

UNIVERSIDAD NACIONAL DE SAN AGUSTÍN ESCUELA DE POSTGRADO

UNIDAD DE POSTGRADO DE LA FACULTAD DE INGENIERÍA
DE PRODUCCIÓN Y SERVICIOS



MODELO DE APRENDIZAJE PARA SISTEMAS DE
RECOMENDACIÓN, CASO: CURSO DE PROGRAMACIÓN WEB

Tesis presentado por el bachiller
Julio Augusto Vera Sancho
Para optar el Grado de Magister en Cs. Informatica
Con mención en Tecnologías de Información
Asesor: Alexander Victor Ocsa Mamani

Arequipa - Perú
2016

**MODELO DE APRENDIZAJE PARA SISTEMAS
DE RECOMENDACIÓN, CASO: CURSO DE
PROGRAMACIÓN WEB**

Julio Augusto Vera Sancho

SERVICIO DE POS-GRADO DE CITEC-UNSA

Data de Depósito: 16/06/2016

Assinatura: _____

MODELO DE APRENDIZAJE PARA SISTEMAS DE RECOMENDACIÓN, CASO: CURSO DE PROGRAMACIÓN WEB

Julio Augusto Vera Sancho

Orientador: *Prof. Dr. Alexander Ocsa Mamani*

Tesis presentado a la Maestría de Tecnologías e Información de la Universidad Nacional de San Agustín como parte del requisito para obtención del título de Magíster de Tecnologías de Información.

**UNSA - Arequipa
Junio/2016**

Agradecimientos

A mi orientador, Profesor. Mg. Alexander Victor Ocsa Mamani, por estar siempre presente y atento. Por todos los años de orientación, de los trabajos de investigación que concluye esta tesis de maestría. Por los desafíos que me proporcionó, y principalmente por su confianza. A mis padres Julio y Hermelinda por los años de educación que me proporcionaron, mis hermanos Andy y Cristian, por todo el apoyo que siempre me han brindado, a mis abuelos Julio y Lidia por siempre haberme proporcionado cariño y apoyo. A Concytec y Cienciactiva por todo el apoyo y darnos la confianza de realizar investigación. A Fincyt por la investigación en el proyecto 217-FINCYT-IA-2013 MLearning: Middleware para la construcción de objetos de aprendizaje móviles en ambientes reactivos y adaptables al contexto, caso educación básica regular. A todos los profesores que nos proporcionaron los conocimientos necesarios para realizar investigación, y finalmente a todas las personas que directa o indirectamente contribuyeron para que llegase hasta aquí.

Resumen

Los sistemas de recomendación en la actualidad nos ayudan a obtener resultados de búsqueda cercano o adaptados a nuestras necesidades, en los últimos años este enfoque ha ido cambiando y se ha centrado en los sistemas e-Learning y dentro de lo que son los sistemas de gestión de aprendizaje, que son tecnologías educativas muy importante para el desarrollo académico de las estudiantes, dentro de los sistemas de recomendación tradicionales se hace un *matching* entre lo que es las entradas del estudiante, que por lo general son las tareas y las notas que se le da al estudiante por su desenvolvimiento ante una tarea, que es proporcionada por un profesor o algún sistema.

Este trabajo propone un modelo de recomendación de contenidos educativos basado en el contexto de un usuario, el cual usa un modelo de contexto que incorpora el rol, las tareas, ejercicios de programación y su aplicación al problema de recomendación. Las recomendaciones se hacen sobre la base de la estimación de la diferencia que existe entre el nivel de conocimiento actual de un usuario frente a las habilidades que requiere en su contexto que se encuentra. En el trabajo se usa una técnica de razonamiento probabilístico para las recomendaciones, para tener en cuenta las especificaciones inexactas de las competencias de los usuarios y los requerimientos en su contexto.

Los experimentos desarrollados en el contexto del estudiante, muestran que, usando un modelo de razonamiento probabilístico ayuda a obtener mejores recomendaciones de contenidos educativos, según a las competencias faltantes de un estudiante respecto a un tema que necesita aprender, lo cual se busca hacer una estandarización para sistemas de recomendación.

Palabras Clave: Sistemas de recomendación, E-learning, Maquinas de aprendizaje, Redes bayesianas.

Abstract

The Recommender systems today help us to obtain results close search adapted to our needs, in recent years this approach has been changing and has focused on e-Learning systems and within what are learning management systems , which are educational technologies very important for the academic development of students within traditional systems of recommendation makes a *matching* between what is the inputs of the student, who usually are the tasks and notes given to the student for his development to a task, which is provided by a teacher or some system.

This project proposes a recommendation model for educational content based on the context of a user, which uses a context model that incorporates the role, tasks, programming exercises and their application to the problem of recommendation.The Recommendations are made on the basis of the estimate of the difference between the current level of knowledge of a user in front of the skills required in their work context.This project work a technique of probabilistic reasoning is used for recommendations to take into account the incorrect specifications of user skills and requirements in their context.

The experiments developed in the context of the student, show that, using a model of probabilistic reasoning helps to get better recommendations of educational content, according to the missing competences of a student on an issue that needs to learn, which seeks to standardization for recommendation systems.

Keywords: Recommendation system, E-learning, Machine Learning, Bayesian network.

Índice general

Resumen	III
Abstract	V
Índice	VII
Lista de Figuras	IX
Lista de Tabelas	1
1. Introducción	3
1.1. Consideraciones iniciales	3
1.2. Definición del problema	5
1.3. Objetivos	6
1.4. Principales contribuciones	6
1.5. Organización de la tesis	6
2. Consultas por similitud	7
2.1. Consideraciones iniciales	7
2.2. Dominio de datos	9
2.2.1. Espacios Métricos	9
2.2.2. Espacios Multidimensionales	10
2.3. Consultas por Similitud	11
2.4. La Maldición de la alta Dimensionalidad	14
2.5. Búsqueda aproximada de los vecinos más cercanos (ANN)	14
2.5.1. <i>Locality Sensitive Hashing</i>	15
2.5.2. LSH con Proyección Aleatoria	20
2.6. Consideraciones Finales	24
3. Minería de Datos & Deep Learning	25
3.1. Consideraciones Iniciales	25
3.2. KDD y Minería de Datos	26
3.3. Aprendizaje de Máquina – <i>Machine Learning</i>	28
3.3.1. Redes Neuronales	28
3.3.2. Redes Neuronales Convolutivas (CNN)	30
3.3.3. Tipos de capas CNN	31
3.3.4. Aplicaciones de Redes Neuronales Convolutivas (CNN)	32
3.3.5. Aprendizaje no Supervisado	34
3.3.6. Compresión de data	35
3.4. Consideraciones Finales	37

4. Teoría Fractal	39
4.1. Consideraciones Iniciales	39
4.2. Dimensión fractal	41
4.3. Algoritmo de Calculo de la Dimensión Fractal	43
4.4. Consideraciones Finales	46
5. Búsqueda aproximada vía <i>deep hashing</i>: Desempeño de métodos de representación y recuperación	47
5.1. Consideraciones Iniciales	47
5.1.1. Approximate Nearest Neighbors & Deep Hashing	48
5.1.2. Protocolos de evaluación de recuperación	50
5.1.3. Dimensión fractal y reducción de dimensionalidad	52
5.1.4. Comparación de representaciones de datos	52
5.1.5. Comparación del rendimiento de recuperación	55
5.2. Conclusions	56
6. Deep Fractal based Hashing - DAsH	59
6.1. Consideraciones Iniciales	59
6.1.1. Teoria del Fractal	61
6.2. Deep Fractal based Hashing - DAsH	61
6.2.1. Usando Fractales para estimar los parámetros LSH	63
6.3. Experimentos	64
6.3.1. Experimento 1: Sintonización de parámetros LSH	64
6.3.2. Rendimiento de Recuperación	65
6.4. Conclusions	65
7. Conclusiones y Trabajos Futuros	67
Referencias Bibliográficas	68

Índice de figuras

2.1. Unificación del modelo de consulta por similitud. (a) Los Métodos de Acceso Métricos (MAMs). (b) <i>Locality Sensitive Hashing</i> (LSH).	9
2.2. (a) Formas geométricas que ilustran la delimitación de una región de búsqueda de acuerdo con la métrica L_p utilizada. (b) Ejemplo de consulta por rango para diferentes métricas de la familia L_p	11
2.3. Ejemplo de consulta por rango.	12
2.4. Ejemplo de consulta de los k-vecinos más cercanos.	13
2.5. Ejemplo de consulta por rango aproximada.	13
2.6. Búsqueda aproximada 5-NN. (a) busca exacta. (b) busca aproximada. .	14
2.7. Interpretación geométrica de proyección de los vectores $\vec{p_1}$ y $\vec{p_2}$ en LSH. .	17
2.8. Estrutura del nodo HashFile. Figura adaptada de [Zhang et al., 2011]. .	22
2.9. Representación de las proyecciones de una consulta por similitud aproximada en HashFile en (a), en (b) se verifica su estructura lógica correspondiente.	23
3.1. Etapas del proceso KDD. Figura adaptada de [Fayyad et al., 1996].	27
3.2. Perceptrón [Aggarwal, 2015].	29
3.3. Perceptrón Multicapa [Aggarwal, 2015].	30
3.4. Puntos en un plano 2d	35
3.5. Arquitectura de auto-codificador no lineal	36
4.1. Pasos del proceso de construcción del triangulo de <i>Sierpinsky</i>	40
4.2. Algunos ejemplos de fractales (a y b) fractales geométricos y (c) fractales algebraicos	40
4.3. Una linea embebida en dos o tres dimensiones donde $D = 1$	41
4.4. El gráfico de conteo de cajas para el triangulo de <i>Sierpinki</i>	43
4.5. Representación de celdas de cuadricula en espacios de 2 y 3 dimensiones.	44
4.6. Ejemplo de la estructura de datos utilizada para calcular la Suma de Ocupaciones de un conjunto de datos con 5 puntos (con tres niveles de resolución)	45
5.1. Hashing para la búsqueda del vecino más cercano. Comprimir un conjunto de vectores $(x_i)_{i=1}^n, x_i \in \mathbb{R}^d$	49
5.2. Supervised Hashing: comprime un conjunto de vectores y sus etiquetas $((x_i, y_i))_{i=1}^n, x_i \in \mathbb{R}^d, y_i \in \{1, \dots, L\}$	50

5.3. Dimensiones Fractal para diferentes métodos de reducción de dimensio-	
nalidad	53
6.1. DAsH. Proceso de entrenamiento e indexación.	62
6.2. DAsH. Proceso de recuperación.	63

Índice de cuadros

5.1. Precisión de la clasificación usando KNN-clasificador en conjuntos de datos MNIST, CIFAR-10, SVHN y Agnews utilizando AutoEncoders y PCA como métodos de reducción de dimensionalidad.	54
5.2. Análisis de precisión de clasificación usando autoencoders para diferentes métricas.	55
5.3. Mean Average Precision(mAP) y el tiempo acumulado para calcular el mAP para diferentes métodos en los conjuntos de datos MNIST, CIFAR-10, SVHN Y AGNEWS. El número k de los vecinos devueltos se establece en 100.	56
5.4. Precisión 1000-NN y tiempo acumulado para diferentes métodos en los conjuntos de datos MNIST, CIFAR-10, SVHN y AGNEWS. El número k de los vecinos devueltos se establece en 1000.	57
6.1. Optimal LSH Params using exhaustive e2lsh and the fractal based method.	65
6.2. Mean Average Precision(mAP), precision, and cumulative time spent to compute mAP for different methods on the MNIST, SVHN and CIFAR-10 datasets.	65

Introducción

1.1. Consideraciones iniciales

La creciente disponibilidad de datos en diversos dominios, tales como multimedia, internet, la biología, entre otros, ha creado la necesidad de desarrollar técnicas y métodos capaces de descubrir conocimientos en grandes volúmenes de datos complejos, motivando varios trabajos en las áreas de base de datos, minería de datos y recuperación de la información. Esto ha impulsado el desarrollo de técnicas escalables y eficientes para organizar y recuperar estos tipos de datos complejos. Las búsquedas por similitud han sido el enfoque tradicional para la recuperación de la información. Considerando que la similitud es el criterio instintivo por el cual las personas hacen comparaciones, las comunidades de recuperación de información usan la similitud para organizar y recuperar estos datos. Sin embargo, causa algunas dificultades en representar datos complejos y también con la escalabilidad de los algoritmos cuando la dimensionalidad de los datos es muy alta (más conocido como la “maldición de la alta dimensionalidad”). Aunque muchos trabajos de investigación se han llevado a cabo en el desarrollo de estructuras y algoritmos eficientes de búsqueda sólo unos pocos presentan una garantía teórica de estabilidad asintótica para datos de alta dimensión [Kulis and Grauman, 2009, Wang et al., 2010].

Muchas soluciones responden a consultas por similitud aprovechando estructuras de índices. Los árboles son las estructuras de índice más comunes para estos dominios. Estas soluciones basadas en jerarquías pueden ser superadas mediante búsqueda secuencial en casos particulares, pues sucede que cuando los datos están en altas dimensiones la mayor parte del tiempo pasado se usa en la etapa de filtrado

[Blott and Weber, 2008]. No todas las aplicaciones requieren respuestas precisas. Por lo tanto, soluciones aproximadas son preferibles para reducir los efectos de la “maldición de la alta dimensionalidad”. Estas soluciones aproximadas se aplican en problemas a gran escala donde la velocidad es más importante que la exactitud. En este contexto, necesitamos optimizar el equilibrio entre el tiempo y la calidad de los resultados.

Uno de los pocos enfoques que aseguran una solución aproximada con el coste de búsqueda sublineal para datos de alta dimensión es *Locality Sensitive Hashing* (LSH) [Datar et al., 2004a]. LSH se basa en la idea de que la cercanía entre dos objetos suele ser preservada en una operación de proyección aleatoria. En otras palabras, si dos objetos están cerca en su espacio original, entonces estos dos objetos permanecerán cerca después de una operación de proyección escalar. Sin embargo, presenta algunas dificultades para consultas kNN aproximadas, en particular, relacionadas con la dependencia de los parámetros del dominio y la calidad de los resultados. Por lo tanto, en dominios complejos, en particular, en problemas con datos con alta dimensión, una solución aproximada con un sólido análisis teórico puede ser la mejor opción en muchas áreas de aplicación debido a su eficiencia en tiempo y espacio.

Por otro lado, también es necesario métodos que permitan obtener información y conocimiento útil a partir de los datos. La minería de datos es un proceso iterativo para el descubrimiento del conocimiento en este conjuntos de datos, realiza una búsqueda y descubrimiento de patrones dentro de estos datos, a pesar que la minería de datos abarca un amplio rango de aplicaciones, muchas de las técnicas utilizadas en la minería de datos son utilizados también en el Aprendizaje de Automático (Machine Learning, ML) que trata de extraer información o conocimiento de un conjunto de ejemplos, viendo similitudes entre los datos, y a su vez generalizando estas similitudes con los otros ejemplos. ML puede dividirse en dos tipos de aprendizaje: aprendizaje supervisado y no supervisado. El primero es aquella en la que se tiene un conjunto de datos que sirve como datos de entrenamiento para un algoritmo, y otro conjunto de datos para realizar las pruebas sobre el algoritmo ya entrenado. El segundo trabaja sobre datos en los cuales se trata de ver relaciones entre los datos para después agruparlos mediante una medida de distancia o similitud [Aggarwal, 2015].

En la línea de *Machine Learning* las imágenes se describen a menudo mediante las características visuales o vectores característica. Sin embargo, estas características manuales no pueden revelar el significado semántico de alto nivel (etiquetas o tags) de las imágenes, y a menudo limitan el rendimiento de la recuperación de imágenes [Li et al., 2015]. Así, para obtener esta información semántica tenemos que trabajar con la información de la etiqueta y procesar los datos en un modo supervisado. Inspirado por recientes avances en Red Neuronal Convolutiva (CNN) para problemas de clasificación en varios dominios de aplicación [Krizhevsky et al., 2012, Szegedy et al., 2013, Liu et al., 2012], muchos métodos resolvieron el problema de la precisión en la recuperación de información utilizando CNN como extracto-

de características para luego construir un código hash compacto de preservación de similitud para la recuperación rápida de imágenes. *Hashing* es ampliamente usado para recuperar imágenes a gran escala, así como las búsquedas de video y documentos porque la representación compacta de una código hash es esencial para el almacenamiento de datos y es eficiente para recuperar información [Shen et al.,].

1.2. Definición del problema

Los algoritmos de búsqueda aproximada basados en *hashing* son propuestos para consultar en conjuntos de datos alta dimensiones debido a su velocidad de recuperación y bajo costo de almacenamiento. Por otro lado, en problemas donde se tiene grandes volúmenes de datos etiquetados las técnicas de aprendizaje profundo, como las redes convolucionales, se muestran más adecuadas conforme el número de ejemplos por clases crece. Estudios recientes, promueven el uso de la Red Neuronal Convolutiva (CNN) con técnicas de *hashing* para mejorar la precisión de la búsqueda de los k-vecinos más cercanos - KNN. Sin embargo, aun hay retos que resolver para encontrar una solución práctica y eficiente para indexar características CNN:

- La necesidad de un proceso de entrenamiento intenso para lograr resultados precisos.
- La dependencia crítica de los parámetros, pues los algoritmos dependen de estos parámetros para una proyección en un subespacio sin perdida de información.
- Uso adecuado de la información semántica de las últimas capas de una red CNN.
- Existe una relación entre el error de clasificación y el error de cuantificación en una red CNN: las activaciones de capas inferiores son más generales [Yosinski et al., 2014], a la vez que el entrenamiento es más eficaz. Sin embargo, estas capas inferiores tienen mapas de activaciones más grandes (cientos/miles de nodos), las cuales son más difíciles de codificar, lo que conduce a un compromiso.

Por otro lado, en problemas de gran escala con grandes volúmenes de datos las técnicas de aprendizaje profundo, se muestran más adecuadas conforme el número de ejemplos por clases crece. Además, para tareas de procesamiento y clasificación de los datos ahora se puede usar el alto rendimiento de las GPUs. En problemas de gran escala, arquitecturas secuenciales (CPUs) imponen limitaciones de desempeño. Por lo tanto, recurrir a las soluciones basadas en GPU es una manera de superar tales limitaciones de desempeño.

1.3. Objetivos

- Estudiar y desarrollar métodos óptimos de proyección en subespacios usando la teoría fractal. El objetivo es mostrar que un buen algoritmo de reducción de dimensionalidad proyecta los datos en un espacio de características con dimensionalidad cercana a la dimensionalidad fractal (FD).
- Estudiar y desarrollar un nuevo método de *hashing* supervisado que sea independiente de los parámetros dominio y diseñado para realizar una búsqueda aproximada escalable. La idea es usar análisis fractal para la optimización de parámetros de dominio para poder integrar los métodos de aprendizaje profundo, vía redes convolucionales como extracto de características y las técnicas de búsqueda aproximada como esquema de recuperación de información.

