinación de PCA+LDA alcanza una exactitud de clasificación de alrededor del 90 %, pero utilizando un mínimo de 8 componentes principales. Los resultados de clasificación para ISOMAP gMod se deterioran significativamente en presencia de espectros LTE.

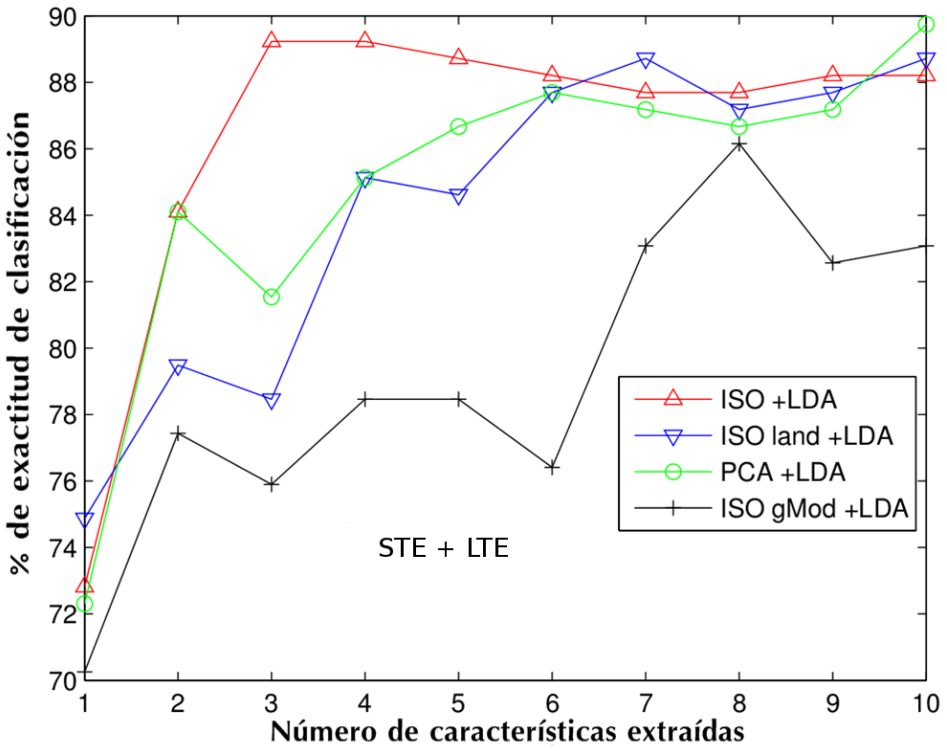

Figura 4.3: Resultados de clasificación LDA para G1, G2 y G3 usando características extraídas de los espectros STE+LTE. Las anotaciones son como en la figura 4.1.

Capítulo 5

Conclusiones y perspectivas

5.1. Conclusiones

La distancia geodésica es una métrica que ha demostrado ser de gran importancia en los modelos de aprendizaje automático, en particular para la reducción de la dimensionalidad no lineal. El cálculo de la distancia geodésica no es trivial y comúnmente se requiere de una aproximación por distancias de grafo. Las características de la implementación del *software* para realizar dicho cálculo tiene un impacto considerable en términos de los requerimientos computacionales, así como también en los resultados finales.

En el presente proyecto de tesis se desarrolló una biblioteca de aprendizaje automático, denominada MMLL, con el fin de proveer las herramientas necesarias para realizar el cómputo de las distancias geodésicas para conjuntos de datos grandes. La biblioteca MMLL, desarrollada en lenguaje C++, incorpora diferentes alternativas para la representación del grafo, así como diferentes algoritmos para hallar los caminos más cortos.

El procedimiento para realizar el cómputo de las distancias geodésicas (CDG), fue evaluado usando diferentes conjuntos de datos del repositorio de la UCI, cuyo número de elementos se incrementó progresivamente. Los resultados experimentales muestran que el uso de la matriz de adyacencia y el algoritmo de Dijkstra es recomendable para el CDG de conjuntos pequeños y medianos. Cuando el número de elementos en el conjunto se incrementa, el uso de la lista de adyacencia como representación de grafo se vuelve crucial para el cómputo. También se muestra que para conjuntos más grandes, el uso de la lista de adyacencia y el algoritmo de Dijkstra con F-heaps

produce mejor desempeño en tiempo que cualquier otra representación de grafo y algoritmo de caminos más cortos. Por otro lado, cabe destacar que el uso de la matriz dispersa como representación de una matriz de adyacencia, no provee ventajas, ni en tiempo, ni en almacenamiento computacional.