1.4. Principales contribuciones

Las contribuciones de nuestro trabajo son las siguientes:

- Estudiar y desarrollar métodos óptimos de proyección en subespacios. Desarrollaremos un conjunto de experimentos que muestren el uso de teoría fractal como herramienta para encontrar el subespacio óptimo de indexación. Estos estudios serán descritos en el la sección 5.2 del capítulo 5.
- Proponemos un nuevo método, llamado *Deep Fractal based Hashing* (DAsH), diseñado para realizar una búsqueda aproximada escalable mediante un esquema de *hash* supervisado. Este método es descrito en la sección 6.2 del capítulo 6.

1.5. Organización de la tesis

El presente trabajo está organizado de la siguiente manera:

1. Introducción
2. Fundamentos Teóricos
 - a) Consultas por similitud
 - b) Minería de Datos & Deep Learning
 - c) Teoría Fractal
3. Búsqueda aproximada vía *deep hashing*
4. *Deep Fractal based Hashing - DAsH*
5. Conclusiones y Trabajos Futuros

Consultas por similitud

2.1. Consideraciones iniciales

En el área de base de datos, las técnicas de indexación tienen como objetivo auxiliar el almacenamiento y la recuperación eficiente de datos. Muchas de estas técnicas se aplican a tareas de recuperación de información. En general, la organización se lleva a cabo a través de una estructura de indexación, también denominado como Método de Indexación (MI) o Método de acceso (MA). Aplicaciones de base de datos emplean métodos de acceso como Métodos de Acceso Espacial (MAEs) y los Métodos de Acceso Métrico (MAM) debido a su capacidad para construir una estructura de datos para gestionar y organizar grandes cantidades de datos de manera eficiente.

Sin embargo, debido a la "maldición de la alta dimensionalidad", el tiempo necesario para llevar a cabo la búsqueda por similitud, como kNN, puede crecer exponencialmente con la dimensionalidad de los datos. Aunque no se conoce ningún método para un buen desempeño en todas las instancias del problema, hay muchos algoritmos para acelerar las consultas, por lo general a través de una aproximación a la respuesta correcta. Así, se definen dos tipos de búsqueda: Métodos de búsqueda exactos y métodos de búsqueda aproximada; sobre los métodos de búsqueda de línea Búsqueda exacta de los métodos de acceso espacial (SAMs) y los métodos de acceso Métrico (MAM) son los más conocidos y utilizados en muchas aplicaciones.

La línea de investigación de los métodos de búsqueda exacta, los Métodos de Acceso Espaciales (MAEs), como Kd-Tree [Bentley, 1979], R-Tree [Guttman, 1984], R*-Tree [Beckmann et al., 1990], R+-Tree [Sellis et al., 1987] y , X-Tree [Berchtold et al., 1996],

describen los datos de entrada como vectores multidimensionales. Estos métodos fueron concebidos para apoyar las operaciones de búsqueda involucrando puntos y objetos geométricos.

Aunque en esta línea, Faloutsos [Faloutsos et al., 1994a] propone el *framework* GEMINI (*GEneric Multimedia INdexIng*), que utiliza los MAEs junto con los métodos de reducción de dimensionalidad para lidiar con los datos en alta dimensionalidad. La idea general de este *framework* consiste en reducir el costo de búsqueda con la utilización de un esquema para identificada de falsas alarmas, para descartar la mayoría de los objetos que no califican como candidatos.

En la misma dirección de los métodos de búsqueda exacta, los Métodos de Acceso Métricos (MAMs) proporcionar operaciones de búsqueda por similitud, exacta, en espacios métricos. Las estructuras de indexación de los MAMs organizan los datos usando un criterio de similitud, asegurando una respuesta eficiente en las consultas. En la literatura son encontrados MAMs basados en árboles, como VP-Tree [Yianilos, 1993], SAT [Navarro, 2002], M-Tree [Ciaccia et al., 1997], Slim-Tree [Traina C. et al., 2002], DBM-Tree [Vieira et al., 2004], DF-Tree [Traina et al., 2002] e PM-Tree [Skopal et al., 2005], y las técnicas basadas en grafos, como t-Spanners [Navarro et al., 2007] e HRG [Ocsa et al., 2007]. Estados extensos sobre MAMs poden ser encontrados en [Chávez et al., 2001, Hjaltason and Samet, 2003, Clarkson, 2006].

Los MAMs utilizan intensamente la propiedad de la desigualdad triangular para reducir el número de cálculos de distancia. Sin embargo, aunque muchos MAMs han sido propuestos para acelerar la búsqueda por similitud, algunas aún sufren con el problema de sobre posición. Por otra parte, algunos trabajos de investigación discuten la idea de que indexación jerárquica de datos en altas dimensiones pode deteriorar las consultas, al igual en comparación con la búsqueda secuencial [Böhm et al., 2001, Blott and Weber, 2008]. Los espacios métricos son definidos a partir de un conjunto de objetos y una función de distancia métrica que mide la disimilitud entre estos objetos, satisfaciendo las siguientes propiedades: positividad, simetría, reflexividad y desigualdades triangular.

En otro enfoque, una técnica promisoria llamada *Locality Sensitive Hashing* (LSH) [Datar et al., 2004b] fui propuesta para realizar consulta por similitud aproximada en datos en altas dimensiones, de manera eficiente. LSH, se basa en la idea de que la proximidad entre dos objetos se preserva generalmente por una operación de proyección aleatoria. En otras palabras, si dos objetos están cerca en el espacio original, por lo que permanecen cerca después de la operación de proyección.

Un modelo de LSH es ilustrado en la Figura 2.1 (b), donde cada uno de los tres sub-índices es definido por un conjunto de funciones *hash* (H_1, H_2, H_3), generando tres tipos de particionamiento en el espacio de búsqueda. Así, cada partición está asociada a una tabla *hash* y su correspondiente conjunto de funciones hash. Cada conjunto de funciones *hash* es utilizado para organizar un conjunto de datos en regiones, de modo que con cierta probabilidad, los objetos en la misma región son

considerado lo suficientemente próximos. En tiempo de consulta, el objeto de consulta q es proyectado (usando cada uno de los tres conjuntos de funciones *hash*, uno por cada partición) en regiones donde la probabilidad de encontrar objetos próximos es muy alta (regiones en color gris en la figura). Finalmente, ya que son considerados varios sub-índices, las regiones candidatas son analizadas a fin de retornar solo los objetos que satisfacen la condición de consulta e que no hayan sido retornados antes.

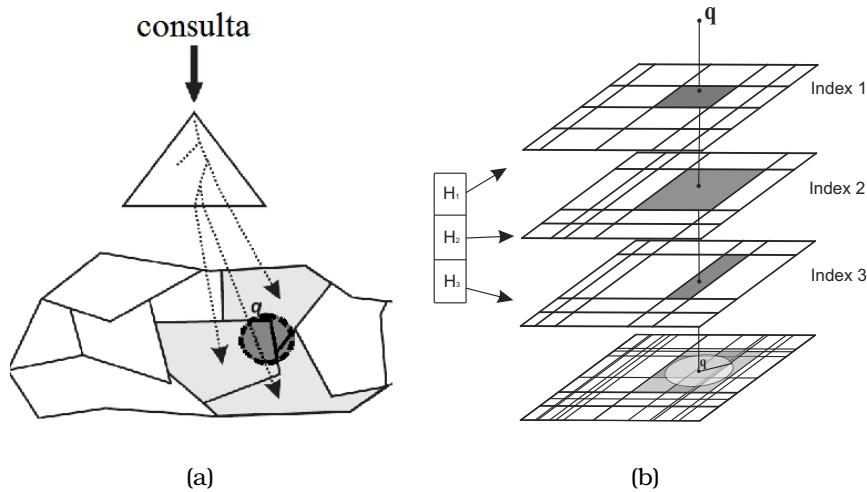


Figura 2.1: Unificación del modelo de consulta por similitud. (a) Los Métodos de Acceso Métricos (MAMs). (b) *Locality Sensitive Hashing* (LSH).

2.2. Dominio de datos

la concepción de un método de búsqueda debe considerar la naturaleza de los datos, a fin de explotar ciertas propiedades del espacio en donde están embebidos.

La gran mayoría de métodos de búsqueda supone que los datos pertenecen a un espacio n -dimensional. Esos espacios están en R^n , lo que permite el uso de propiedades geométricas en la solución. Por otro lado, algunos métodos tienen exigencias menos rigurosas, como el caso de los métodos basado en espacios métricos, que solo usan las distancias entre los puntos, y ninguna otra propiedad geométrica.

2.2.1. Espacios Métricos

Un espacio métrico es definido como $\mathbb{M} = \langle S, d \rangle$, donde S es un universo de elementos y d es una función de distancia (o métrica), definida sobre los elementos en S , que mide la disimilitud los objetos y satisfacen las siguientes condiciones, $\forall x, y, z \in S$:

- | | |
|---|------------------------|
| (1) $d(x, y) \geq 0$ | positividad |
| (2) $d(x, y) = 0 \leftrightarrow x = y$ | reflexibilidad |
| (3) $d(x, y) = d(y, x)$ | simetría |
| (4) $d(x, y) \leq d(x, z) + d(y, z)$ | desigualdad triangular |

La desigualdad triangular es considerado una de las propiedades mas importantes, pues es utilizada para establecer los límites del valor de distancia entre dos objetos sin la necesidad del cálculo real de distancia, acelerando así los algoritmos de consulta por similitud. Mas específicamente, dados los valores de distancia $d(x, z)$ y $d(y, z)$, los límites para el valor (desconocido) de $d(x, y)$ son $|d(x, z) - d(y, z)| \leq d(x, y) \leq d(x, z) + d(y, z)$.

No todas las propiedades (1)-(4) son necesarias para todos los métodos. Por ejemplo, se puede substituir (2) por una propiedad mas débil $\forall x \in S, d(x, x) = 0$, tornando el espacio pseudo-métrico. Los métodos que no obedecen las propiedades (3), (4) o ambas son típicamente llamados no métricos.

2.2.2. Espacios Multidimensionales

Si los objetos del dominio S corresponden a los vectores de valores numéricos entonces el espacio es llamado Espacio Multidimensional o Espacio Vectorial con Dimensión Finita. Los objetos de un espacio multidimensional de dimensión n (o n -dimensional) son representados por n coordenadas de valores reales x_1, \dots, x_n .

Las funciones de distancia métrica mas común para medir la similitud entre elementos en un Espacio Multidimensional son de la familia L_p , o Minkowski, definidas por:

$$L_p((x_1, \dots, x_n), (y_1, \dots, y_n)) = \left(\sum_{i=1}^n |x_i - y_i|^p \right)^{1/p} \quad (2.1)$$

- La distancia L_1 es también conocida como distancia **Manhattan**, y corresponde a una simple suma de las diferencias absolutas de los componentes.
- La distancia L_2 es también conocida como distancia **Euclíadiana**, y corresponde a la idea usual de distancia en espacios 2D o 3D.
- La distancia L_∞ es también conocida como **Chebychev**, y corresponde a la diferencia máxima absoluta entre los componentes, definida por:

$$L_\infty((x_1, \dots, x_n), (y_1, \dots, y_n)) = \max_{i=1}^n |x_i - y_i| \quad (2.2)$$

La Figura 2.2(a) ilustra, las distancias de la familia L_p , el conjunto de puntos que están a la misma distancia r , a partir de un centro q , en un espacio bi-dimensional.

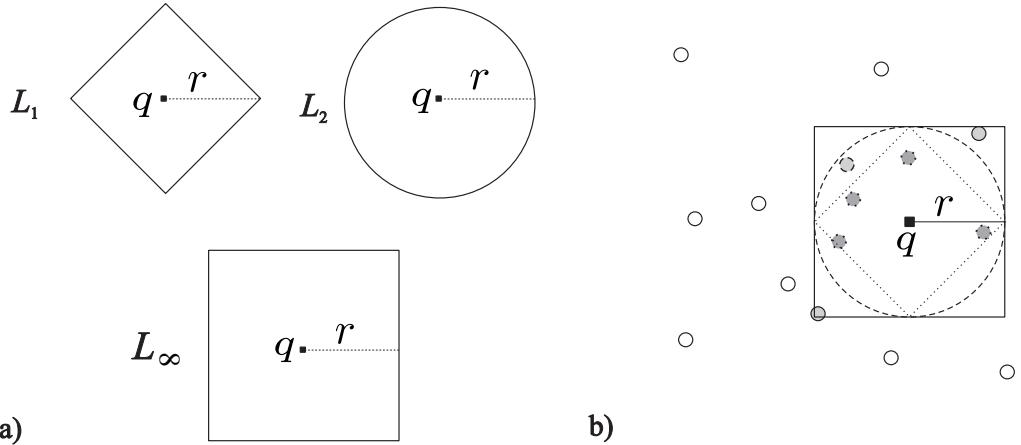


Figura 2.2: (a) Formas geométricas que ilustran la delimitación de una región de búsqueda de acuerdo con la métrica L_p utilizada. (b) Ejemplo de consulta por rango para diferentes métricas de la familia L_p .

La Figura 2.2(b) representa un conjunto de objetos en un espacio bi-dimensional, un objeto de consulta q , un radio de consulta r , los conjuntos de respuesta y tres métricas utilizadas: L_1 , L_2 e L_∞ .

La **distancia de coseno** presentada en la Ecuación 2.3 es otra métrica de distancia popular para la búsqueda de NN que ha demostrado ser particularmente eficaz para la recuperación de documentos [Manning and Schutze, 2008, Ravichandran and Hovy, 2005].

$$d_{cosine}(X_i, X_j) = 1 - \frac{\sum_{k=1}^D x_{ik}x_{jk}}{\sqrt{\sum_{k=1}^D x_{ik}^2} \sqrt{\sum_{k=1}^D x_{jk}^2}} \quad (2.3)$$

También encontraremos la **distancia de Hamming** ampliamente en esta tesis, ya que es la métrica por defecto para comparar cadenas binarias (Ecuación 2.4)

$$d_{hamming}(b_i, b_j) = \sum_{k=1}^D \delta[b_{ik} \neq b_{jk}] \quad (2.4)$$

La función $\delta(.) = 1$ si su argumento es verdadero, y 0 en caso contrario. Por lo tanto, la distancia de Hamming cuenta el número de dimensiones correspondientes (bits) que no son iguales en los dos *hashcodes*.

2.3. Consultas por Similitud

Aplicaciones en la recuperación de información por similitud, por lo general crea un universo de objetos \mathbb{U} y una función $d : \mathbb{U} \times \mathbb{U} \rightarrow \mathbf{R}$ que mide la distancia entre dos objetos en \mathbb{U} . En los espacios métricos, definimos $\mathbb{S} \subseteq \mathbb{U}$ como un conjunto finito de

objetos, donde la función $d()$ mide la disimilitud entre objetos.

Dado un objeto de consulta $q \in \mathbb{U}$, los objetos similares a q se pueden recuperar de acuerdo a los siguientes tipos de consultas por similitud:

Consulta por radio (*range query* $Rq(q, r)$) consulta que tiene como objetivo recuperar los objetos similares a q que están dentro del rango de la consulta r .

$$Rq(q, r) = \{u \in S | d(u, q) \leq r\} \quad (2.5)$$

En la Figura 2.3 se representa una consulta por rango en un espacio bi-dimensional con la métrica L_2 . Los elementos contenidos por el radio r componen la respuesta. Vale recordad que no es necesario que el elemento de consulta pertenezca al conjunto de datos de búsqueda, debiendo este pertenecer al mismo dominio de datos.

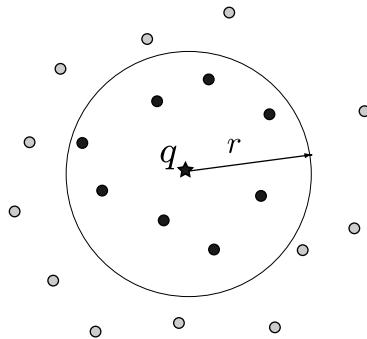


Figura 2.3: Ejemplo de consulta por rango.

Consulta de los k -vecinos más cercanos (*k -Nearest neighbor query - $kNN(q, k)$*) consulta que tiene como objetivo recuperar los k objetos más cercanos al objeto de búsqueda q . O más formalmente, los k vecinos más próximos definen el conjunto $C = \{s_1, s_2, \dots, s_k\}$ en que:

$$\forall s_i \in C, \forall x_j \in S - C, d(q, s_i) \leq d(q, x_j) \quad (2.6)$$

La Figura 2.4 ejemplifica una consulta kNN en un espacio bi-dimensional con la métrica L_2 . En la figura, la consulta tiene como entrada el elemento de consulta q y el valor de k igual a 3. Los elementos conectados a q corresponden al consulto de respuesta, considerando que en el ejemplo el elemento q no pertenece al conjunto de datos.

Consulta por rango aproximada - $(1 + \varepsilon)Rq(q, r)$ Consulta que busca recuperar los objetos cercanos a q que se encuentran dentro del radio de consulta $(1 + \varepsilon) \times r$. O, más formalmente:

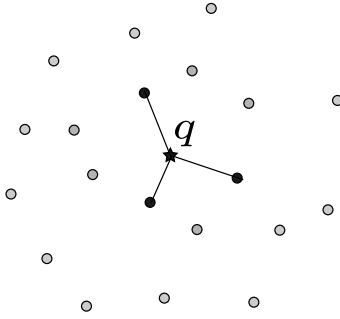


Figura 2.4: Ejemplo de consulta de los k -vecinos más cercanos.

$$(1 + \varepsilon)Rq(q, r) = \{u \in S | d(u, q) \leq (1 + \varepsilon) \times r\} \quad (2.7)$$

La Figura 2.5 ejemplifica una consulta de este tipo para L_2 . Los elementos dentro del radio de consulta $(1 + \varepsilon) \times r$ componen la respuesta aproximada.

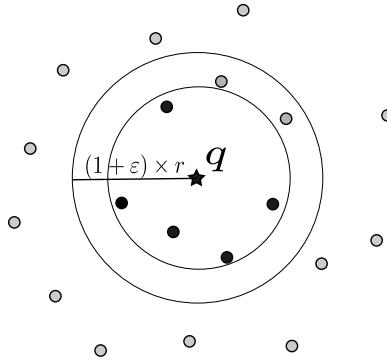


Figura 2.5: Ejemplo de consulta por rango aproximada.

Consulta Aproximada a los k -vecinos más cercanos - $(1 + \varepsilon) - kNN(q, k)$ Sea r la distancia entre el objeto de consulta q y el elemento más distante entre los k verdaderos vecinos más próximos, esto es,

$$r = \max\{s_j \in S | d(q, s_j)\} \quad (2.8)$$

La búsqueda $(1 + \varepsilon) - kNN$ para los k elementos mas similares a q consiste en encontrar un conjunto $C' = \{s'_1, s'_2, \dots, s'_k\}$ en donde,

$$\forall s'_i \in C', d(q, s'_i) \leq (1 + \varepsilon) \times r \quad (2.9)$$

El factor $(1 + \varepsilon)$ es usualmente llamado **factor de aproximación**, y indica que el conjunto de la solución C' está dentro de un **error relativo** ε a la respuesta exacta. Ver Figura 2.6.

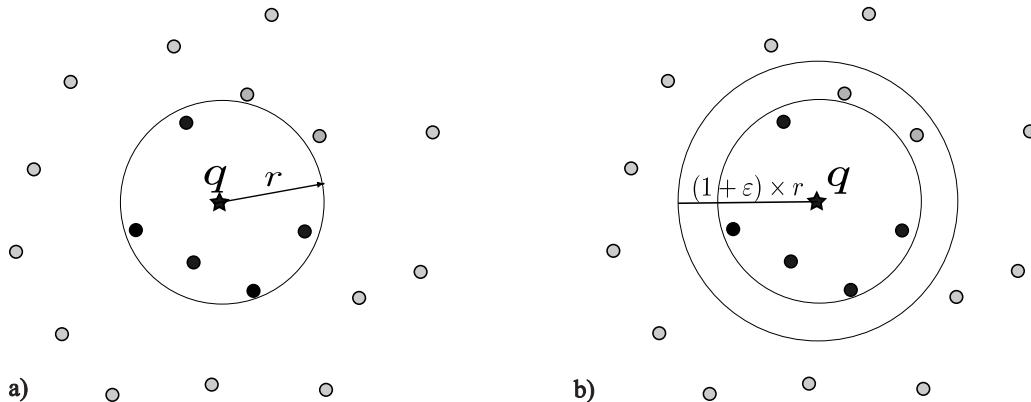


Figura 2.6: Búsqueda aproximada 5-NN. (a) busca exacta. (b) busca aproximada.

2.4. La Maldición de la alta Dimensionalidad

La eficiencia de los métodos de búsqueda por similitud dependen mucho de la dimensionalidad. Aunque el tiempo de búsqueda pueda alcanzar un costo logarítmico en relación al tamaño de los datos, este va crecer exponencialmente con la dimensionalidad de los datos.

Para dimensiones moderadas, la mayoría de los métodos ejecuta las consultas de manera suficientemente eficiente para permitir una solución exacta en un tiempo razonable. Para dimensiones más altas, se vuelve viable el uso de métodos aproximados, lo que implica en un *trade-off* entre la precisión y la eficiencia. Para dimensiones más altas ese *trade-off* va convertirse progresivamente más relevante, resultando en penalidades mayores en términos de precisión, a fin de obtener una eficiencia aceptable.

El problema de “la maldición de la alta dimensionalidad” fue estudiado originalmente por Bellman [Bellman, 1961], observando que particiones del espacio de soluciones en problemas de optimización son ineficientes en problema con datos en altas dimensiones. Los efectos de “maldición” en Métodos de Acceso Espaciales (MAEs) y Métodos de Acceso Métricos (MAMs) son discutidos en [Böhm et al., 2001, Blott and Weber, 2008, Volnyansky and Pestov, 2009].

2.5. Búsqueda aproximada de los vecinos más cercanos (ANN)

En esta sección definimos formalmente el problema de la búsqueda del vecino más cercano (Nearest Neighbor search - NN). Se examinará entonces la versión simplificada de la búsqueda NN conocida como búsqueda NN aproximada, el campo en el que se basa esta tesis, y describirá cómo difiere de los algoritmos alternativos para resolver el problema de búsqueda NN.

Los trabajos descritos en este capítulo todos comparten la definición del mismo problema. Dado un conjunto de datos X que consiste de N puntos ($X \in R^{NxD} =$

$[x_1, x_2, \dots, x_N]^T$), donde cada punto $x_i \in R^D$ es un vector D-dimensional de valores reales (vector característica). El objetivo es construir un conjunto de m funciones hash $\{h_m : R^D \rightarrow \{0, 1\}\}_{m=1}^K$ cuya salida puede ser concatenada como $[h_1(x_i), h_2(x_i), \dots, h_m(x_i)]$ el cual cumple la función de codificar el vector característica. Para codificar se necesitan funciones *hash* que preserven la similitud del espacio original.

La búsqueda de vecinos más cercanos se puede definir como el problema de recuperar el punto de datos más cercano $NN(q)$ para una consulta $q \in R^D$ en un conjunto de datos de N puntos $[x_1, x_2, \dots, x_N]^T$ donde $x_i \in R^D$. La similitud entre los puntos se define por una función de distancia $\{d(\cdot, \cdot) : R^D \times R^D\}$. Es fácil generalizar esta definición de problema para devolver los vecinos K más cercanos a la consulta. Esta variante se conoce como búsqueda k-NN y es un componente fundamental en una amplia gama de diferentes métodos de aprendizaje de máquinas. La función de distancia $d(x, y)$ entre los puntos de datos se suele calcular utilizando una métrica de distancia genérica como la $l_p - norm$ (Ecuación 2.1).

Para buscar NNs a una consulta necesitamos construir una estructura de datos o algoritmo que tome nuestra noción seleccionada de distancia y recupera los puntos de datos que están cerca de la consulta bajo esa métrica de distancia específica. La búsqueda por fuerza bruta es un algoritmo sencillo para resolver el problema de búsqueda del vecino más cercano con cualquier métrica de distancia deseada. En la búsqueda por fuerza bruta se calcula la distancia a todos los puntos en la base de datos y los puntos con la menor distancia a la consulta son devueltos como el vecino más cercano. Las ventajas de la búsqueda de la fuerza bruta son su simplicidad de la puesta en práctica y su garantía que los vecinos más cercanos serán recuperados eventualmente. Sin embargo, comparar exhaustivamente la consulta con cada punto en la base de datos da una complejidad temporal $O(ND)$ que hace rápidamente la búsqueda de fuerza bruta intratable para la búsqueda de vecinos más cercanos a través de conjuntos de datos con muchos puntos de datos (N) y una dimensionalidad moderada a alta (D). En esta situación, se requiere un enfoque más informado del problema de búsqueda del vecino más cercano.

En la sección 2.5.1, introduciremos el método *Locality Sensitive Hashing (LSH)*, una familiar de algoritmos que proveen un método para resolver búsqueda aproximada de los vecinos mas próximos con un tiempo de consulta constante.

2.5.1. Locality Sensitive Hashing

Algunos trabajos de búsqueda aproximada [Gionis et al., 1999, Datar et al., 2004b, Andoni and Indyk, 2008] exploran la idea de mapear los objetos de un conjunto de datos y agruparlos en *buckets* con el objetivo de ejecutar consultas por similitud aproximada dentro de los *buckets* asociados al objeto de consulta. En particular, el método *Locality Sensitive Hashing (LSH)* fue ideado para resolver eficientemente

consultas por rango aproximada ($(1 + \epsilon)Rq(q, r)$). Como fue definido en la Sección 2.3, este tipo de consulta busca recuperar los objetos que se encuentran dentro de un radio de consulta $(1 + \epsilon) \times r$. La idea principal es que, si dos objetos son próximos en el espacio original, esos dos objetos tienden a permanecer próximos después de una operación de proyección escalar randomica. Así, dada una función *hash* $h(x)$ que mapea un objeto (x) de dimensión n a un valor unidimensional, la función es sensible a la localidad si la posibilidad de mapeamiento de dos objetos x_1, x_2 al mismo valor crece a la medida que la distancia $d(x_1, x_2)$ disminuye. Formalmente:

Definición Dado un valor de distancia r , un factor de aproximación $1 + \epsilon$, las probabilidades P_1 y P_2 , tal que $P_1 > P_2$, la función *hash* $h()$ es sensible a la localidad de los datos si satisface las siguientes condiciones:

- *Se* $d(x_1, x_2) \leq r \Rightarrow \Pr[h(x_1) = h(x_2)] \geq P_1$.

Esto quiere decir que para dos objetos x_1 y x_2 en R^n que están lo suficientemente próximos, hay una probabilidad P_1 que ellos caigan en el mismo *bucket*.

- *Se* $d(x_1, x_2) > (1 + \epsilon) \times r \Rightarrow \Pr[h(x_1) = h(x_2)] \leq P_2$.

Esto quiere decir que para dos objetos x_1 y x_2 en R^n que están distantes, hay una probabilidad $P_2 < P_1$ que ellos caigan en el mismo *bucket*.

El esquema propuesto en [Datar et al., 2004b] fue proyectado para trabajar con distribuciones p-estables de la siguiente manera: calcular el producto escalar $\vec{a} \cdot \vec{x}$ para atribuir un valor *hash* para cada vector x . La función *hash* precisa de valores randomicos \vec{a} y b , en que \vec{a} es un vector n-dimensional con valores independientemente escogidos a partir de distribuciones p-estables (Cauchy o Gaussiana) y b es un número real escogido dentro de un intervalo $[0, \omega]$. Por tanto la función *hash* $h(x)$ esta dada por:

$$h(x) = \lfloor \frac{\vec{a} \cdot \vec{x} + b}{\omega} \rfloor \quad (2.10)$$

La ecuación 2.10 tiene una interpretación simple. Considere la Figura 2.7: \vec{p}_1 y \vec{p}_2 son dos vectores en R^2 , \vec{a} es un vector normal unitario e b es un número randomico real. A inclinación de la recta pasando por el origen coincide con la dirección de \vec{a} . Así, \vec{p}_1 es proyectado sobre la línea \vec{a} calculando el producto escalar $\vec{a} \cdot \vec{p}_1$. Esa proyección es cuantizada en intervalos de tamaño fijo ω , definiendo el punto A. El mismo procedimiento es repetido para \vec{p}_2 , definido el punto B.

Para amplia a diferencia entra las probabilidades P_1 y P_2 se puede usar m funciones *hash* diferentes. Esto aumenta las probabilidades dado que $(P_1/P_2)^m > P_1/P_2$, asegurando así que si dos objetos están muy distantes, la probabilidad de que caigan en el mismo *bucket* sea baja. Sin embargo, aunque sea importante evitar que objetos distantes caigan en el mismo *bucket* para preservar la proximidad espacial, esto no

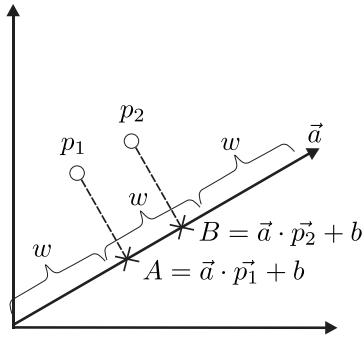


Figura 2.7: Interpretación geométrica de proyección de los vectores \vec{p}_1 y \vec{p}_2 en LSH.

es una condición suficiente. Es igualmente importante asegurar que objetos próximos en el espacio original tengan una alta probabilidad que aparecer en el mismo *bucket*. Así, como eso no puede ser alcanzado usando solo una estructura *hash*, LSH resuelve ese problema considerando L proyecciones independientes, ósea, la construcción de L tablas *hash* (L subíndices) [Slaney and Casey, 2008, Tao et al., 2010].

Así usando colecciones diferentes de m funciones *hash* $H = \{h_1, h_2, \dots, h_m\}$, siendo una colección cada sub-índice, cada objeto x de un conjunto de datos es mapeado en *buckets*, efectivamente particionando el conjunto de datos en varios grupos pequeños (los *buckets*) para cada uno de los L subíndices. La estructura de datos será entonces formada por L subíndices. LSH divide el espacio de búsqueda empleando m funciones *hash* escogidas aleatoriamente de una distribución Gaussiana (o Cauchy). Un número de funciones *hash* m determina cuán esparzo o denso será el espacio de búsqueda. Al aumentar el valor de m , los objetos tienden a ser distribuidos en *buckets* de manera bastante uniforme, reduciendo la precisión de las consultas ya que es más probable que objetos similares caigan en *buckets* diferentes. Para atenuar ese efecto negativo, son necesarias muchas tablas *hash*. Por otro lado, al disminuir el valor de m , el número de colisiones aumenta, así como el número de elemento a ser tratados en tiempo de consulta. Así, el desempeño de las consultas disminuye.

El proceso de indexación de un conjunto de datos S usando LSH es mostrado en el **Algoritmo 2.1**. Cada elemento x del conjunto de elementos S es proyectado usando m funciones *hash* que transforma el elemento x en m números reales (x' , el vector *hash*). Esos m números son entonces cuantificados con un único número (g).

Algoritmo 2.1: Algoritmo de construcción para LSH

```

1 Entrada: El conjunto de datos  $S$ 
2 Salida: Todos los objetos mapeados en las  $L$  tablas hash
3   for  $i = 1$  to  $L$  do
4      $H_i \leftarrow \{h_1, \dots, h_m\}$  // Inicializar tabla  $T_i$  con un conjunto de funciones hash  $H_i$ 
5   end for
6   for  $i = 1$  to  $L$  do
7     foreach  $x$  en  $S$  do
8        $x' \leftarrow \langle h_1(x), \dots, h_m(x) \rangle$  // Proyectar  $x$ :  $m$  veces usando el conjunto de funciones hash  $H_i$ 
```

```

9       $g \Leftarrow \text{quantize}(x') // Calcular el valor hash final$ 
10      $I \Leftarrow T_i[ g \bmod |T_i| ] //localizar el bucket I$ 
11     Insertar la entrada  $\langle x, g \rangle$  en  $I$ , almacenando la referencia de  $x$  y el valor  $g$ 
12   end for
13 end for

```

Una vez que cada *bucket* es definido en términos de similitud (esto es, elementos semejantes tienden a ser encontrados en el mismo *bucket*, es probable encontrar los vecinos más próximos a un objeto q en los *buckets* recuperados por cada conjunto de funciones *hash*). Así, en vez de hacer comparaciones para reducir el espacio de búsqueda, LSH mapea directamente los objetos a los *buckets*. Luego, es posible obtener un costo sub-linear en las consultas. En una consulta kNN, a fin de retornar los k vecinos más próximos, apenas las distancias entre q y los elementos dentro de los *buckets* son calculados. En una consulta por rango sucede lo mismo, mas la condición de consulta es diferente y solo los elementos dentro del radio de consulta son retornados. El algoritmo de consulta por rango aproximada es descrito en el Algoritmo 2.2.

Algoritmo 2.2: Consulta por rango aproximada usando LSH

<ol style="list-style-type: none"> 1 Entrada: El objeto de consulta q y el radio de consulta r 2 Salida: Los objetos que satisfacen las condiciones de consulta. <pre> 3 for $i = 1$ to L do 4 <math>q' \Leftarrow \langle h_1(q), \dots, h_m(q) \rangle //Proyectar x: m veces usando el conjunto de funciones hash H_i</math> 5 $g \Leftarrow \text{quantize}(q') // calcular el valor hash final$ 6 $I \Leftarrow T_i[g \bmod T_i] //localizar el bucket I$ 7 foreach e en I do 8 if $e.g = g \wedge d(q, e.x) \leq r$ then 9 retornar $e.x$ si no fue reportado 10 end if 11 end for 12 end for </pre>

El algoritmo kNN puede ser obtenido por medio de simples modificaciones en el Algoritmo 2.2. Básicamente, el algoritmo kNN terminará cuando se encuentre k objetos distintos o si no hubiere más *buckets* para explorar. No obstante, es importante recordar que LSH garantiza resultados de calidad previsible apenas para consultas por rango aproximada ($(1 + \epsilon) - Rq(q, r)$). Además, algunos inconvenientes de LSH no fueron resueltos por completo, por ejemplo: (1) LSH requiere varios subíndices de moco que cada subíndice organiza el conjunto de datos entero usando una tabla *hash* con funciones *hash* independientes. Esta exigencia es extremadamente crítica para mejorar la precisión de búsqueda, mas lleva a un alto consumo de memoria; (2) LSH tiene una dependencia crítica de los parámetros de dominio, que determina el número de funciones *hash* y el número de tablas *hash*.

En *hash* los autores propusieron LSH Multi-probe, que busca mantener un costo de memoria aceptable. LSH Multi-probe está basado en el LSH clásico, mas en cuanto

el algoritmo de consulta LSH examina apenas un *bucket* para cada tabla *hash*, LSH Multi-probe examina los *buckets* que son susceptibles de contener los resultados de consulta por cada tabla *hash*. Consecuentemente, el número de tablas *hash* es reducido sin perdida significativa de precisión. Intuitivamente esta propuesta verifica los *buckets* de forma más inteligente, generando τ *probes* por cada tabla *hash*, lo que permite explorar τ *buckets* por sub-índice, y como consecuencia, aumentar el número de candidatos sin incrementar el número de subíndices.

Una investigación más reciente sobre el método LSH [Dong et al., 2008] muestra que el desempeño en las consultas no solo dependen de la distribución general del conjunto de datos, mas también de la distribución local en torno a un objeto de consulta. En el método LSH Multi-probe, un número fijo de *probes* puede ser insuficiente para algunas consultas y mayor del que es necesario para otras. No obstante aun es complicado ajustar el número de funciones *hash* asociado al radio de consulta.

Dado que los parámetros también dependen del número de objetos a ser indexados, el método LSH no es una solución incremental. Para lidiar con este problema, una nueva propuesta fue presentada, el método LSH-Forest [Bawa et al., 2005]. Esa técnica fue desarrollada para soportar auto-ajuste en relación al número de funciones *hash*. Esencialmente, el método LSH-Forest es una colección de arboles de prefijos donde cada una puede tener un número diferente de funciones *hash*. No obstante, aun es necesario ajustar el número de subíndices, o sea, el número de arboles prefijo. Además, no está claro su desempeño en sistemas que trabajan con datos distorsionados, en donde se tiende a verificar más candidatos del que es necesario en pequeñas áreas densas y tener candidatos insuficientes para los puntos de consulta en áreas esparzas.

En ese mismo escenario, la técnica LSH Multi-level [Ocsa and Sousa, 2010], desarrollada en el contexto de este trabajo, propone un nuevo esquema de *hashing* para resolver algunos de esos problemas. Específicamente, utilizan un esquema multiresolución para resolver la dependencia de parámetros de dominio. El método no espera los parámetros del dominio de datos, pues, se adapta dinámicamente durante el proceso de indexación, gracias a las habilidades auto-adaptativas de la estructura del índice multi-nivel. Este método presenta un mejor desempeño en término de tiempo y espacio comparado con otras propuestas *hash*, pues la técnica multi-nivel distribuir los objeto en *buckets* de manera uniforme en todos los niveles. Esto porque, en contraste con LSH, el método LSH Multi-level usa la estructura multi-resolución para calcular y localizar los vectores *hash* apropiados para una consulta específica. Así, varios vectores *hash* de diferentes resoluciones son calculados por cada índice en el proceso de consulta y, como consecuencia, no se precisa de mas índices para garantizar resultados de la misma calidad.