También, a partir de los resultados obtenidos por las implementaciones del procedimiento para el CDG, desarrolladas en C++ y Matlab, es claro que la implementación en C++ para este procedimiento usando MMLL, es mucho más rápido que su correspondiente en Matlab.

El método para la reducción de la dimensionalidad no lineal, ISOMAP, fue implementado usando el procedimiento optimizado y fue empleado para analizar algunos conjuntos de datos de espectros RMN, correspondientes a tumores cerebrales. Dichos conjuntos son pequeños, pero de alta dimensionalidad. En este tipo de problemas, el diagnóstico de tales tumores es de gran importancia, y dicha interpretación puede ser parcialmente facilitada mediante procedimientos de reducción de la dimensionalidad. Los resultados experimentales muestran que ISOMAP gMod supera a las demás alternativas en términos del porcentaje explicado de la varianza de estos datos con muy pocas características extraídas.

El diagnóstico de tumores cerebrales es un desafío en el área médica. Algunos métodos de aprendizaje automático e inteligencia artificial, pueden asistir a los médicos y expertos en estas tareas. La clasificación de tumores cerebrales con base en conjuntos de datos de alta dimensionalidad, permiten que el uso de métodos para reducción de dimensionalidad sea adecuado. Técnicas de aprendizaje de variedades usando distancias geodésicas han demostrado ser una alternativa muy prometedora en este tipo de tareas, y por ello, se han realizado esfuerzos por optimizar el uso intensivo de los recursos computacionales.

ISOMAP gMod además de realizar el CDG de manera rápida, describe la varianza de los datos apropiadamente. A partir de ello, se realiza la clasificación de tumores cerebrales usando las características extraídas como entrada. Los resultados experimentales han mostrado evidencia de que el modo de construir la variedad subyacente puede comprometer la exactitud de la clasificación. La implementación básica de ISOMAP provee, para algunos casos, exactitud en sus resultados comparable y mejor que la obtenida por PCA, utilizando menos características extraídas, especialmente cuando se combinan diferentes tiempos de eco.

El manejo de conjuntos de datos grandes, es una necesidad, toda vez que las ciencias emplean grandes volúmenes de datos obtenidos del mundo real. Sin embargo,

algunas limitaciones impuestas por el sistema operativo, impiden el manejo de conjuntos muy grandes, lo que se debe tener en cuenta al elegir un ambiente de trabajo para el CDG^2.

5.2. Perspectivas

Algunas alternativas para extender el presente trabajo, que se pueden realizar a futuro, tomando las experiencias del presente como base, se listan a continuación:

- Inserción del módulo para el cómputo de las distancias geodésicas (CDG) desarrollado, en otros métodos de aprendizaje automático. Aquí se propuso emplear el CDG, en el algoritmo ISOMAP. Sin embargo, queda abierta la posibilidad de ser usado con otros métodos de reducción de la dimensionalidad no lineal que utilicen la distancia geodésica como base, o bien, a otras técnicas de aprendizaje automático, como el algoritmo de propagación de etiquetas (Zhu y Ghahramani, 2002) para mejorar su desempeño al trabajar con conjuntos con geometrías intrincadas.

- Experimentación con otras alternativas en el proceso del cómputo de las distancias geodésicas. Por ejemplo, el uso de otra estructura de datos como cola de prioridad para los algoritmos de Dijkstra y Prim, o el uso de la regla-ϵ para iniciar la construcción del grafo.

- Programación multiproceso. Se sugiere el uso de programación multiproceso para poder manejar archivos (conjuntos de datos) más grandes, de los presentados aquí.

- Extensión de biblioteca. Se propone agregar más métodos de aprendizaje automático a la biblioteca MMLL para continuar con su desarrollo, hasta ser una opción real entre la comunidad interesada.

Bibliografía

L. Baoli, Y. Shiwen, and L. Qin. An improved k-nearest neighbor algorithm for text categorization. *CoRR*, cs.CL/0306099, 2003.

D. Bautista-Villavicencio and R. Cruz-Barbosa. On geodesic distance computation: An experimental study. In *11th Mexican International Conference in Computer Science (ENC 2011), Advances in Computer Science and Applications, Research in Computing Science*, volume 53, pages 115–124, 2011.

M. Belkin and P. Niyogi. Using manifold structure for partially labelled classification. In *NIPS*, pages 929+, 2002.

M. Bernstein, V. de Silva, J. C. Langford, and J. B. Tenenbaum. Graph approximations to geodesics on embedded manifolds. Technical report, Stanford University, CA, U.S.A., 2000.