Recientemente, en [Tao et al., 2010] los autores propusieron el método LSB-Forest. Este trabajo mejor la técnica LSH-Forest, y a fin de garantizar la calidad

y eficiencia de recuperación de datos multidimensionales, utiliza representaciones basadas en *space-filling curves* y arboles B. De ese modo, los valores *hash* son representados como valores unidimensionales usando tanto la proyección LSH como las curvas Z. Además, varios arboles B son usados para indexar los datos con el objetivo de mejorar la calidad de los resultados. Una desventaja de los arboles LSB es que usa lecturas/escrituras randomicas, lo que conlleva requiere un número considerable de accesos a disco cuando el conjunto de datos es grande. Para resolver ese problema, el método HashFile [Zhang et al., 2011] fue propuesto. Como será detallado en la siguiente sección, en comparación a los métodos actuales LSH, el método HashFile solo divide recursivamente los *buckets* densos para alcanzar particiones mas equilibradas. Cada *bucket* almacena un número fijo de objetos, mas el método aprovecha la lectura secuencial en vez de usar el acceso randomico, como es el caso en los arboles B. Este método esta basado en conceptos de proyección aleatoria (*Random Projection*).

2.5.2. LSH con Proyección Aleatoria

El algoritmo de proyección aleatoria (Random Projection) básicamente usa vectores aleatorios provenientes de una distribución estable. El algoritmo es robusto contra el ruido y utiliza un enfoque probabilístico, consiguiendo reducir significativamente el costo computacional de la búsqueda con una pequeña perdida de precisión. La naturaleza aleatoria de la técnica permite una comparación eficiente de secuencias muy largas para el descubrimiento de características relevantes.

Un caso de esta familia de funciones que usan protección aleatoria es el método HashFile [Zhang et al., 2011] fue propuesto para responder consultas kNN exactas en el espacio L_1 y L_2 . El método combina las ventajas de la proyección aleatoria y la lectura secuencial.

Proyección Aleatoria

La proyección aleatoria es un método de reducción de la dimensionalidad de los datos que es computacionalmente eficiente y suficientemente preciso. Dado un conjunto de datos S con N puntos de dimensión n y una matriz aleatoria $R_{n \times k}$, la proyección es calculada de la siguiente manera.

$$S'_{N \times k} = S_{N \times n} \times R_{n \times k} \quad (2.11)$$

La proyección resulta en un conjunto de datos S' k-dimensional con N puntos. La proyección aleatoria puede preservar la distancia L_1 o L_2 en un espacio reducido, lo que es especificado por el Lema Jonhson-Lindenstrauss.

Lema de Jonhson-Lindenstrauss: Dado $\epsilon > 0$ es un numero N , sea k un entero positivo tal que $k \geq k_0 = O(\epsilon^{-2} \log N)$. Para cada conjunto S de N puntos en \mathbb{R}^n , existe $f : \mathbb{R}^n \rightarrow$

\mathbb{R}^k tal que para todo $u, v \in S$, se tiene:

$$(1 - \epsilon) \| u - v \|^2 \leq \| f(u) - f(v) \|^2 \leq (1 + \epsilon) \| u - v \|^2 \quad (2.12)$$

Algunos ejemplos de aplicación fueron presentados en [Indyk and Motwani, 1998]. Por ejemplo, Indyk y Motwani mostraron que el Lema JL es útil para resolver el problema de búsqueda aproximada al vecino más cercano, donde después de algunos procedimiento de pre-procesamiento del conjunto de datos, consultas de este tipo son respondidas: Dado un punto arbitrario x y un punto $y \in P$, para cada punto $z \in P$ se satisface $\|x - z\| \geq (1 - \epsilon)\|x - y\|$.

Restricción de la distancia para consultas kNN usando L_1

Supongo que \mathcal{H} es una función *hash* derivada de $R_{n \times k}$ tal que esta mapea un objeto o de dimensión n en un valor unidimensional.

$$\mathcal{H}(o) = \lfloor \sum_{i=1}^n h_i \cdot o_i \rfloor \quad (2.13)$$

Dado que cada elemento h_i está dentro de \mathcal{H} para valores randomicos $\{-1, 0, 1\}$, se puede fácilmente alcanzar un *lower bound* para consultas con la distancia L_1 .

Limite Inferior para L_1 Dado dos puntos x y y de dimensión n , y una función *hash* $\mathcal{H} : \mathbb{R}^n \rightarrow \mathbb{R}^1$ con $h_i \in \{-1, 0, 1\}$, se tiene.

$$\| x - y \|_{L_1} \geq |\mathcal{H}(x) - \mathcal{H}(y)| - 1 \quad (2.14)$$

Así, una proyección aleatoria puede ser usada para particionar un gran volumen de datos en *buckets*, en un espacio unidimensional. Las páginas en disco pueden ser almacenados secuencialmente en orden creciente al valor *hash*. Así, dado un punto de consulta q y la distancia λ al vecino más próximo ya encontrados, si el valor *hash* de q y h_q , entonces de acuerdo con la relación presentada en la Ecuación 2.14, solo las páginas con valor *hash* entre $[h_q - \lambda, h_q + \lambda]$ necesitan ser accesadas. Los demás puntos fuera del intervalo pueden ser seguramente podados.

Construcción del índice

El método HashFile es construido siguiendo el esquema *top-down*. Inicialmente, una función *hash* es generada en el nodo raíz y se crea un archivo en disco para almacenar los datos. Dado que no es posible predecir la gama de valore *hash*, no se puede atribuir una colección de páginas de tamaño fijo con antecedencia. Así, apenas una página es reservada con un intervalo (∞, ∞) . El tamaño de la página es fijo B , indicando que ella puede acomodar hasta B puntos. Los primero B objetos pueden ser insertados con éxito en esa página. Ahora, un nuevo objeto puede ser insertado en base a su valor *hash*. Como nuevos objetos son insertados de forma continua, una

operación de división ocurre y esos intervalos *hash* se hacen menores. Finalmente, un objeto puede ser mapeado para un *bucket* lleno donde los puntos tienen un mismo valor *hash* y por tanto no puede ser dividido. Para insertar un nuevo objeto, se crea un nodo hijo con una nueva función *hash*. Los puntos en esa página son extraídos y son re-mapeados junto con un nuevo objeto en el nodo hijo.

La Figura 2.8 ilustra un ejemplo de la estructura del árbol del método HashFile, así como la estructura lógica de un nodo interno del árbol. El árbol es construido por la inserción continua de los datos. Cuando un *bucket* en el nodo padre no puede ser dividido más, nodos hijos son creados para particionar el *bucket* más denso. Por tanto, cada nodo interno contiene una lista de nodos hijos, como es ilustrado en la figura del ejemplo. Cada bloque representa una página con un intervalo *hash* $[l_i, h_i]$.

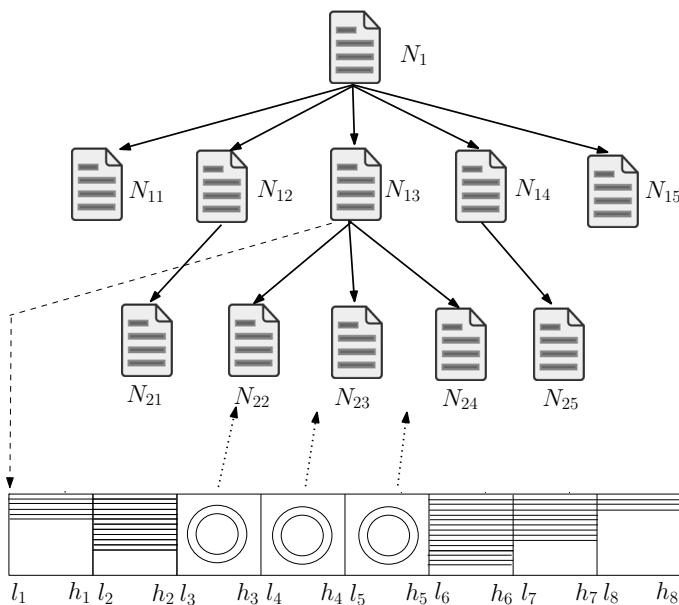


Figura 2.8: Estrutura del nodo HashFile. Figura adaptada de [Zhang et al., 2011].

Algoritmo 2.3: Consulta kNN aproximada usando HashFile

```

1 algorithm ApproximateNN( $q, \lambda$ )
2   init  $\delta$ 
3    $\delta := \text{ApproximateNNInNode}(q, \text{root}, \delta, \lambda)$ 
4   return  $\delta$ 
5 end.
6 procedure [ $\delta$ ] = ApproximateNNInNode( $q, \text{node}, \delta, \lambda$ )
7    $h_q := \text{node.hash}(q)$ 
8   for cada  $n$  child do
9      $wdist := |\text{child.hash\_value} - h_q|$ 
10    if  $wdist < \lambda$  then
11      insertar child en el heap ordenado por wdist
12    end if
13  end for
14  for cada página em disco  $p_i$  do

```

```

15     buscar la lista de paginas de  $p_{start}$  a  $p_{end}$  en disco entre los intervalos  $[h_q - \lambda, h_q + \lambda]$ 
16 end for
17 cargar las páginas entre  $[h_q - \lambda, h_q + \lambda]$  en el bloque  $B$ 
18 for cada objeto  $o$  en  $B$  do
19      $dist := \|o - q\|_{L_2}$ 
20     if  $dist < \delta$  then
21          $\delta := dist$ 
22     end if
23 end for
24 end.

```

Consulta kNN Aproximada

La Figura 2.9 (a) presenta las proyecciones de una consulta por similitud aproximada en Hash-File, donde el objeto de consulta q es mapeado usando las funciones hash (\mathcal{H}_1 e \mathcal{H}_2) en los dos niveles de la estructura. Su estructura lógica correspondiente es presentada en la Figura 2.9 (b). En este ejemplo, las proyecciones generan particiones diferentes en el espacio de búsqueda, donde cada función hash es utilizada para organizar todo el conjunto de datos, de manera que la proximidad de objetos en el espacio original es mantenida en la proyección. Durante el procesamiento de la consulta, para cada una de las proyecciones el objeto de consulta q es proyectado para el valor hash h_q , y apenas los objetos dentro del intervalo hash $[h_q - \lambda, h_q + \lambda]$ necesitan ser analizados. De este modo cada nivel contribuye en la precisión de los resultados ampliando el espacio de búsqueda.

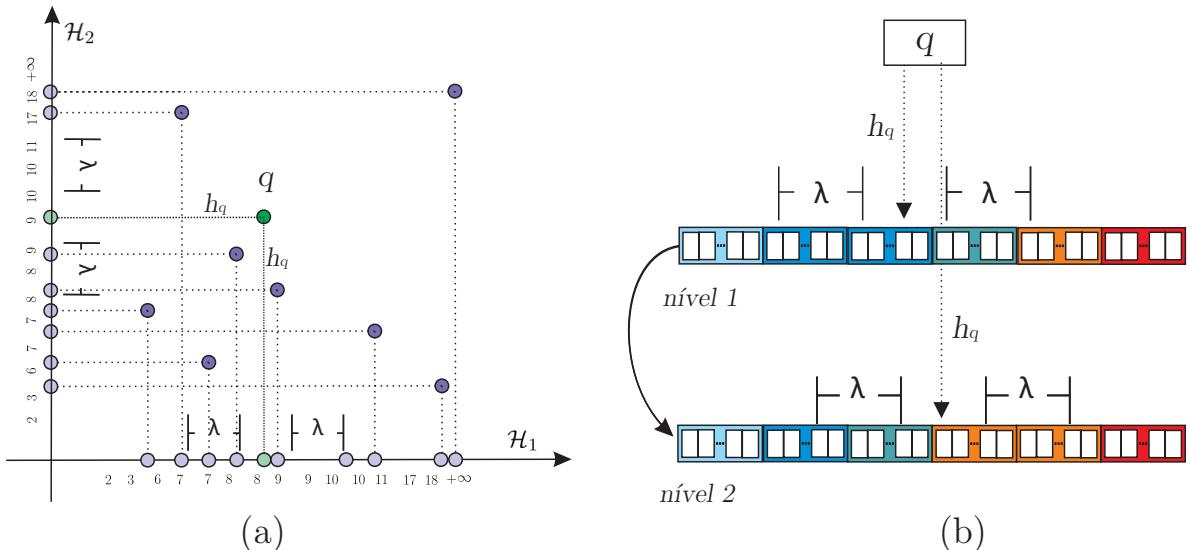


Figura 2.9: Representación de las proyecciones de una consulta por similitud aproximada en HashFile en (a), en (b) se verifica su estructura lógica correspondiente.

Por otro lado, es sabido que los hiperplanos que generar LSH tienden poco poder de discriminación conforme el número de funciones hash crece y además muchas

tablas son requeridas para un nivel adecuado de precisión. Esta ineficiencia es debido a su naturaleza de procesamiento de manera independiente a los datos, donde los hiperplanos *hash* son generados sin saber nada sobre la distribución de los datos. Este problema ha sido recientemente investigado en muchos métodos basados en *hash* para permitir aprender funciones *hash* adaptadas a la distribución de los datos. Estos métodos serán discutidos en la siguiente sección.

2.6. Consideraciones Finales

Como fue discutido en este capítulo, soluciones exactas para resolver el problema de búsqueda por similitud en altas dimensiones han sido estudiadas durante años, por otro lado algoritmos probabilísticos han sido poco explorados. De acuerdo con la literatura, es eficiente ejecutar consultas kNN exacta para datos en bajas dimensiones, mas en dimensiones altas las soluciones aproximadas son también una buena opción. El equilibrio entre la eficiencia y la eficacia se vuelve progresivamente más crítico conforme las dimensiones de los datos crece, debido a un fenómeno conocido como la “maldición da la alta dimensionalidad”.

Muchas técnicas exactas y aproximadas, como los Métodos de Acceso Métricos (MAMs) y los métodos basados en *Locality Sensitive Hashing* (LSH) y sus extensiones, fueron propuestas para resolver búsquedas en altas dimensiones. Entre ellas, los métodos basado en LSH son de las pocas técnicas con garantía teórica de costo sub-lineal. No obstante, muchas de sus implementaciones actuales presentan dificultades para ser implementados en problemas reales, como por ejemplo, tareas de minería de datos que dependen de algoritmo de búsqueda al vecino más próximo. He aquí el potencial de los métodos aproximados de búsqueda kNN.

Aún considerando soluciones aproximadas como una vía para mejorar la precisión y desempeño de las consultas por similitud, problemas como la búsqueda AllkNN en altas dimensiones aun son considerados poco prácticos debido a su alto costo computacional. Estudios recientes, promueven el uso de la Red Neuronal Convolutiva (CNN) con técnicas de *hashing* para mejorar la precisión de la búsqueda de los k-vecinos más cercanos - KNN. Sin embargo, aun hay retos que resolver para encontrar una solución práctica y eficiente para indexar este tipo de características, como es descrito en el Capítulo 6.

Minería de Datos & Deep Learning

3.1. Consideraciones Iniciales

Los rápidos avances en las tecnologías de recolección y de almacenamiento permitieron el almacenamiento de grandes cantidades de datos. Sin embargo, se probó que extraer conocimiento puede ser extremadamente difícil. En este contexto, la minería de datos combina métodos tradicionales de análisis de datos con algoritmos sofisticados para el procesamiento de grandes volúmenes de datos [Tan et al., 2005].

La Minería de Datos (*Data Mining*) es un área de investigación dentro en un contexto más amplio llamado Descubrimiento del Conocimiento en Bases de Datos (*KDD – Knowledge Discovery in Databases*), cuyo objetivo principal es extraer de un conjunto de datos, el conocimiento a ser utilizado en procesos decisarios. Más específicamente, los principales objetivos de los métodos de minería de datos son una descripción de un conjunto de datos y una predicción de valores futuros de interés basado en conocimiento previo de un banco de datos [Fayyad et al., 1996].

Como se contrastará en la siguiente sección, el enfoque de la minería de datos es la extracción de patrones, propiedades desconocidas en los datos. En cambio el enfoque del aprendizaje de maquina (*Machine Learning*) es la generación de modelos de predicción, basado en propiedades conocidas (experiencia) aprendidas de los datos de entrenamiento. Por otro lado el aprendizaje profundo (*Deep Learning*) es un subconjunto de del área de Aprendizaje de Maquina. El aprendizaje profundo es un mecanismo de aprendizaje jerárquico (en profundidad) que utiliza un conjunto de algoritmos para diversas tareas de aprendizaje de maquina, mientras que la minería de datos es en su mayoría un proceso de extracción de conocimientos que utiliza varios algoritmos que no necesariamente son “profundos”. Del mismo modo, el aprendizaje profundo puede ser utilizado para la minería de datos [Buduma, 2016].

Para hacer frente a problemas de aprendizaje basado en la experiencia tenemos que utilizar un tipo muy diferente de enfoque. Muchas de las cosas que aprendemos en la escuela tienen mucho en común con los programas informáticos tradicionales. Aprendemos cómo multiplicar números, resolver ecuaciones, y hacer derivadas mediante la guía de un conjunto de instrucciones. Pero las cosas que aprendemos a una edad muy temprana, las cosas que nos parecen naturales, se aprenden con el ejemplo, no por fórmulas [Buduma, 2016].

Deep Learning es un subconjunto de un campo más amplio de la inteligencia artificial llamado aprendizaje de maquina o *Machine Learning*, que se basa en la idea de aprender con el ejemplo. En lugar de enseñar a una computadora las reglas para resolver un problema, le damos un modelo con el que se puede evaluar ejemplos y un pequeño conjunto de instrucciones para modificar el modelo cuando se comete un error. Esperando que, con el tiempo, un modelo fuera capaz de resolver el problema con gran precisión [Buduma, 2016].

En la Sección 3.2 son discutidos los principales conceptos de Minería de Datos y Descubrimiento del Conocimiento en Bases de Datos (KDD). En la Sección 3.3, es presentado los conceptos de Aprendizaje de Máquina (*Machine Learning*) y Redes Neuronales. En la Sección 3.3.2, se analizan los conceptos de Redes Neuronales Convolutivas (CNNs) así como las principales aplicaciones y casos de éxito. La Sección 3.3.5, se analiza la teoría de Auto-codificadores y en la Sección 3.4 se presentan las consideraciones finales del capítulo.

3.2. KDD y Minería de Datos

Descubrimiento de Conocimiento en Base de Datos (KDD) - es el proceso de: primero, a partir de los datos, identificar patrones válidos, nuevos, potencialmente útiles y comprensivos. Segun [Fayyad et al., 1996], este proceso está compuesto de cinco etapas: selección de los datos; pre-procesamiento y limpieza de los datos; transformación de los datos; Minería de los Datos (*Data Mining*); e interpretación y validación de los resultados. La interpretación entre estas diversas etapas puede ser observada en la Figura 3.1, siendo que las tres primeras pueden ser interpretadas como un análisis exploratorio de los datos.

Los principales pasos para el proceso de KDD son:

1. **Limpieza de datos:** Las bases de datos del mundo real a menudo tienen datos incompletos, con ruido e inconsistentes que pueden dañar el análisis dificultando la detección de patrones. Los procedimientos de limpieza de datos trabajan para preparar los datos para los siguientes pasos en el procesos KDD mediante el llenado de los valores que faltan, corrigiendo los datos con ruido, identificando y removiendo valores atípicos, y resolviendo inconsistencia.
2. **Integración de datos:** En algunas circunstancias, los datos de múltiples fuentes

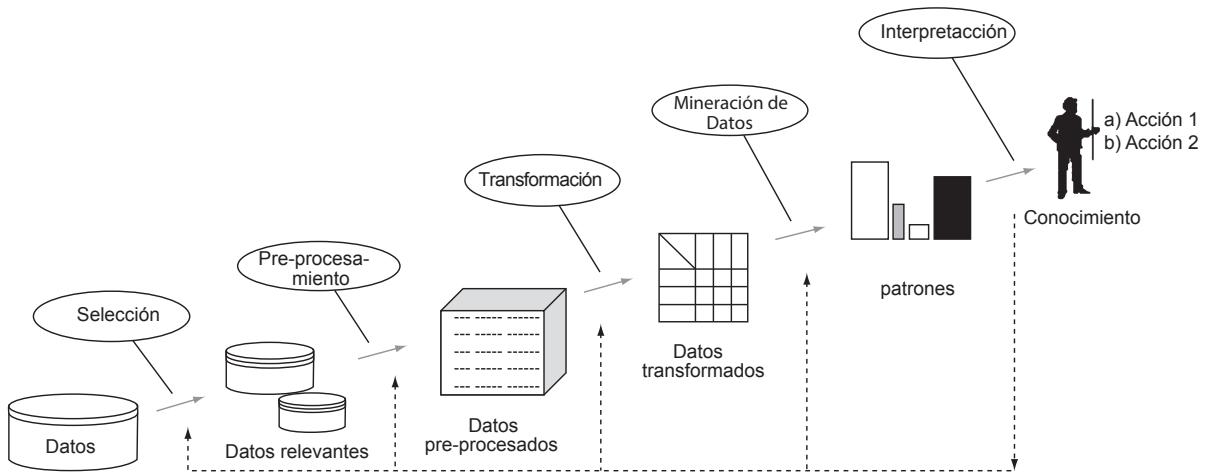


Figura 3.1: Etapas del proceso KDD. Figura adaptada de [Fayyad et al., 1996].

deben fusionarse y transformadas en formas apropiadas para ser incluidas en el mismo análisis. Por lo tanto, este paso implica técnicas para integrar correctamente múltiples bases de datos, cubos de datos o archivos en un almacenamiento de datos.

3. **Selección de datos:** Este paso corresponde a la identificación de datos relevantes para la tarea de análisis y su recuperación a partir de las bases de datos.
4. **Transformación de datos:** En este paso, los datos son transformados o consolidados en adecuadas formas para la extracción usando operaciones, tales como el resumen, la agregación, la generalización o la normalización.
5. **Minería de datos:** Este paso es fundamental en el proceso KDD, donde se aplican técnicas computacionales para extraer patrones desconocidos y útiles a partir de los datos.
6. **Evaluación del patrón:** En este paso, se utilizan medidas interesantes para identificar los patrones que representan el conocimiento.

La etapa de minería de datos incluye la definición de las tareas de minería a ser realizadas, la elección del algoritmo a ser aplicado y la extracción de patrones de interés. En la etapa siguiente, los patrones son interpretados y validados, y si los resultados no fueran satisfactorios (validos, nuevos, útiles y comprensibles), el proceso retorna a uno de los dos estados anteriores. Caso contrario, el conocimiento descubierto es consolidado [Fayyad et al., 1996]. El proceso KDD se refiere a todo el proceso de descubrimiento de conocimiento útil para los datos, en cuanto a la Minería de los Datos se refiere a la aplicación de algoritmos para extraer modelos de los datos.

3.3. Aprendizaje de Máquina – *Machine Learning*

El Aprendizaje de Máquina es un subconjunto de Inteligencia Artificial que incluye técnicas estadísticas que permiten a las máquinas generar modelos de predicción basado en propiedades conocidas aprendidas de los datos de entrenamiento, esta categoría incluye el aprendizaje profundo [[Smola and Vishwanathan, 2008](#)]. El Aprendizaje de Máquina se puede clasificar en 3. Aprendizaje Supervisado, Aprendizaje no supervisado, y Aprendizaje Semi supervisado

El aprendizaje supervisado es una técnica de aprendizaje que utiliza datos de entrenamiento y sobre la observación de estos crea una función capaz de predecir el valor de un nuevo dato y sobre la observación de estos crea una función capaz de predecir el valor de un nuevo dato que la función no haya visto, la supervisión en este tipo de aprendizaje se da mediante instancias etiquetadas en un conjunto de datos de entrenamiento.

Clasificación

La clasificación trabaja con conjunto de datos ya participados en grupos, categorías o clases. Este conjunto de datos sirve para entrenar un algoritmo de clasificación. Por ejemplo se tiene una base de datos de correos electrónicos, cada correo tiene un atributo que indica si este es o no un spam, estos datos sirven como datos de entrenamiento para un algoritmo de clasificación, el cual quiere determinar si un nuevo correo es o no spam.

Para clasificar se pueden usar algoritmos basados en Regresión Logística, en Maquinas de Soporte Vectorial y Redes Neuronales. Nuestro interés son los algoritmos basados en redes neuronales y lo detallamos a continuación.

3.3.1. Redes Neuronales

Las redes neuronales son modelos que simulan el sistema nervioso humano. La clave para la eficacia de una red neuronal es la arquitectura usada para organizar las conexiones entre nodos. Existe una amplia variedad de arquitecturas, a partir de una red con una sola capa, y otras redes mas complejas con múltiples capas [[Aggarwal, 2015](#)].

Una arquitectura de red o topología juega un papel importante para la clasificación, la topología óptima dependerá del problema en cuestión. A menudo el conocimiento del dominio del problema que podría ser de carácter informal o heurística puede incorporarse fácilmente en la arquitectura de redes a través de opciones, como número de capas ocultas, unidades, conexiones de retroalimentación. Por lo tanto el establecimiento de la topología de la red es la selección del modelo heurístico. La facilidad práctica en la selección de modelos (topologías de red) y la estimación de parámetros (formación a través de propagación hacia atrás) permiten

a los diseñadores de clasificadores probar modelos alternativos de manera bastante simple. [Duda et al., 2001]

En la Figura 3.2 se puede visualizar la arquitectura de un perceptrón. El perceptrón contiene dos capas de nodos, los cuales corresponden a los nodos de entrada y al nodo de salida, el número de nodos de entrada es exactamente igual a la dimensionalidad d de una instancia, cada nodo de entrada recibe y entrega un atributo numérico al nodo de salida. Los nodos de entrada solo transmiten valores de entrada para el nodo de salida, el nodo de salida es el único que realiza una función matemática en sus entradas.

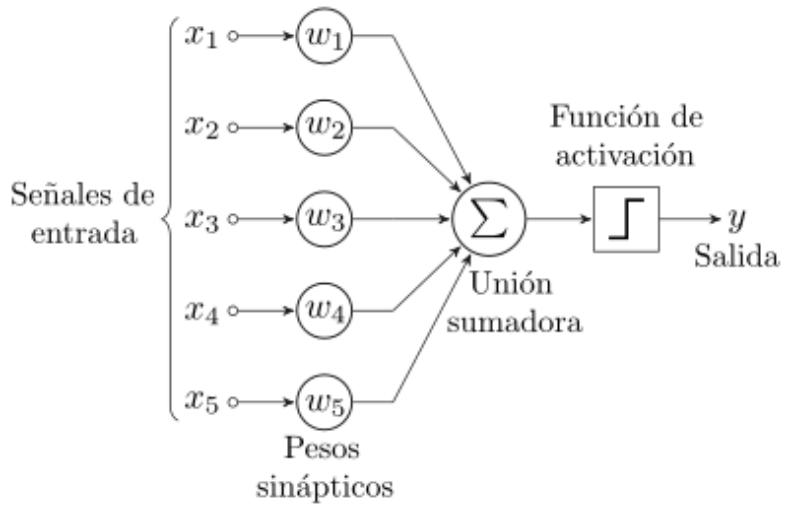


Figura 3.2: Perceptrón [Aggarwal, 2015].

En el caso de una red neuronal de múltiples capas (Figura 3.3), se tiene una capa de entrada, una capa de salida, y una o más capas intermedias llamadas capas ocultas. Lo mismo que antes, las salidas multiplicadas por sus pesos, es la entrada de la siguiente capa, sin olvidar que en cada nodo se aplica una función de activación, los nodos de salida son quienes dicen si la instancia pertenece a una clase o no, de acuerdo a la función de activación estos tomarán valores de 0 a 1, o de 1 a +1.

Para entrenar a una neurona se realizan dos fases.

Fase Fordward: en esta fase se introducen los valores de una instancia en la capa de entrada de la red, los pesos se toman de manera aleatoria, esta fase se realiza hacia adelante, comenzando en la capa de entrada y terminando en la capa de salida, los valore de la capa de salida son comparados con la etiqueta de la instancia para ver si se tiene un error.

Fase Backward: el objetivo de esta fase es corregir los errores en cada nodo. Recorre la red desde la capa oculta hacia la capa de entrada, en cada nodo se calcula el error y se actualizan los pesos de entrada que recibieron.

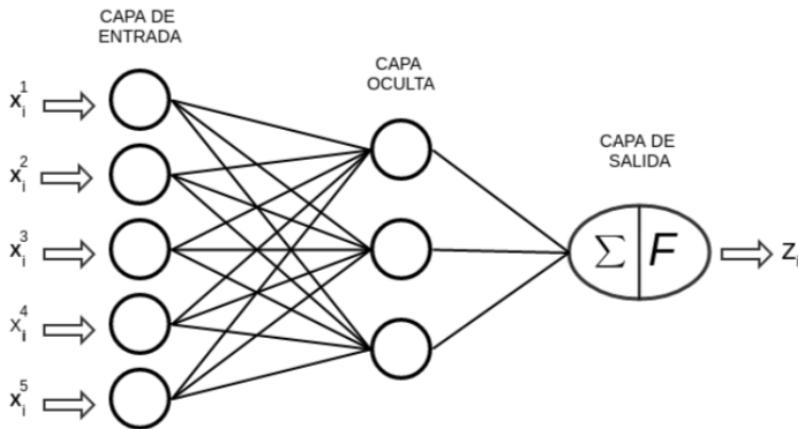


Figura 3.3: Perceptrón Multicapa [Aggarwal, 2015].

El proceso de *Fordward* y *Backward* se repiten para cada instancia, al finalizar se vuelve a repetir todo el proceso para toda la base de datos de entrenamiento, hasta un numero n de épocas si es necesario.

3.3.2. Redes Neuronales Convolutivas (CNN)

Una red neuronal convolucional (CNN) es una arquitectura jerárquica [LeCun, 1998] que consta de varias capas convolucionales apiladas (opcionalmente seguido por una capa de normalización y una capa de agrupación), capas completamente conectadas y una capa de salida en la parte superior. Las capas convolucionales generan mapas de características por filtros convolucionales lineales seguidos por funciones de activación no lineales (Rectificador, Sigmoid, TanH, etc.). La capa completamente conectada (*fully connected layer*) tiene conexiones completas a todas las activaciones en los mapas de características y el vector unidimensional resultante se puede alimentar en la capa de salida para la optimización de la función de pérdida.

Hay dos etapas principales para entrenar la red neuronal convolucional: una etapa *fordward* y una etapa de *backward*. En primer lugar, la etapa de *fordward* es representar la imagen de entrada con los parámetros actuales (pesos y sesgo/bias) en cada capa. A continuación, la salida de la última capa se utiliza para calcular la función de pérdida con las etiquetas de verdad. En segundo lugar, basándose en el costo de la pérdida, la etapa *backward* calcula los gradientes de cada parámetro con reglas de cadena. Todos los parámetros se actualizan en función de los gradientes y se preparan para el siguiente cálculo directo. Después de suficientes iteraciones de las etapas *fordward* y *backward*, la red podría ser optimizada. La red neuronal convolucional ha sido aplicada en diversas aplicaciones de visión por computadora y ha demostrado sus ventajas significativas y alto rendimiento.

3.3.3. Tipos de capas CNN

En esta sección presentamos una visión general de los diferentes tipos de capas y luego revisaremos brevemente las aplicaciones de visión computacional basadas en CNN.

Capas convolucionales(Convolucional layers): Las capas convolucionales de la arquitectura CNN utilizan k filtros (o núcleos) para envolver la imagen de entrada para generar k mapas de características. Hay tres ventajas principales de la operación de convolución [Zeiler, 2013]: en primer lugar, el mecanismo de reparto de parámetros se utiliza en capas convolucionales de tal manera que el número de parámetros podría reducirse significativamente. En segundo lugar, la conectividad local aprende correlaciones entre píxeles vecinos. En tercer lugar, es invariante a la ubicación del objeto. Debido a estos beneficios introducidos por la operación de convolución, algunos trabajos de investigación bien conocidos también lo utilizan como un reemplazo de las capas totalmente conectadas para acelerar el proceso de aprendizaje [Szegedy, 2015, Oquab, 2015].

Capas de agrupamiento(Pooling layers): Una capa de agrupación es una capa opcional que sigue una capa convolucional que sub-muestrea su entrada. La agrupación media(*average pooling*) y la agrupación máxima(*max pooling*) son las operaciones de agrupación más utilizadas. La razón para utilizar una operación de agrupación en la red neuronal convolucional es que: en primer lugar, puede reducir las dimensiones de la salida y el número de parámetros de la red, manteniendo la información más destacada. En segundo lugar, una operación de agrupación también proporciona una invariancia básica para la traducción (desplazamiento) y la rotación. Para un *max pooling* y *average pooling*, Boureau et al. [Boureau, 2010] proporcionó un análisis teórico detallado de su funcionamiento. Scherer et al. [Scherer, 2010] realizó una comparación entre las dos operaciones de agrupación y encontró que *max pooling* puede conducir a una convergencia más rápida, selección de características invariantes superiores y mejorar la generalización.

Capas completamente conectadas(Fully-connected layers): Después de varias capas convolucionales y de agrupación máxima, el razonamiento de alto nivel en la red neuronal convolucional se realiza a través de las capas completamente conectadas. Una capa completamente conectada toma todas las neuronas de la capa anterior (ya sea totalmente conectado, agrupación o convolucional) y lo conecta a cada neurona que tiene. Las capas completamente conectadas no están espacialmente localizados , ya que los mapas de características de entrada se convierten en un vector de características unidimensional. El vector de característica unidimensional podría alimentar el *forward* del vector en una capa de pérdida o tomarlo como una representación característica para el procesamiento de seguimiento [Girshick, 2014]. El inconveniente de la capa totalmente conectada es que contiene muchos parámetros, lo que da lugar a grandes costes computacionales y de

almacenamiento. Por lo tanto, GoogleNet [Szegedy, 2015] diseñó una red profunda y amplia, manteniendo el costo computacional constante, cambiando de arquitectura totalmente conectado a escasamente conectadas. La arquitectura *Network in Network* (NIN) [Lin, 2013] reemplaza la capa completamente conectada por una capa de agrupación media *average pooling* global.

3.3.4. Aplicaciones de Redes Neuronales Convolutivas (CNN)

Recientemente, el aprendizaje profundo, especialmente para las CNN, produjo el estado del arte de rendimiento en diversas aplicaciones de visión computacional, tales como clasificación de imágenes, búsqueda de imágenes, detección de objetos, segmentación de imágenes semánticas, estimación de la postura humana, etc.

Clasificación de imágenes: Krizhevsky et al. [Krizhevsky, 2012] establece un hito para la clasificación de imágenes a gran escala cuando el entrenamiento de una gran CNN en el conjunto de datos ImageNet [Deng, 2009], lo que demuestra que CNN podría funcionar bien en la clasificación de imagen natural. OverFeat [Sermanet, 2013] propuso un marco de ventana multi-escala y deslizante, que podría encontrar la escala óptima de la imagen y cumplir tareas diferentes simultáneamente, por ejemplo, clasificación de imágenes, detección de objetos y localización. Debido a que las CNN existentes requieren datos de imagen de tamaño fijo como entrada, el modelo SPP-Net eliminó esta restricción mediante una estrategia espacial de agrupación de pirámides en las CNN y mejoró la precisión de clasificación de una variedad de arquitecturas CNN a pesar de sus diferentes diseños. La propuesta posterior de VGGNet [Simonyan, 2014] y Google Net [Szegedy, 2015] mejoró significativamente el rendimiento de la clasificación de imágenes al aumentar el ancho y la profundidad de las arquitecturas de red.

Detección de objetos: Un esquema general para sistemas de detección de objetos de alto rendimiento es generar un gran número de propuestas de la región del objeto candidato y clasificarlas utilizando sus características CNN de alto rendimiento. El enfoque más representativo son las regiones con características CNN (RCNN) [Girshick, 2014]. Utiliza la búsqueda selectiva [Uijlings, 2013] para generar propuestas de región de objetos y extrae las características de CNN para cada región candidata. Las características se introducen en un clasificador SVM para decidir si las ventanas candidatas relacionadas contienen el objeto o no. RCNNs mejoró el punto de referencia de la detección de objetos por un gran margen, y se convirtió en el modelo base para muchos otros algoritmos prometedores [Hariharan, 2014, Zhu, 2015, Zhang, 2015]. Además, los RCNN originales eran computacionalmente caros, los trabajos recientes [He, 2014, Girshick, 2015] emplearon la estrategia de compartir convoluciones en las propuestas de la región para reducir el coste de cálculo. Los últimos frameworks de Fast RCNN [Girshick, 2015] y Faster RCNN [Ren, 2015] logran tasas casi en tiempo real utilizando redes muy profundas.

Recuperación de imágenes: :El éxito de AlexNet [Krizhevsky, 2012] sugiere que las CNN pueden ser usadas como extractoras de características de alto nivel y universales. Las características emergentes en las capas totalmente conectadas de la CNN pueden servir como una representación de imagen de alto nivel para la clasificación de imágenes. Motivado por esto, muchos estudios recientes hacen uso de las activaciones de las capas superiores totalmente conectadas en CNNs para la recuperación de imágenes [Gong, 2014, Liu, 2015a, Sharif Razavian, 2014, Wan, 2014, Sun, 2014, Babenko, 2014]. Investigaciones recientes también sugieren explorar las características de las capas convolucionales profundas en CNNs. Estas características tienen propiedades muy útiles: en primer lugar, se pueden extraer eficientemente de una imagen de cualquier tamaño y relación de aspecto. En segundo lugar, las características de las capas convolucionales tienen una interpretación natural como descriptores de regiones de imágenes locales correspondientes a campos receptivos de las características particulares. Por último, las operaciones de agrupación simple puede agregar mapas de características de capas convolucionales profundas en características de baja dimensión y altamente distintivas [Gong, 2014, Sharif Razavian, 2014, Babenko, 2014, Razavian, 2014, Babenko, 2015]. Las representaciones de imágenes basadas en CNN han demostrado sus resultados competitivos e incluso mejores en comparación con los métodos tradicionales de *visual words*, *BoW*, *VLAD*, y *Fisher Vector*.