D. Bommes and L. Kobbelt. Accurate computation of geodesic distance fields for polygonal curves on triangle meshes, 2007.

I. N. Bronshtein, K. A. Semendyayev, G. Musiol, and H. Muehlig. *Handbook of Mathematics*. Springer-Verlag, 2005.

V. Bryant. *Metric Spaces: Iteration and Application*. Cambridge University Press, 1994.

L. Chun-Guang, G. Jun, and Z. Hong-Gang. Pruning neighborhood graph for geodesic distance based semi-supervised classification. In *Proceedings of the 2007 International Conference on Computational Intelligence and Security*, CIS '07, pages 428–432, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-3072-9.

T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. McGraw-Hill, 1990.

R. Cruz-Barbosa. *Generative Manifold Learning for the Exploration of Partially Labeled Data*. PhD thesis, Universitat Politècnica de Catalunya, Spain, 2009.

R. Cruz-Barbosa and A. Vellido. *Geodesic Generative Topographic Mapping*, volume 5290 of *LNAI*, pages 113–122. Springer-Verlag, 2008.

R. Cruz-Barbosa and A. Vellido. Semi-supervised geodesic generative topographic mapping. *Pattern Recognition Letters*, pages 202–209, 2010.

R. Cruz-Barbosa and A. Vellido. Semi-supervised analysis of human brain tumours from partially labeled MRS information, using manifold learning models. *International Journal of Neural Systems*, 21(1):17–29, 2011.

R. Cruz-Barbosa, D. Bautista-Villavicencio, and A. Vellido. On the computation of the geodesic distance with an application to dimensionality reduction in a neuro-oncology problem. In *16th Iberoamerican Congress on Pattern Recognition (CIARP 2011)*, 2011a. Aceptado para publicación.

R. Cruz-Barbosa, D. Bautista-Villavicencio, and A. Vellido. Comparative diagnostic accuracy of linear and nonlinear feature extraction methods in a neuro-oncology problem. In J. Martínez-Trinidad, J. Carrasco-Ochoa, C. Ben-Youssef Brants, and E. Hancock, editors, *Pattern Recognition - 3rd Mexican Conference in Pattern Recognition (MCPR 2011)*, volume 6718 of *Lecture Notes in Computer Science*, pages 34–41. Springer Berlin / Heidelberg, 2011b. ISBN 978-3-642-21586-5.

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

A. Frank and A. Asuncion. UCI machine learning repository, 2010. URL http://archive.ics.uci.edu/ml.

M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987. ISSN 0004-5411.

J. M. García-Gómez, S. Tortajada, C. Vidal, M. Julià-Sapé, J. Luts, A. Moreno-Torres, S. Van Huffel, C. Arús, and M. Robles. The effect of combining two echo

times in automatic brain tumor classification by MRS. *NMR in Biomedicine*, 21 (10):1112–1125, 2008.

G. H. Golub and C. F. Van Loan. *Matrix computations*. The Johns Hopkins University Press, 1996.

S. I. Grossman. *Linear Algebra*. McGraw Hill, 1996.

J. Hua, Z. Lai, M. Dong, X. Gu, and H. Qin. Geodesic distance-weighted shape vector image diffusion. In *IEEE Transactions on Visualization and Computer Graphics*, volume 14, pages 1643–1650. IEEE Computer Society, 2008.

I. T. Jolliffe. *Principal Component Analysis*. Springer, 2002.

M. Julià-Sapé, D. Acosta, M. Mier, C. Arús, D. Watson, and the INTERPRET Consortium. A multi-centre, web-accessible and quality control-checked database of in vivo MR spectra of brain tumour patients. *Magn Reson Mater Phys*, 19:22–33, 2006.

R. Koch. Notes of mathematics. http://chaos.swarthmore.edu/courses/Phys130_2004/LectureNotes/Koch_DiffGeom.pdf, 2002. [Accessed: 06-10-2009].

J. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964a.

J. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(2):115–129, 1964b.

J. A. Lee and M. Verleysen. How to project 'circular' manifolds using geodesic distances? In *Proceedings of the 12th European Symposium on Artificial Neural Networks (ESANN 2004)*, pages 223–230, 2004.

J. A. Lee and M. Verleysen. *Nonlinear Dimensionality Reduction*. Springer, 2007.

J. A. Lee, A. Lendasse, and M. Verleysen. Curvilinear distance analysis versus isomap. In *Proceedings of the 10th European Symposium on Artificial Neural Networks (ESANN 2002)*, pages 185–192, 2002.

J. M. Lee. *Introduction to Topological Manifolds*. Springer, 2000.

G. McCarty. *Topology: An Introduction with Application to Topological Groups*. Courier Dover Publications, 1988.

J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou. The discrete geodesic problem. In *SIAM Journal on Computing archive*, pages 647–668. Society for Industrial and Applied Mathematics Philadelphia, PA, USA, 1987.

W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes - The Art of Scientific Computing*. Cambridge University Press, 2007.

R. C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, pages 1389–1401, 1957.

D. Purves. *Neuroscience*. Sinauer Associates, Inc., 2007.

S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *SCIENCE*, 290:2323–2326, December 2000.

S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003.

J. K. Seong, W. K. Jeong, and E. Cohen. Anisotropic geodesic distance computation for parametric surfaces. In *IEEE Int. Conf. on Shape Modeling ans Applications*, pages 179–186, 2008.

G. Shakhnarovich, T. Darrell, and P. Indyk. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice (Neural Information Processing)*. The MIT Press, 2006. ISBN 026219547X.

C. Shi, S. Zhang, Z. Zheng, and Z. Shi. Geodesic distance based som for image clustering. In *Proceedings of the International Conference of Sensing, Computing and Automation*, pages 2483–2488. Watam Press, 2006.

J. Shlens. A tutorial on principal component analysis. http://www.snl.salk.edu/ shlens/pub/notes/pca.pdf, December 2005. URL http://www.snl.salk.edu/~shlens/pub/notes/pca.pdf.

V. De Silva and J. B. Tenenbaum. Global versus local methods in nonlinear dimensionality reduction. In *Advances in Neural Information Processing Systems 15*, volume 15, pages 705–712, 2003.

S. S. Skiena. *The Algorithm Design Manual.* Springer-Verlag, 2008.

L. I. Smith. A tutorial on principal components analysis. Technical report, Cornell University, USA, February 2002.

V. Surazhsky, T. Surazhsky, D. Kirsanov, S. J. Gortler, and H. Hoppe. Fast exact and approximate geodesics on meshes. In *Proceedings of ACM SIGGRAPH 2005, ACM Transactions on Graphics*, pages 553–560. ACM New York, NY, USA, 2005.

A. R. Tate, C. Majós A., Moreno, F. A. Howe, J. R. Griffiths, and C. Arús. Automated classification of short echo time in In vivo ^{1}H brain tumor spectra: a multicenter study. *Magn Res Med*, 49:29–36, 2003.

J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *SCIENCE*, 290:2319–2323, December 2000.

R. P. Tewarson. *Sparse Matrices.* Acedemic Press, 1973.

W. Torge. *Geodesy.* Walter de Gruyter, 2001.

A. Unwin, M. Theus, and H. Hofmann. *Graphics of Large Datasets.* Springer, 2006.

O. Veblen and J. Whitehead. *The Foundations of Differential Geometry.* Cambridge University Press, 1932.

A. Vellido, E. Romero, F. F. González-Navarro, L. A. Belanche Muñoz, M. Julià-Sapé, and C. Arús. Outlier exploration and diagnostic classification of a multi-centre ^{1}H-MRS brain tumour database. *Neurocomputing*, 72(13-15):3085–3097, 2009.

J. H. Wilkinson. Householder's method for symmetric matrices. *Numerische Mathematik*, 4:354–361, 1962. ISSN 0029-599X.

H. Yin. *Nonlinear Principal Manifolds - Adaptaive Hybrid Learning Approaches*, volume 5290 of *LNAI*, pages 15–29. Springer-Verlag, 2008.

X. Zhu and Z. Ghahramani. Learning from labeled and unlabeled data with label propagation. Technical report, Carnegie Mellon University, 2002.

Anexo A

Pseudocódigo de algoritmos utilizados en la tesis

A continuación, se presentan los pseudocódigos de algunos de los algoritmos empleados en esta tesis.

A.1. Algoritmo de Dijkstra

Algoritmo A.1: Algoritmo de Dijkstra

Entrada: Un grafo G, con vértices V y aristas E, y los pesos w entre cada par de nodos.

Salida: Los caminos más cortos entre el nodo origen y todos los demás nodos en V.

1 **para cada** $v \in V$ **hacer**

   (a) $d[v] \leftarrow \infty$;

   (b) $\pi[v] \leftarrow NULL$;

2 **fin-para-cada**

3 $d[s^a] \leftarrow 0$;

4 $S \leftarrow \emptyset$;

5 **mientras** $S \neq V$ **hacer**

   (a) Elegir el nodo $u \in V - S$, tal que $d[u]$ sea la mínima.