Segmentación de imágenes semánticas: La segmentación de imágenes semánticas puede ser referida como un problema de clasificación o etiquetado a nivel de píxeles. Recientemente, CNNs y modelos gráficos probabilísticos se utilizaron para abordar esta tarea y produjo un progreso prometedor [Zheng, 2015, Liu, 2015b, Long, 2015, Chen, 2014a, Lin, 2015]. Los métodos de segmentación de imágenes semánticas basados en CNN usualmente convierten una arquitectura CNN existente construida para su clasificación a una red completamente convolucional (FCN). Esto se debe principalmente a que la arquitectura FCN acepta toda una imagen como entrada y realiza una inferencia rápida y precisa. Long et al. [Long, 2015] reemplazó las últimas capas completamente conectadas de una CNN por capas convolucionales, y obtuvo un mapa de etiqueta gruesa de la red clasificando cada región local en la imagen, luego realizó una deconvolución simple, que se implementa como interpolación bilineal, para el etiquetado a nivel de pixeles. DeepLab [Chen, 2014a] propuso un modelo similar basado en FCN que obtuvo mapas de puntuación densa dentro del framework FCN para predecir etiquetas en pixeles y refinado la etiqueta de mapa utilizando el campo aleatorio condicional (CRF) totalmente conectado. En lugar de usar CRF como una etapa de post-procesamiento, Lin et al. [Lin, 2015] entrenaron conjuntamente a la FCN y CRFs por medio de un entrenamiento eficiente en piezas.

Estimación de la postura humana: Estimar la postura humana mediante la localización de las articulaciones del cuerpo humano o hitos faciales es una tarea difícil, debido a los cambios en la pose, iluminación, oclusión y etc. Como las CNNs

han demostrado un rendimiento excepcional en la clasificación visual y la localización de objetos, la estimación de pose humana se puede formular como un problema de regresión basado en CNN hacia las articulaciones del cuerpo humano. Los proyectos más representativos [Toshev, 2014, Y et al., 2010] propusieron utilizar una CNN basados en regresores para razonar las posiciones de las articulaciones del cuerpo o los hitos faciales. Esta red CNN puede ser vista como una especie de proceso holístico que toma la imagen completa como la entrada y la salida de la posición final de las articulaciones corporales o puntos de referencia faciales en la imagen sin utilizar ningún modelo gráfico explícito o detectores de partes. Los métodos de procesamiento basados en partes proponen detectar las partes del cuerpo humano individualmente, seguido de un modelo gráfico para incorporar la información espacial. En lugar de capacitar a la red utilizando toda la imagen como entrada, Chen et al. [Chen, 2014b] utilizaron los parches de la parte local para entrenar una CNN, con el fin de aprender probabilidades condicionales de la presencia parcial y las relaciones espaciales. Al incorporarse con modelos gráficos, el algoritmo ganó un rendimiento prometedor. Tompson et al. [Tompson, 2014] diseñaron arquitecturas ConvNet multi-resolución para realizar la regresión de probabilidad de mapa de calor para cada parte del cuerpo, seguido de un modelo gráfico implícito para promover aún más la consistencia conjunta. El modelo se amplió y mejoró [Tompson, 2014], lo que argumenta que las capas de agrupación en la CNN limitaría la exactitud de localización espacial y tratar de recuperar la pérdida de precisión causada por el proceso de agrupación. Además, Fan et al. [Fan, 2015] propuso una red convolucional neutral de doble fuente (DS-CNN) para integrar la visión holística y parcial en una arquitectura de dos CNN. Requiere parches y parches corporales como entradas para combinar la información local y contextual para una estimación más exacta de la postura.

3.3.5. Aprendizaje no Supervisado

En el aprendizaje no supervisado no se tienen instancias etiquetadas, es decir que se tiene un conjunto de datos, en la cual no existe ninguna clase, lo que se trata de hacer es un agrupamiento de estos datos para encontrar o descubrir dichas clases.

Auto-codificadores

Los auto-codificadores pertenecen a una clase de algoritmos de aprendizaje conocidos como aprendizaje no supervisado. A diferencia de los algoritmos supervisadas, los algoritmos de aprendizaje sin supervisión no necesitan información de la etiqueta para los datos. En otras palabras, nuestros datos sólo tienen de $x(i)$, pero no tienen los $y(i)$, que vendrían a ser la etiquetas de los datos [Le et al., 2015, Ng et al., 2016]. Un auto-codificador es una técnica muy utilizada en el aprendizaje no supervisado, aunque también haya sido utilizada de distintas maneras y con distintos objetivos.

3.3.6. Compresión de data

En [Le et al., 2015], se considera el siguiente ejemplo, se desea desarrollar un programa para enviar algunos datos del teléfono móvil a la nube. Para limitar el uso de la red, se debe optimizar cada *bit* de datos que vamos a enviar. Los datos son una colección de puntos, cada uno tiene dos dimensiones, como se ve en la Figura 3.4

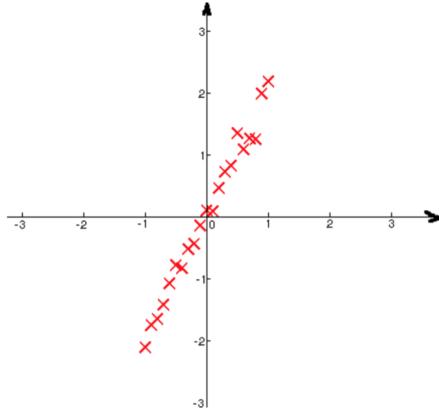


Figura 3.4: Puntos en un plano 2d
[Le et al., 2015]

En la Figura 3.4, las cruces rojas son los puntos de datos, el eje horizontal es el valor de la primera dimensión y el eje vertical es el valor de la segunda dimensión. Tras la visualización, notamos que el valor de la segunda dimensión es aproximadamente el doble que de la primera dimensión. Teniendo en cuenta esta observación, se puede enviar sólo la primera dimensión de cada punto de datos a la nube. Luego, en la nube, solo se necesita calcular el valor de la segunda dimensión, duplicando el valor de la primera dimensión. La compresión es con perdida, pero reduce el tráfico de red en un 50%. Ya que el tráfico de red es lo que tratamos de optimizar, esta idea parece razonable [Le et al., 2015].

El objetivo de los auto-codificadores es poder resolver el ejemplo anterior de manera sistemática. Formalmente, suponemos que tenemos un conjunto de puntos de datos $\{x(1), x(2), \dots, x(m)\}$, donde cada punto de datos tiene varias dimensiones. La pregunta es si hay una manera general de asignarlos a algún conjuntos de datos $\{z(1), z(2), \dots, z(m)\}$, donde z tiene una dimensión menor a x y los $z(i)$ pueden fielmente reconstruir las $x(i)$. Para responder a esto, se nota que en el ejemplo anterior, para enviar datos desde el teléfono celular a la nube, tiene tres pasos:

- Codificación: Desde el celular, se asigna la data $x(i)$ comprimida a $z(i)$.
- Envío: Se envía $z(i)$ a la nube.
- Decodificación: En la nube, se asigna desde la data comprimida $z(i)$ a $\tilde{x}(i)$, que es una aproximación de $x(i)$.

Para asignar los datos de un lado a otro de manera sistemática, definimos que z y \tilde{x} son funciones de entrada, de la siguiente manera:

$$z(i) = W_1 x(i) + b_1 \quad (3.1)$$

$$\tilde{x}(i) = W_2 z(i) + b_2 \quad (3.2)$$

Si $x(i)$ es un vector de dos dimensiones, puede ser posible visualizar los datos para encontrar W_1, W_2, b_1, b_2 analíticamente, donde W_1, W_2 son matrices bidimensionales de pesos y b_1, b_2 son el componente *bias*. En casos prácticos, es difícil encontrar esas matrices usando la visualización, por lo que es necesario utilizar el gradiente descendente [Bottou, 2012]. La meta es tener un $\tilde{x}(i)$ aproximado a $x(i)$, para esto se establece la siguiente función objetivo, que es la suma de diferencia de cuadrados entre $\tilde{x}(i)$ y $x(i)$:

$$\begin{aligned} J(W_1, b_1, W_2, b_2) &= \sum_{i=1}^m (\tilde{x}(i) - x(i))^2 \\ &= \sum_{i=1}^m (W_2 z(i) + b_2 - x(i))^2 \\ &= \sum_{i=1}^m (W_2 (W_1 x(i) + b_1) + b_2 - x(i))^2 \end{aligned} \quad (3.3)$$

En la Figura 3.5, Se observa como se trata de comprimir datos de 4 dimensiones a 2 dimensiones utilizando una red neuronal con una capa oculta. La función de activación de la capa oculta es no lineal. Si los datos fueran altamente no lineales, se podría añadir más capas ocultas a la red para tener un auto-codificador profundo.

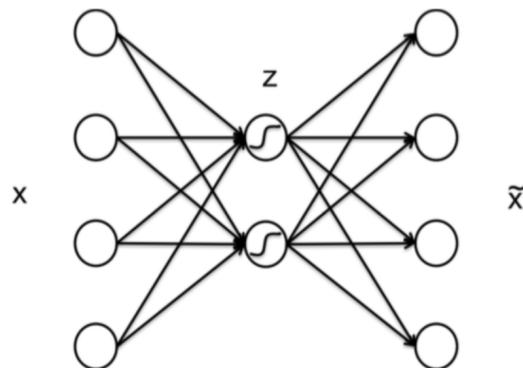


Figura 3.5: Arquitectura de auto-codificador no lineal
[Le et al., 2015]

3.4. Consideraciones Finales

En este capítulo fue presentada una visión general sobre minería de datos y *deep learning*, destacando los conceptos y aplicaciones que son de interés para este trabajo. En específico, fueron discutidas técnicas de aprendizaje supervisado y no supervisado basado en redes neuronales artificiales con énfasis en arquitecturas CNN y autocodificadores.

A pesar de las recientes investigaciones en métodos de clasificación, las redes neuronales basados en arquitecturas CNN pueden ser la mejor opción en muchos dominios de aplicación debido a su eficiencia en precisión y escalabilidad. En problemas de gran escala con grandes volúmenes de datos las técnicas de aprendizaje profundo, se muestran más adecuadas conforme el número de ejemplos por clases crece. Además, para tareas de procesamiento y clasificación de los datos ahora se puede usar el alto rendimiento de las GPUs.

Teoría Fractal

4.1. Consideraciones Iniciales

Un fractal es definido como un objeto que presenta aproximadamente las mismas características independientemente de la escala donde es analizada, es un objeto que se parece a si mismo. Por otro lado, las partes del fractal son similares, exactas o estadísticamente al fractal completo. Esto es, a una escala mejor los detalles son similar a las características de una escala mayor [Schroeder, 1991, Traina et al., 2005].

Por ejemplo, el triangulo *Sierpinkski* es un fractal geométrico, que ha sido construido en un proceso iterativo, teóricamente infinito. En un triangulo equilátero ABC, donde primero se ha removido el triangulo central A', B', C', de cada uno de los tres triángulos restantes cuyos lados longitud igual a la mitad del lado de ABC, retiramos de nuevo el triángulo central. La figura 4.1 muestra los pasos iniciales del proceso de construcción de un triángulo de *Sierpinsk*. El triángulo restante tiene “agujeros” independientes de la escala y cada triángulo dentro de la primera es una “miniatura” de todo el triángulo. Hay muchas otras estructuras matemáticas definidas como fractal, como las curvas de *Koch*, el conjunto de *Cantor* y el conjunto de *Mandelbrot* que son presentados en la figura 4.2. Son también ejemplos de fractales en la naturaleza, por ejemplo: nubes, montañas, hortalizas, árboles, La costa de continentes, islas entre otros.

Los conceptos de fractal se han aplicado a varias tareas en el análisis de datos y la minería de datos. Una de ellas es la estimación de la dimensión intrínseca (D) de un conjunto de datos, que es relacionado con el concepto de dimensión embebida (E)

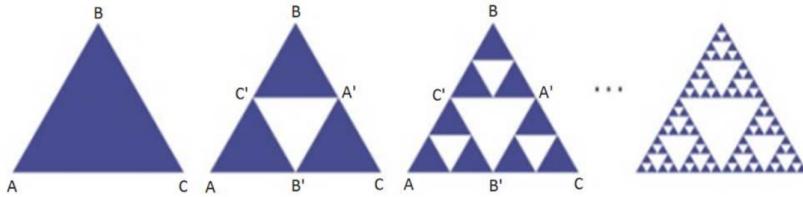


Figura 4.1: Pasos del proceso de construcción del triangulo de *Sierpinsky*

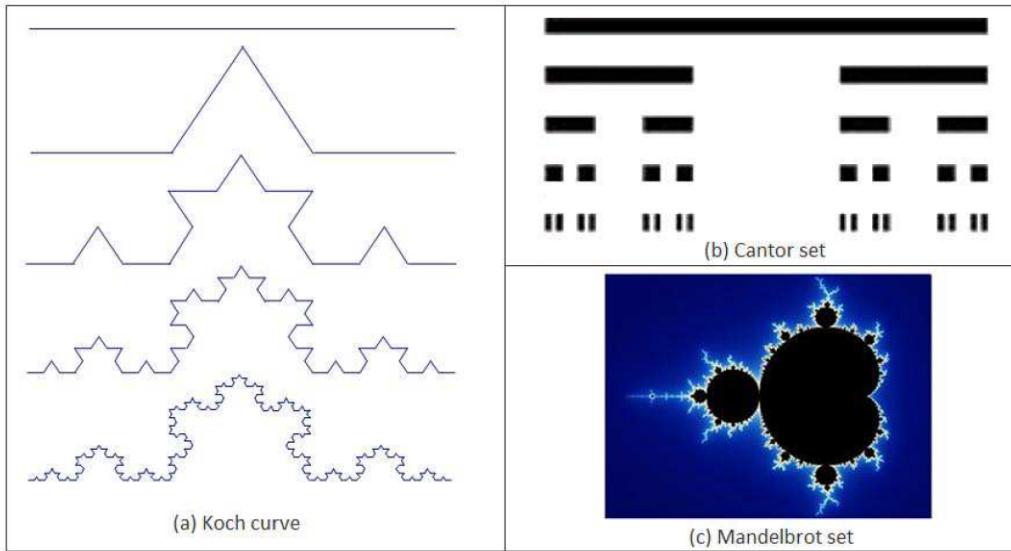


Figura 4.2: Algunos ejemplos de fractales (a y b) fractales geométricos y (c) fractales algebraicos

[Faloutsos and Kamel, 1994].

Definición 3.1 Dimensión embebida E: Dado un conjunto finito de datos A , la dimensión embebida $E \in N$ es el número de atributos que definen A . Es decir, E es la dimensión de el espacio en el que se encaja el conjunto de datos.

Definición 3.2 Dimensión intrínseca D: Dado un conjunto de datos finitos A , su dimensión intrínseca $D \in R^+$, es la dimensionalidad del objeto representado por los datos, independientemente de la dimensión del espacio en el que está embebido.

La dimensión intrínseca (D) es una medida de la cantidad de información que el conjunto de datos representa. Por ejemplo, la dimensión intrínseca de un conjunto de puntos distribuidos a lo largo de una linea es igual a uno; si el conjunto está embebido en un espacio dimensional superior, la dimensionalidad intrínseca continúa igual a uno tal como se ilustra en la Figura 4.3.

Faloutsos Kamel (1994) [Faloutsos and Kamel, 1994] propusieron el uso de la dimensión intrínseca como una herramienta para medir el comportamiento no

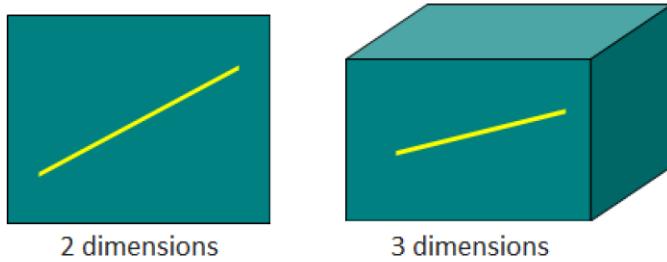


Figura 4.3: Una linea embebida en dos o tres dimensiones donde $D = 1$.

uniforme de conjuntos de datos reales. Además, los autores presentaron estudios empíricos para demostrar que los datos reales suelen tener un comportamiento de auto-similitud, que es una característica fundamental de los objetos fractales. Por lo tanto, la dimensión intrínseca D de un conjunto de datos real puede calcularse usando la dimensión fractal.

La dimensión intrínseca basada en la dimensión fractal ha sido empleada como una herramienta para el análisis de agrupamiento [Barbará and Chen, 2003], reglas de asociación temporal [Barbará et al., 2004], selección de atributos [Jr. et al., 2010a], series cronológicas [Chakrabarti and Faloutsos, 2002] y la minería de datos espaciales [Traina et al., 2001].

En este capítulo presentamos los principales conceptos relacionados con la teoría fractal que son usados en algunos de los métodos propuestos en esta tesis. La sección 2 muestra diferentes maneras de Calcular la dimensión fractal. El cálculo de correlación indicado por la correlación por la dimensión fractal se detalla en la Sección 3. La teoría fractal empleada para analizar datos en la sección 3.

4.2. Dimensión fractal

Los fractales suelen tener características inusuales que pueden considerarse paradojas. Por ejemplo, el triángulo de *Sierpinski* tiene un perímetro infinito (proporcional a $\lim_{i \rightarrow \infty} (1 + 1/2)^i$) y nulo (proporcional a $\lim_{i \rightarrow \infty} (3/4)^i$) ya que a cada iteración de su proceso de construcción, que teóricamente es infinito, su perímetro aumenta y su área disminuye. Debido a estas propiedades, este fractal no puede ser considerado un objeto euclíadiano unidimensional (ya que su perímetro no es finito) ni un objeto euclíadiano bidimensional (puesto que su área es nula). Así, es posible considerar una dimensionalidad de fraccionamiento llamada fractal [Di Ieva, 2016]. Intuitivamente, la dimensión fractal del triángulo de *Sierpinski* está entre un valor de 1 y 2. Matemáticamente, el valor preciso es 1,58.

Hay varias definiciones para la dimensión fractal, que se presentan brevemente en este sección. La medida básica de la dimensión fractal se dedica a los fractales denominados exactamente auto-similares. Este tipo de fractal se compone de M

rélicas que son una versión reducida $1 : s$ del fractal original.