   (b) Añadir u a S.

   (c) Actualizar el peso de todos los nodos $v \in V - S$, que se alcanzan desde u.
   si $d[v] > d[u] + w(u, v)$ **entonces**

      (I) $d[v] \leftarrow d[u] + w(u, v)$;

      (II) $\pi[v] \leftarrow u$;

   fin-si

6 **fin-mientras**

[a] El nodo $s \in V$, es el nodo origen y es elegido de forma arbitraria.

A.2. Algoritmo de Floyd-Warshall

Algoritmo A.2: Algoritmo de Floyd-Warshall

Entrada: Un grafo ponderado y conexo $G = (V, E)$, donde V es un conjunto finito de n vértices (nodos), y E es una colección de arcos que conectan a pares de vértices.

Salida: Los caminos más cortos entre todos los pares de nodos en V.

```
1 Inicializar la matriz path, de tal forma que path[i][j] = weight(i, j), donde
  weight(i, j) regresa el peso del arco de i a j;
2 para cada k = 1...n hacer
3 |   para cada i = 1...n hacer
4 |   |   para cada j = 1...n hacer
5 |   |   |   path[i][j] = min(path[i][j], path[i][k] + path[k][j]);
6 |   |   fin-para-cada
7 |   fin-para-cada
8 fin-para-cada
```

A.3. K-vecinos más cercanos

Algoritmo A.3: K-vecinos más cercanos

Entrada: El valor de k y un conjunto de datos numéricos (m instancias, cada una con n atributos)

Salida: Un grafo representado mediante una matriz de adyacencia

```
1  Crear matriz de adyacencia (matriz cuadrada de m × m) e inicializar a 0;
2  Crear vector de distancias (vector de tamaño m) e inicializar a ∞;
3  para i ← 0, 1, ..., m − 1 hacer
4      para j ← 0, 1, ..., m − 1 hacer
5          si i ≠ j entonces
6              Calcular distancia de elemento i a elemento j y almacenar en vector
               de distancias en la posición j
7          sino
8              Almacenar ∞ en vector de distancias en la posición j
9          fin-si
10     fin-para
11     Ordenar vector de distancias y actualizar matriz de adyacencia en función
       de k
12 fin-para
```

A.4. Algoritmo de Prim

Algoritmo A.4: Algoritmo de Prim

Entrada: Un grafo ponderado y conexo, con vértices V y aristas E.

Salida: Los conjuntos V_{mst} y E_{mst} que conforman el MST.

1 $V_{mst} = s$[a];

2 $E_{mst} = \emptyset$;

3 **mientras** $V_{mst} \neq V$ **hacer**

(a) Elegir la arista $(u, v) \in E$ con peso mínimo tal que, $u \in V_{mst}$ y $v \notin V_{mst}$ (si existen múltiples aristas que cumplan la condición, elegir arbitrariamente).

(b) Añadir v a V_{mst}, añadir (u, v) a E_{mst}.

4 **fin-mientras**

[a] El nodo $s \in V$, es el nodo origen y es elegido de forma arbitraria.

Anexo B

Estructuras de datos

B.1. Montículos de Fibonacci

Un montículo (*heap*, en inglés) es una estructura de datos basada en árboles, que consiste en un conjunto de ítems, cada uno con una llave con valor real k, y que satisface la propiedad de montículo: si B es el nodo hijo de A, entonces $k(A) \leq k(B)$. Esto implica que el elemento menor, es decir, aquel con la llave menor, es siempre el nodo raíz (por esto algunas veces el montículo es también llamado *min-heap*[1]). Los montículos son una estructura de datos simple y elegante que soporta eficientemente las operaciones básicas del tipo de datos abstracto denominado "cola de prioridad": Insertar() y Extraer-Menor(). Las colas de prioridad implementadas con montículos son útiles en muchas aplicaciones, particularmente, son cruciales en muchas implementaciones de algoritmos de grafos, para hacerlos eficientes (Fredman y Tarjan, 1987; Cormen et al., 1990; Skiena, 2008).

Un montículo se encuentra sujeto a las siguientes operaciones:

- Crear montículo: Regresa un montículo nuevo y vacío.
- Insertar(i, h): Inserta un ítem i nuevo con llave previamente definida, en el montículo h.
- Encontrar mínimo(h): Regresa un ítem con la llave mínima, del montículo h. Esta operación no realiza cambios sobre h.