Definición 3.3 Dimensión fractal D: Sea M el número de rélicas y s el factor de escala por el que se reduce cada réplica, la dimensión fractal D de un fractal auto-similar es definido en un espacio E -dimensional como:

$$D \equiv \frac{\log M}{\log s} \quad (4.1)$$

El triángulo de *Sierpinski*, por ejemplo, es un fractal exactamente auto-similar, porque su regla de construcción genera tres rélicas en escala $1 : 2$ en cada iteración. Por lo tanto, la dimensión fractal de *Sierpinski* es $D = \frac{\log 3}{\log 2} \approx 1,58$. Similarmente, en la Figura 4.2 el conjunto de Cantor conjunto y la curva de Koch son fractales auto-similares con la dimensión fractal $D = \frac{\log 2}{\log 3} \approx 0,63$ y $D = \frac{\log 4}{\log 3} \approx 1,26$ respectivamente.

Esta definición de la dimensión fractal D es adecuada para Fractales auto-similares matemáticamente exactos. Sin embargo, para conjuntos de datos llamados Fractales auto-similares estadísticamente, que no tienen reglas de construcción bien definidas, es más apropiado calcular la dimensión fractal mediante el método de recuento de cajas (box-counting) [Schroeder, 1991], que define la Correlación de Dimensión Fractal D_2 tal como se presenta en Ecuación 4.2

Definición 3.4 Correlación de Dimensión Fractal D_2 : Dado un conjunto de datos auto-similar en el rango de escalas $[r_1, r_2]$, su dimensión fractal de correlación $D_2 \rightarrow [\mathbb{R}^+]$ se mide como

$$D_2 \equiv \frac{\partial \log(\sum_i C_{r,1}^2)}{\partial(r)} \quad r \in [r_1, r_2] \quad (4.2)$$

donde r es el lado de las celdas en una (hiper) cuadricula que divide el espacio del conjunto de datos, y $C_{r,i}$ es el recuento de puntos en la i -ésima celda.

De forma práctica, el valor derivado que define la dimensión fractal D_2 puede obtenerse mediante la construcción de la gráfica de recuento de bloques, que representa los valores de $\sum_i C_{r,1}^2$ y $\log(r)$ en un gráfico. Para los conjuntos fractales, la curva resultante es lineal en un intervalo (r_1, r_2) y la dimensión fractal D_2 se estima por la pendiente de la línea que mejor se ajusta al intervalo analizado.

La figura 4.4 muestra un conjunto de 6561 puntos en el triángulo de *Sierpinski* y la gráfico en la escala log-log de la suma de la ocupación cuadrada $\sum_i C_{r,1}^2$ contra el tamaño de la celda (radio) r .

La dimensión fractal fue calculada por el algoritmo box-counting(), el cual tiene un coste lineal respecto al número de elementos en el conjunto de datos [Traina et al., 2010]. Este algoritmo es descrito en la siguiente sección.

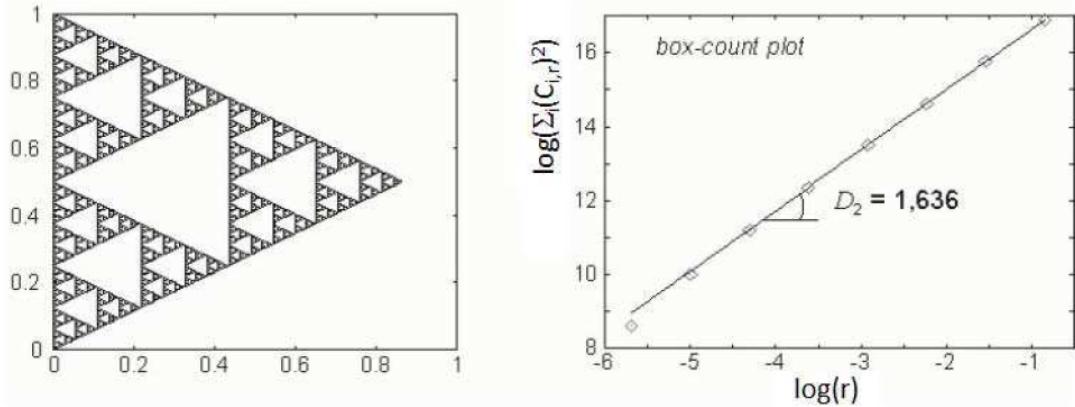


Figura 4.4: El gráfico de conteo de cajas para el triangulo de *Sierpinki*.

4.3. Algoritmo de Calculo de la Dimensión Fractal

Esta sección presenta un algoritmo para calcular la dimensión fractal D de cualquier conjunto dado de puntos en cualquier espacio E -dimensional. Una manera práctica de estimar D a partir de un conjunto de datos espaciales está utilizando el método de recuento de cajas”[[Schroeder and Fractals](#),]. Teóricamente, este método da una aproximación cercana a la dimensión fractal [[Traina et al., 2010](#) [Traina Jr et al., 1999](#)]. Uno de los mejores algoritmos publicados para calcular D de un conjunto de datos es un algoritmo $O(N \log(N))$, donde N es el número de puntos en el conjunto de datos [[Belussi and Faloutsos, 1998](#)].

Sin embargo, hemos desarrollado un nuevo, muy rápido, $O(N)$ algoritmo para su implementación, que se presenta a continuación. Considere el espacio de direcciones de un conjunto de puntos en un espacio E -dimensional e imponga una E -rejilla con cuadrículas de tamaño de lado r . Centrándose en la i -ésima célula, sea $C_{r,i}$ el recuento ('ocupaciones') de los puntos 2 en cada celda. Luego, calcule el valor $S(r) = iC_{r,i}$. La dimensión fractal es la derivada de $\log(S(r))$ con respecto al logaritmo del radio.

Al asumir conjuntos de datos auto-similares, esperamos que esta derivada resulte en un valor constante. Así, podemos obtener la dimensión fractal D de un conjunto de datos que traza $S(r)$ en escalas log-log para diferentes valores del radio r , y calcular la pendiente de la línea resultante. Es necesario procesar $S(r)$ para una cantidad R de valores de r , por lo que podemos lograr una aproximación estadística adecuada de la recta. Para evitar la lectura del conjunto de datos de nuevo para cada valor del radio, proponemos crear una estructura de cuadrícula de niveles múltiples, donde cada nivel tiene un radio de la mitad del tamaño del nivel anterior ($r = 1, 1/2, 1/4, 1/8, etc.$). Cada nivel de la estructura corresponde a un radio diferente, por lo que la profundidad de la estructura es igual al número de puntos en el gráfico resultante.

La estructura se crea en la memoria principal, por lo que el número de puntos en el gráfico está limitado por la cantidad de memoria principal disponible. Si este gráfico es lineal para un rango adecuado de radios, el conjunto de datos es un fractal y su dimensión fractal D es la pendiente de la línea de ajuste de este gráfico. El algoritmo propuesto es lineal en el número de puntos en el conjunto de datos. La complejidad computacional del algoritmo es $O(N * E * R)$, donde N es el número de objetos en el conjunto de datos, E es la dimensionalidad de inclusión y R es el número de puntos utilizados para representar la función $S(r)$. Esto muestra que el algoritmo es escalable a conjuntos de datos de cualquier tamaño.

Para cada lado de celda dado r , sólo se mantienen las celdas que tienen al menos un punto ya procesado, contando la suma de ocupaciones Cr, i de esta celda. De esta manera, cada nuevo punto está directamente asociado a una célula en cada nivel, sin necesidad de ser comparado con los puntos de lectura anteriores. La figura 2 muestra la estructura utilizada en el algoritmo para conjuntos de datos bidimensionales y tridimensionales.

El lado de celda más grande del espacio de puntos genera $2n$ células. En el siguiente nivel, cada célula se divide en otras $2 n$ células, y así sucesivamente. Dado que siempre se conoce la posición de cada célula en el espacio, cada célula está representada por: la suma de ocupaciones Cr, i en esta celda, y los punteros a las celdas en el siguiente nivel cubierto por esta celda (véase la Figura 2). Esta estructura es una especie de multi-dimensional "quad-árbol"(oct-tree para un espacio 3D, o E-dim-árbol). La Figura 3 muestra un ejemplo de esta estructura para un conjunto de datos con cinco puntos en tres niveles en un espacio bidimensional.

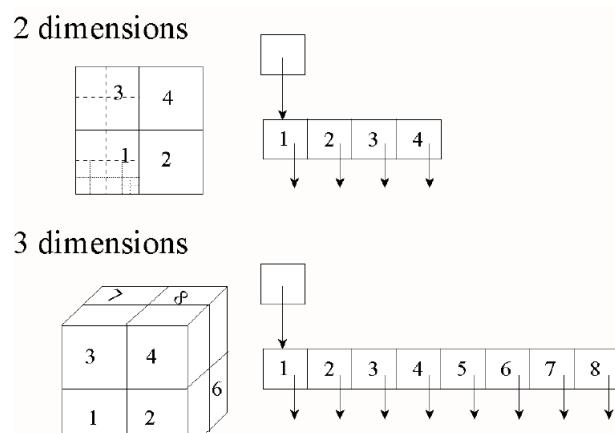


Figura 4.5: Representación de celdas de cuadricula en espacios de 2 y 3 dimensiones.

Observe que se agregan nuevas celdas a la estructura cuando se solicita. Así, sólo

se crean células ocupadas por al menos un punto ($C_r, i > 0$). El algoritmo procesa los puntos establecidos sólo una vez, por lo que es realmente muy rápido. El algoritmo 1 resume este proceso de cálculo.

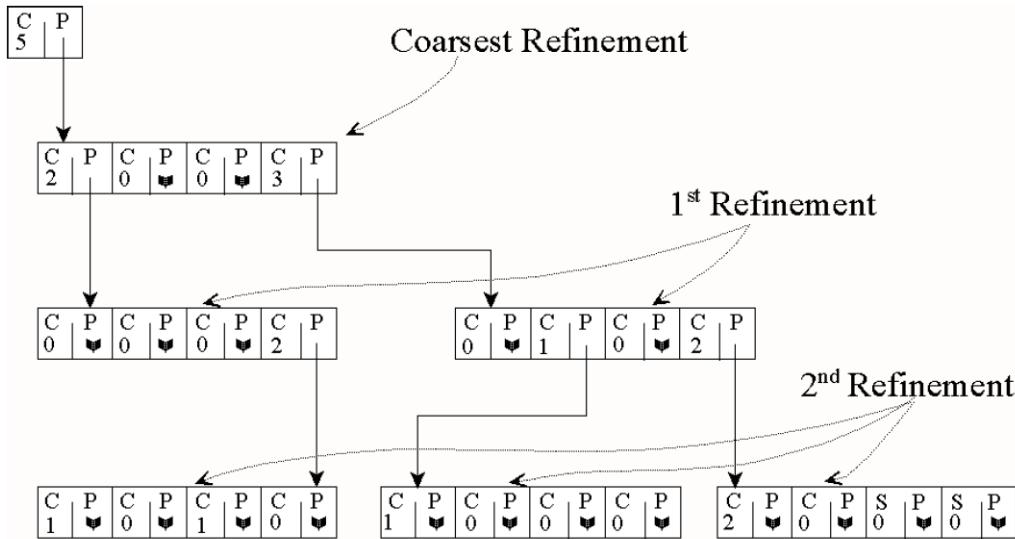


Figura 4.6: Ejemplo de la estructura de datos utilizada para calcular la Suma de Ocupaciones de un conjunto de datos con 5 puntos (con tres niveles de resolución)

Observe que se agregan nuevas celdas a la estructura bajo demanda. Así, sólo las células ocupadas Por al menos un punto ($C_{r,i} > 0$). El algoritmo procesa sólo los puntos establecidos Una vez, por lo que es realmente muy rápido. El algoritmo 4.1 resume este proceso de cálculo.

Algoritmo 4.1: Procesar la dimensión fractal D del dataset A(conteo de cajas aproximadas)

Entrada: Normalizar el Dataset A(N filas, con E dimensiones/cada atributo)

Salida: Dimension fractal D

- 1: **para** Cada tamaño de grid deseada $r = 1/2^j, j = 1, 2, \dots, l$ **hacer**
 - 2: **para** Cada punto en el dataset **hacer**
 - 3: Decide cual celda del grid cae en (la i-esima celda)
 - 4: Incrementa la cuenta C_i
 - 5: **fin-para**
 - 6: Procesa la sumatoria de los ocupados $S(r) = \sum C_i^2$
 - 7: **fin-para**
 - 8: Imprime los valores de $\log(r)$ y $\log(S(r))$ generando un gráfico
 - 9: Retorna la parte linear del gráfico cuesta abajo(regresión linear) de las dimensiones fractales D del dataset A
-

A medida que aumenta el lado de la cuadricula, también aumenta el número

de punteros a celdas vacías. Por lo tanto, para los conjuntos de datos de alta dimensión vale la pena mantener las celdas como listas enlazadas en lugar de arrays. Implementamos esta estructura como un objeto, usando una matriz para conjuntos de datos con la dimensión de incrustación menos o igual a tres, y usando una lista vinculada para conjuntos de datos con mayor dimensionalidad.

4.4. Consideraciones Finales

Búsqueda aproximada vía *deep hashing*: Desempeño de métodos de representación y recuperación

5.1. Consideraciones Iniciales

La disponibilidad cada vez mayor de datos en diversos dominios ha creado la necesidad de desarrollar técnicas y métodos para descubrir el conocimiento a partir de volúmenes masivos de datos, motivando muchos trabajos de investigación en bases de datos, *machine learning* y comunidades de recuperación de información. Esto ha impulsado el desarrollo de técnicas escalables y eficientes para organizar y recuperar este tipo de datos. *Similarity search* ha sido el enfoque tradicional para la recuperación de información. Aunque se han propuesto varios algoritmos de búsqueda de similitud para acelerar las consultas de similitud, la mayoría de ellos se ven afectados por la bien conocida "*curse of dimensionality*". Recuperar datos complejos causa problemas de estabilidad cuando la dimensionalidad de los datos es muy alta [Böhm et al., 2001]

Se han estudiado diferentes enfoques para resolver el problema de "*curse of dimensionality*". Una de las líneas de investigación es tratar de evitar el problema de dimensionalidad relajando la precisión de la consulta para acelerar el tiempo de consulta. Potencialmente, este enfoque es factible para aplicaciones que no requieren respuestas exactas y cuya velocidad es más importante que la precisión de búsqueda. Además, la definición del espacio métrico ya conduce a una aproximación de la respuesta verdadera, y por lo tanto una segunda aproximación en el tiempo de búsqueda puede ser aceptable [Chávez et al., 2001].

Se propusieron algoritmos de *approximate nearest neighbor search* basados en *hashing* para consultar conjuntos de datos de alta dimensión debido a su alta velocidad de recuperación y bajo costo de almacenamiento. Locality Sensitive Hashing (LSH) [Datar et al., 2004a] es una de las recientes técnicas basadas en hash propuestas para organizar y consultar datos de alta dimensión. De hecho, LSH es una de las pocas técnicas que proporciona análisis teóricos sólidos y pérdida predecible de precisión en los resultados. Sin embargo, existe una dependencia en los valores de los parámetros para los esquemas de búsqueda de similitud aproximada basados en LSH, que determinan el número de funciones hash y el número de tablas hash.

Para responder consultas de similitud, LSH busca solo regiones, que están representadas por *buckets*, a los que se aplica el hash del objeto de consulta (es decir, los candidatos de los *buckets* que contienen los objetos del conjunto de datos con una alta probabilidad de similitud con el objeto de consulta). Por lo tanto, no es necesario explorar completamente los datos de índice, y solo los objetos en los candidatos de los *buckets* que requieren un procesamiento adicional de acuerdo con la condición de consulta (por ejemplo, $d(x, q) \leq r$) [Ocsa and Sousa, 2010].

Hashing produce una representación compacta para documentos, para realizar la tarea como clasificación o base de recuperación en estos códigos hash. Cuando se supervisa el hash, los códigos se entregan usando etiquetas en los datos de entrenamiento. Inspirado por los avances recientes en Red Neuronal Convolutiva (CNN) [Krizhevsky et al., 2012], muchos métodos mejoran la precisión de la recuperación de similitud al usar CNN como extractor de características y luego crean un código hash compacta de preservación de similitud para una rápida recuperación de imágenes. En un trabajo reciente [Kelvin Lin, 2015], un método de hashing supervisado entrena el modelo con una capa oculta de *binary-hash* como características para tareas de clasificación de imágenes. Al combinar la extracción de características de imagen y el aprendizaje de *binary-code*, estos métodos han demostrado una precisión de recuperación muy mejorada. Sin embargo, existe un equilibrio entre el error de clasificación y el error de cuantificación: las activaciones de las capas inferiores son más comunes [Yosinski et al., 2014], por lo que el entrenamiento es más efectivo. Sin embargo, las capas inferiores tienen mapas de activación más grandes (muchos nodos), que son más difíciles de codificar lo que lleva a un compromiso.

5.1.1. Approximate Nearest Neighbors & Deep Hashing

Hashing para la Búsqueda del Vecino más Cercano

En primer lugar, es necesario definir un universo de objetos \mathbb{U} y una función $d : \mathbb{U} \times \mathbb{U} \rightarrow \mathbf{R}$ que mida la distancia entre dos objetos de \mathbb{U} . En los espacios métricos, $S \subseteq \mathbb{U}$ es un conjunto finito de datos que pueden preprocesarse, donde la función $d()$ mide la disimilitud.

El problema del vecino más cercano, que encuentra el punto más cercano en un conjunto de datos para un punto de consulta específico, es importante en muchas áreas de Ciencia de la Computación. Definimos cuatro problemas de búsqueda de vecinos más cercanos, que usaremos para probar nuestra implementación. Dado un objeto de consulta $q \in \mathbb{U}$, para recuperar objetos similares a q .

Approximate Nearest Neighbor Search $ANN(q, \varepsilon)$: encuentra los vecinos más cercanos aproximados de q . La respuesta es aproximadamente correcta con cierta probabilidad, es decir, la probabilidad de éxito $\delta = 1 - \varepsilon$. Siempre hay una pequeña posibilidad de que otro objeto (la verdadera solución) esté más cerca.

El $(1 + \varepsilon) - kNNq$ busca los elementos k más similares a q . Supongamos que $S' = \{s'_1, s'_2, \dots, s'_k\}$ es el conjunto de resultados de la consulta, luego $\forall s'_i \in S', d(q, s'_i) \leq (1 + \varepsilon) \times \xi$.

La figura 5.1.1 ilustra el modelo de búsqueda de similitud para la búsqueda ANN utilizando hashing. Cada objeto del conjunto de datos se comprime utilizando códigos hash que generan el nuevo espacio de búsqueda en el espacio reducido. Por lo general, se usa un índice para organizar el conjunto de datos de manera que los objetos cercanos en el espacio original estén juntos bajo cierta probabilidad en el espacio reducido. En el momento de la consulta, el objeto de consulta se asigna a regiones con alta probabilidad de encontrar objetos similares. Finalmente, las regiones de calificación se analizan para informar solo los objetos que satisfacen la condición de consulta.