[1] De forma alternativa, si el criterio de comparación es inverso, es decir que el elemento mayor se encuentre siempre en el nodo raíz, resulta un *max-heap*

- Extraer menor(h): Elimina un ítem con la llave mínima, del montículo h y lo devuelve.

Asimismo, las operaciones siguientes frecuentemente son útiles al trabajar con montículos:

- Mezclar(h_1, h_2): Regresa el montículo formado al unir los montículos h_1 y h_2 con ítems disjuntos. Esta operación destruye h_1 y h_2.

- Decrementar llave(h, i, Δ): Decrementa la llave del ítem i en el montículo h asignándole el número no negativo Δ. Esta operación asume que la posición de i en h es conocida.

- Eliminar(i, h): Elimina arbitrariamente el ítem i del montículo h. Esta operación asume que la posición de i en h es conocida.

Los montículos de Fibonacci o *F-heaps* (de *Fibonacci heaps*), son una colección de árboles basada en montículos. El trabajo de mantener la estructura puede ser demorado hasta que sea conveniente realizar dicha operación. Desde un punto de vista teórico, los montículos de Fibonacci son deseables cuando el número de operaciones Extraer menor(h) y Eliminar(i, h), es pequeña comparada con las demás operaciones. Al igual que otros tipos de montículos, los F-heaps no están optimizados para realizar búsquedas entre sus elementos.

Para algunos algoritmos, los montículos de Fibonacci pueden ser utilizados para mejorar el tiempo de ejecución asintótico; por ejemplo, el algoritmo de Prim para hallar el árbol de expansión mínima y el algoritmo de Dijkstra para hallar las rutas más cortas en un grafo (Fredman y Tarjan, 1987; Cormen et al., 1990).

Anexo C

Manual de usuario del software

El código fuente correspondiente a la aplicación *geodistances*, que realiza el cómputo de las distancias geodésicas para un conjunto de datos dado, fue desarrollado en C++ y usando MMLL. A partir de dicho código, se generaron ejecutables para GNU/Linux y MS Windows, sin embargo, es posible crear ejecutables para otras plataformas. Los ejecutables se probaron exitosamente en Ubuntu (x86) 9.04, 9.10 y 11.04, y Windows XP (x64) y 7 (x86).

A continuación se describe el proceso de instalación y ejecución de la aplicación en Ubuntu. Para otros sistemas operativos, el proceso puede variar un poco.

C.1. Instalación de la aplicación

Al tener disponibles los archivos ejecutables de *geodistances*, no es necesario realizar ningún proceso de compilación, y por lo tanto, tampoco se requiere la instalación del compilador o del entorno de desarrollo. Tampoco hay que instalar algún otro *software*, puesto que la aplicación no tiene dependencia de aplicaciones de terceros. Basta con tener el archivo ejecutable en una ubicación conocida. En esta guía supondremos que el archivo ejecutable se encuentra en la carpeta personal del usuario.

C.2. Ejecución de la aplicación

La ejecución de la aplicación se realiza desde la línea de comandos, por lo que es preciso abrir una terminal y situarse en el directorio donde se encuentra el archivo

ejecutable.

Al abrir la terminal aparece el *prompt* del sistema:

```
user@pc:~$
```

A continuación, para situarse en la carpeta personal del usuario, se escribe:

```
user@pc:~$ cd ~
```

y al final de este y los comandos subsecuentes se presiona la tecla *Enter*.

Ahora, se invoca la aplicación *geodistances* con los parámetros correspondientes al conjunto de datos a utilizar. Por simplicidad, se recomienda que el conjunto de datos se encuentre en la misma ruta donde se encuentra la aplicación.

Para ejecutar *geodistances* se escribe en la terminal:

```
user@pc:~$ ./geodistances input_dataset lbl-class_position k
```

donde:

- `input_dataset`, es el nombre y la ruta del archivo que contiene al conjunto de datos.
- `lbl-class_position`, determina el formato del archivo de entrada, como sigue:
  - 0: no hay etiqueta de clase.
  - 1: la etiqueta de clase es la primera entrada de cada línea.
  - 2: la etiqueta de clase es la última entrada de cada línea.
- `k`, es el valor del parámetro k para el algoritmo $k$nn.

Cabe destacar que existen diferentes versiones de la aplicación *geodistances*. La diferencia entre cada una de ellas, reside en la combinación de representación de grafo y algoritmo de caminos más cortos, que emplea para realizar el cómputo de las distancias geodésicas.

Así, se tienen las siguientes versiones:

Versión	Combinación utilizada
geodistances-mdjk	Matriz de adyacencia y algoritmo de Dijkstra
geodistances-mdfh	Matriz de adyacencia y algoritmo de Dijkstra con *montículos de Fibonacci*
geodistances-mfw	Matriz de adyacencia y algoritmo de Floyd-Warshall
geodistances-ldjk	Lista de adyacencia y algoritmo de Dijkstra
geodistances-ldfh	Lista de adyacencia y algoritmo de Dijkstra con *montículos de Fibonacci*
geodistances-lfw	Lista de adyacencia y algoritmo de Floyd-Warshall

Por tanto, cuando nos referimos a la aplicación como *geodistances*, es en generalización de cualquiera de las versiones listadas previamente. Los comandos que se explican aquí, aplican para cualquiera de las versiones.

Para ejemplificar su uso, se utilizará el conjunto de datos *Ecoli* (descrito en §3.3.1) que está almacenado en el archivo *ecoli.data*, y que se encuentra en la carpeta personal del usuario.

Para calcular la distancia geodésica para el conjunto de datos *ecoli.data* usando *geodistances* con la combinación matriz de adyacencia y algoritmo de Dijkstra, se escribe en la terminal:

```
user@pc:~$ ./geodistances-mdjk ecoli.data 1 10
```

donde:

- `./geodistances-mdjk`, invoca la aplicación para el cómputo de las distancias geodésicas, usando la combinación de matriz de adyacencia como representación de grafo, y el algoritmo de Dijkstra como algoritmo de caminos más cortos.
- `ecoli.data`, le indica a la aplicación que el archivo que contiene al conjunto de datos de entrada se llama `ecoli.data`, y que se encuentra en el mismo directorio que la aplicación.
- `1`, le indica a la aplicación que la etiqueta de clase es la primera entrada de cada línea del archivo.
- `10`, le indica a la aplicación que conecte cada nodo con sus 10 vecinos más cercanos.

Después de un tiempo (en función al tamaño del conjunto), el *prompt* nos mostrará lo siguiente:

```
user@pc:~$ ./geodistances-mdjk ecoli.data 1 10
<./geodistances-mdjk>   Copyright (C) 2010  David Bautista Vvcio.
This program is free software: you can redistribute it and/or
modify it under the terms of the GNU General Public License
version 3 as published by the Free Software Foundation.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.


Geodesic distances computation finished..!
```

para indicar que la aplicación ha terminado.

Al finalizar su ejecución, la aplicación genera el archivo *geoD_mtx.temp*, que almacena la matriz de distancias geodésicas.

Ayuda

La aplicación integra una breve ayuda sobre su utilización. Para acceder a ella, basta con ejecutar la aplicación sin ningún parámetro:

```
user@pc:~$ ./geodistances
```

A continuación, después del mensaje correspondiente a la licencia, aparecerá la ayuda:

```
Use: "./geodistances input_dataset [lbl-class_position] [k]"
where:
lbl-class_position, determines the 'input_dataset' file's
  format as follows:
   0 - no label class (default)
   1 - label class is the first entry of each line
   2 - label class is the last entry of each line
k, is the K value (integer) for knn. (default k=1)
```

donde se resume lo indicado previamente.

Anexo D

Glosario

Ávido

Se dice que un algoritmo es ávido (*greedy*, en inglés), cuando en alguno de sus pasos se debe elegir una de entre muchas alternativas, donde, la estrategia ávida indica que se debe elegir la que se presente como mejor opción en ese momento. Dicha estrategia no es recomendada de manera general para encontrar soluciones óptimas globales, sin embargo, existen algunos problemas donde ésta estrategia sí provee la solución óptima global (Cormen et al., 1990).

Espacio de encaje

Un espacio de encaje (*embedding space*), es una representación de un objeto topológico (una variedad, una gráfica) en cierto espacio, usualmente $\mathbb{R}^D$ para algún D, de tal manera que sus propiedades topológicas se preserven. De manera más general, un espacio $\mathcal{X}$ se encuentra encajado en otro espacio $\mathcal{Y}$ cuando las propiedades de $\mathcal{Y}$ restringidas a $\mathcal{X}$ son las mismas propiedades de $\mathcal{X}$ (Lee y Verleysen, 2007).

Espacio topológico

Un espacio topológico es un conjunto para el cual se especifica una topología. Para un conjunto $\mathcal{Y}$, se define una topología T como una colección de subconjuntos de $\mathcal{Y}$ que cumplen las siguientes propiedades:

- De forma trivial, $\emptyset \in T$ y $\mathcal{Y} \in T$.

- Para cualesquiera dos conjuntos que se encuentren en T, también su intersección lo está.
- Para cualesquiera dos conjuntos que se encuentren en T, también su unión lo está.

Esta definición de topología se satisface tanto para el espacio Cartesiano ($\mathbb{R}^D$) como para grafos (Lee y Verleysen, 2007).

Topología

En matemáticas, la topología estudia las propiedades de los objetos, que se conservan a pesar de sufrir deformaciones, torceduras y estiramientos. En un sentido más general, aquellas operaciones que garantizan que la estructura intrínseca o conectividad de los objetos no es alterada.

Una de las ideas centrales de la topología es que el conocimiento relativo a los objetos no depende de cómo son representados, o encajados, en el espacio. En otras palabras, la topología es usada para abstraer la conectividad intrínseca de los objetos aunque se ignoren los detalles de su forma. Si dos objetos comparten las mismas propiedades topológicas, se dice que son homeomorfos. Los "objetos" de la topología son definidos formalmente como espacios topológicos (Lee y Verleysen, 2007).

Variedad

Una variedad topológica (*manifold*, en inglés) $\mathcal{M}$, es un espacio topológico que es Euclideano localmente, lo que significa que alrededor de cada punto de $\mathcal{M}$ existe una vecindad que es topológicamente la misma que la de la bola unitaria abierta en $\mathbb{R}^D$ (Lee y Verleysen, 2007).

Informalmente, se puede pensar en una variedad como una curva o superficie, pero posiblemente de más alta dimensión. Cada variedad tiene una dimensión, la cual se puede decir, es el número parámetros independientes que se necesitan para especificar un punto. El prototipo de una variedad n-dimensional es el espacio Euclideano n-dimensional $\mathbb{R}^n$, para el cual cada punto es una n-tupla de números reales.

Una variedad n-dimensional es un objeto modelado localmente en $\mathbb{R}^n$; lo que significa que tiene exactamente n números para especificar un punto, al menos si no nos desviamos demasiado del punto de inicio dado.

Las variedades de dimensión 1 son comúnmente denominadas *curvas* (aunque no es una condición necesaria serlo). El ejemplo más simple es la línea real. Otros ejemplos son los círculos, parábolas, o la gráfica de cualquier función de la forma $y = f(x)$, entre otros. Para cada uno de estos ejemplos, se puede especificar un punto único por un único número real, sin ambigüedad.

Las variedades de dimensión 2 por su parte, son superficies. Los ejemplos más comunes son los planos y las esferas (nótese que en este contexto, el término esfera, denota una superficie esférica (la cual es bidimensional), y no así, a una bola sólida (que es tridimensional)). Otros ejemplos de superficies incluye cilindros, elipsoides, paraboloides, entre otros. En estos casos, se necesitan dos coordenadas para determinar un punto. Normalmente, en el plano se usan coordenadas Cartesianas o polares, mientras que en una esfera, latitud y longitud.

La única variedad de alta dimensión que podemos visualizar es el espacio Euclideano, o comúnmente llamada tercera dimensión (3D) (Lee, 2000).

Acerca de...

El documento

Este documento fue creado con LaTeX 2_ε, usando el editor *Texmaker* bajo la plataforma GNU/Linux. Los paquetes empleados fueron: *natbib.sty*, para las referencias bibliográficas; *algorithm2e.sty*, para los algoritmos; *hyperref.sty* y *xcolor.sty*, para los enlaces y el manejo de colores respectivamente, en el documento digital; y *fancyhdr.sty*, para los encabezados y pies de página.

Una copia electrónica de este documento puede ser obtenida de la biblioteca digital de la Universidad Tecnológica de la Mixteca, en: http://biblioteca.utm.mx/biblio_dig.php

Las figuras

La figura 2.1, fue creada con Inkscape. La figura 2.2, fue creada con Gimp. Las figuras 3.1, 3.2, 3.3, 3.4 y 3.5 fueron creadas con Dia. Las figuras 4.1, 4.2 y 4.3 fueron creadas con Matlab.

El código fuente

El desarrollo de la biblioteca y la aplicación se realizó usando el *Eclipse SDK* 3.5 (Galileo) con el *CDT* 5.1 bajo GNU/Linux Ubuntu 9.04, 9.10 y 11.04.

El código fuente de la biblioteca, el manual del desarrollador y los ejecutables del procedimiento, se pueden obtener bajo solicitud con el Dr. Raúl Cruz Barbosa en: rcruz@mixteco.utm.mx

Índice alfabético