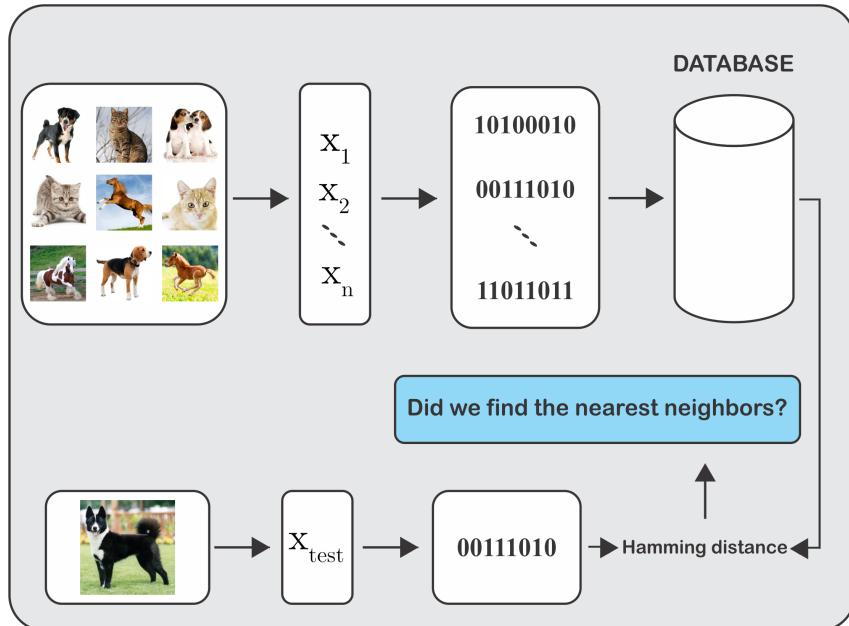


Figura 5.1: Hashing para la búsqueda del vecino más cercano. Comprimir un conjunto de vectores $(x_i)_{i=1}^n, x_i \in \mathbf{R}^d$

Supervised Hashing

La figura 5.1.1 ilustra un modelo de búsqueda de similitud para Hashing Supervisado. Cada objeto del conjunto de datos se comprime utilizando códigos hash que generan el nuevo espacio de búsqueda en el espacio reducido. Cuando se supervisa el hash, los códigos se entregan usando etiquetas en los datos de entrenamiento.

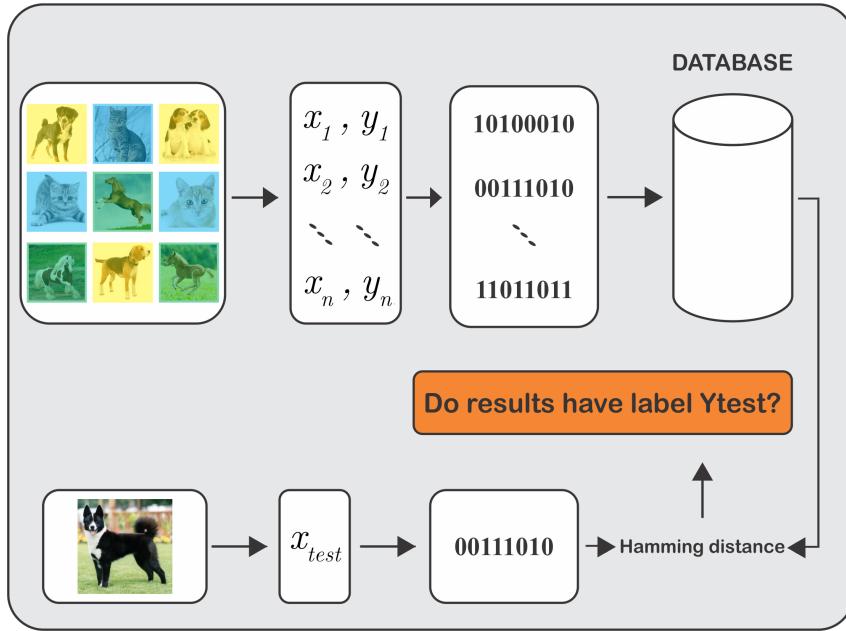


Figura 5.2: Supervised Hashing: comprime un conjunto de vectores y sus etiquetas $((x_i, y_i))_{i=1}^n, x_i \in \mathbb{R}^d, y_i \in \{1, \dots, L\}$.

- Supervised Hashing(SH) [1, 2]: etiquetas y_i conocido por todos $x_i (i = 1, \dots, n)$ en el conjunto de referencia
- Semi-supervised hashing (SSH) [3, 4]: etiquetas y conocido solo por n_{label} muestras

5.1.2. Protocolos de evaluación de recuperación

En esta sección, describimos los protocolos utilizados en la literatura para SSH y SH y explicamos cómo una estrategia simple resuelve eficientemente los problemas similares.

Protocolos de evaluación de SSH y SH

SSH consiste en indexar un conjunto de datos de n imágenes \mathcal{I}_{train} , de los cuales un subconjunto $\mathcal{I}_{label} \subseteq \mathcal{I}_{train}$ es etiquetado. Pero, SH es el caso extremo $\mathcal{I}_{label} = \mathcal{I}_{train}$. Si tenemos una imagen de consulta sin etiqueta q , el sistema debe devolver una lista ordenada de imágenes del \mathcal{I}_{train} . Para evaluar, tenemos un conjunto de datos

de consultas; el evaluador conoce todas las etiquetas de las consultas, así como las etiquetas en \mathcal{I}_{train} , también en la configuración SSH, y luego una imagen se considera correcta si tiene una misma etiqueta. El rendimiento se mide por precisión o *mean average precision* (mAP).

Entonces, dada una consulta q , si la imagen i^{th} es correcta para q definimos $\delta(q, i) = 1$, y 0 de lo contrario. La precisión en el rango k viene dada por $P(q, k) = \frac{1}{k} \sum_{i=1}^k \delta(q, i)$. Denotando por $cl(q) = \sum_{i=1}^n \delta(q, i)$ el número total de imágenes correctas en \mathcal{I}_{train} , y la precisión promedio en k es $AP(q, k) = \frac{1}{cl(q)} \sum_{i=1}^k \delta(q, i) P(q, i)$. El mAP en k (o simplemente mAP cuando $k = n$) es el AP promedio sobre todas las consultas de prueba [Sablayrolles et al., 2016a].

Representación de datos y medidas de similitud

Se han propuesto muchos métodos en la bibliografía para representar datos complejos con dimensionalidades reducidas, que admiten búsquedas de similitud y tareas de minería de datos. Estos métodos se pueden representar en dos categorías: representaciones de datos adaptables y representaciones de datos no adaptativas [Xiaoyue Wang and Keogh, 2013].

En la representación de *Adaptive Data* tenemos los siguientes métodos: *Singular Value Decomposition* (SVD) que transforma la serie de tiempo A_i en otra serie de tiempo B_i de longitud n cuyos elementos están en orden decreciente [Bettaiah and Ranganath, 2014]. *Principal Component Analysis* (PCA) que involucra procedimientos matemáticos que transforman un número de (posibles) variables de correlación en un número (más pequeño) de variables no correlacionadas, llamadas componentes principales [Elena E. and Matteucci, 2001]. En PCA, los datos se resumen como una combinación lineal de conjuntos de vectores ortonormales. La PCA actúa de forma similar a los Autoencoders (AE) que aplican la retropropagación estableciendo los valores objetivo para que sean iguales a las entradas $y(i) = x(i)$ [Elena E. and Matteucci, 2001].

Los métodos más comunes utilizados en la representación de datos no adaptativos son: *Discrete Cosine Transformation* (DCT), es una transformación ortonormal real. DCT es similar a *discrete Fourier transform* (DFT) [Faloutsos et al., 1994b], con la diferencia de que usa funciones de coseno en lugar de senos y cosenos para transformar del dominio de tiempo al dominio de frecuencia [Bettaiah and Ranganath, 2014]. *Discrete Wavelet Transformation* (DWT) procesa datos a diferentes escalas o resoluciones en contraste con DFT [Faloutsos et al., 1994b] donde solo los componentes de frecuencia se consideran [Bettaiah and Ranganath, 2014].

Junto con esto, hay más de una docena de medidas de distancia para la búsqueda de similitudes en la literatura. Las medidas de similitud más comunes utilizadas son Euclidean, Manhattan y Minkowski. La distancia euclídea y la de Manhattan satisfacen algunas propiedades matemáticas: no negatividad, identidad

de indiscernibles, simetría, desigualdad de triángulo [Han et al., 2011].

5.1.3. Dimensión fractal y reducción de dimensionalidad

Como se mostró [cit, 2007] que un algoritmo de reducción de dimensionalidad exitoso proyecta los datos en un espacio de características con dimensionalidad cercana a la dimensionalidad fractal (FD) de los datos en el espacio original y conserva las propiedades topológicas.

Por lo tanto, para encontrar la dimensionalidad objetivo (m) necesaria, podemos seguir la siguiente heurística. Podemos comenzar con el valor en $m_1 = 2^2$, calcular el FD del nuevo espacio con solo eso, luego incrementar el valor en $m_2 = 2^3$, recalcular el FD, y continuar haciendo esto hasta que $t(m_t = 2^t)$ donde podemos ver un aplanamiento en la dimensión fractal, lo que significa que más características no cambian la dimensionalidad fractal del conjunto de datos.

Teoría Fractal

Un fractal se caracteriza por la propiedad de auto-similitud, es decir, es un objeto que presenta aproximadamente las mismas características cuando se analiza en una amplia gama de escalas [Jr. et al., 2010b]. De la Teoría Fractal, *Correlation Fractal Dimension* \mathfrak{D} es particularmente útil para el análisis de datos, ya que se puede aplicar para estimar la dimensión intrínseca de conjuntos de datos reales que exhiben comportamiento fractal, es decir, conjuntos de datos idénticos o estadísticamente idénticos [Bones et al., 2016]. Se ha demostrado que, dado un conjunto de N objetos en un conjunto de datos con una función de distancia $d(x, y)$, el número promedio de k vecinos dentro de una distancia dada r es proporcional a r elevado a \mathfrak{D} . Por lo tanto, el *pair-count* $PC(r)$ de pares de elementos dentro de la distancia r sigue la ley:

$$PC(r) = K_p \times r^{\mathfrak{D}} \quad (5.1)$$

donde, K_p es una constante de proporcionalidad, y \mathfrak{D} es la dimensión fractal de correlación del conjunto de datos. En consecuencia, un fractal se define por la propiedad de auto-similitud, que es la característica principal que representa exacta o estadísticamente la similitud entre las partes de todo el fractal.

5.1.4. Comparación de representaciones de datos

Dimensión fractal y reducción de dimensionalidad

Para obtener la representación más general para el aprendizaje, algunos autores sugieren utilizar la salida de la última capa convolucional de la CNN [Sablayrolles et al., 2016b]. Entonces, en esta investigación, lo usamos como nuestro método de extracción de características. Sin embargo, el espacio reducido donde viven estos datos sigue siendo de gran dimensión. El objetivo de esta sección es encontrar el

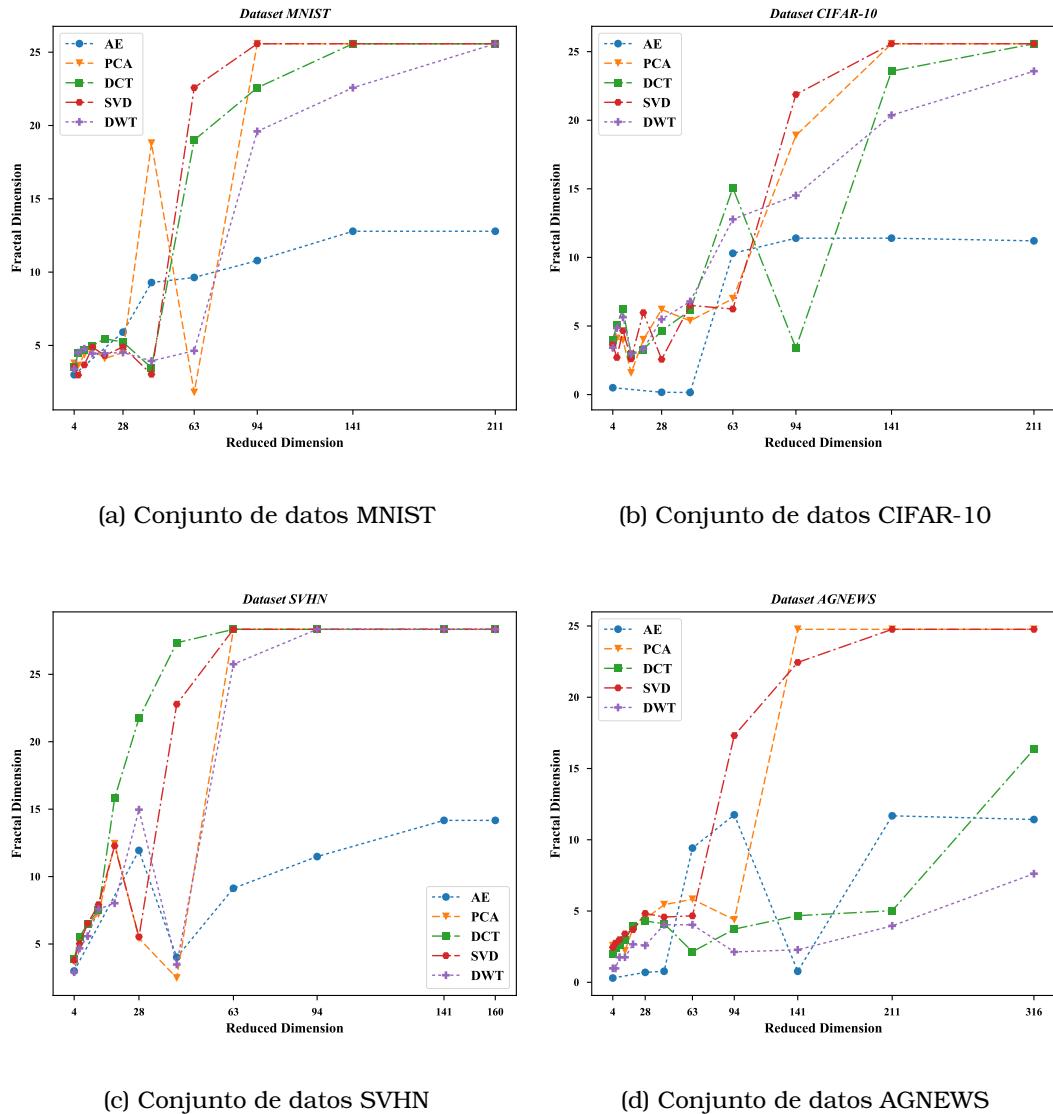


Figura 5.3: Dimensiones Fractal para diferentes métodos de reducción de dimensionalidad .

Cuadro 5.1: Precisión de la clasificación usando KNN-clasificador en conjuntos de datos MNIST, CIFAR-10, SVHN y Agnews utilizando AutoEncoders y PCA como métodos de reducción de dimensionalidad.

	ORIGINAL		AE		PCA	
	[HTML]E6E6E6Dim.	Acc.	[HTML]E6E6E6Dim.	Acc.	[HTML]E6E6E6Dim.	Acc.
MNIST	[HTML]E6E6E6		[HTML]E6E6E663	0.980	[HTML]E6E6E642	0.982
	[HTML]E6E6E6300	0.982	[HTML]E6E6E141	0.982	[HTML]E6E6E694	0.975
CIFAR10	[HTML]E6E6E6		[HTML]E6E6E642	0.902	[HTML]E6E6E663	0.890
	[HTML]E6E6E696	0.899	[HTML]E6E6E694	0.894	[HTML]E6E6E6141	0.885
SVHN	[HTML]E6E6E6		[HTML]E6E6E663	0.872	[HTML]E6E6E628	0.880
	[HTML]E6E6E652	0.881	[HTML]E6E6E141	0.880	[HTML]E6E6E663	0.890
AGNEWS	[HTML]E6E6E6		[HTML]E6E6E694	0.874	[HTML]E6E6E663	0.840
	[HTML]E6E6E604	0.867	[HTML]E6E6E6211	0.875	[HTML]E6E6E6141	0.820
			[HTML]E6E6E6316	0.863	[HTML]E6E6E6316	0.760

mejor método de reducción de dimensionalidad y el espacio dimensional más pequeño donde puedan existir los datos sin perder información.

Hemos implementado los métodos de reducción descritos en la sección 5.1.2, y los aplicamos en los vectores característicos de los conjuntos de datos MNIST [LeCun and Cortes, 2010], CIFAR-10 [Krizhevsky et al.,], SVHN [Netzer et al., 2011] y AgNews [Del Corso et al., 2005]. Para nuestros experimentos, estos vectores de características serán los datos de entrada, y utilizaremos la dimensión fractal para medir el grado de representación de cada nueva dimensión.

Podemos ver la dimensión fractal de cada espacio reducido para cada conjunto de datos en las figuras 5.3. Con la ayuda de los gráficos, visualizamos que las curvas generadas se estabilizan a medida que aumenta la dimensión reducida, de acuerdo con los experimentos realizados podemos proponer la siguiente hipótesis: "La aproximación de la dimensión reducida óptima es el valor inicial donde la curva del dimensión fractal vs. la dimensión reducida se estabiliza ". Entonces, cualquier reducción de dimensión después del punto de estabilidad, no reducirá la pérdida de información. Para demostrar esto, evaluamos la precisión de clasificación de las dimensiones reducidas, generadas por los métodos PCA y Autoencoders. El método de clasificación que vamos a utilizar será la clasificación KNN. Los resultados se muestran en la tabla 5.1.

Ahora vamos a hacer un análisis de precisión de clasificación. La idea es trabajar con *nearest neighbor classifier* (1NN) en cada conjunto de datos para evaluar la eficacia de la medida de distancia.

Entonces, comparamos las diferentes varianzas de L_p -norms [Sutherland, 1975]. En la tabla 5.2, podemos ver la precisión para tres dimensiones reducidas mediante Autoencoders de cada conjunto de datos. Encontramos que las distancias *Euclidean*

[Gower, 1982] y Minkowski tienen un rendimiento similar, y en ambos casos ambas superan en un mínimo a la distancia de Manhattan.

Cuadro 5.2: Análisis de precisión de clasificación usando autoencoders para diferentes métricas.

	Dim.	L1	L2	L-inf
MNIST	63	0,9803	0,9806	0,9806
	141	0,9808	0,9823	0,9823
	316	0,9826	0,9821	0,9821
CIFAR-10	42	0,9007	0,9023	0,9023
	94	0,8965	0,8942	0,8942
	211	0,9009	0,8982	0,8982
SVHN	63	0,8705	0,8729	0,8729
	141	0,8798	0,8808	0,8808
	211	0,8781	0,8795	0,8795
AGNEWS	94	0,8714	0,8748	0,8748
	211	0,8750	0,8752	0,8752
	316	0,8674	0,8634	0,8634

5.1.5. Comparación del rendimiento de recuperación

En esta sección, describimos las dos tareas de evaluación propuestas en [Sablayrolles et al., 2016a], es decir, la recuperación de *unseen classes*. Corresponden a casos de aplicación en 4 grandes conjuntos de datos.

Definición del dataset. en el momento de la prueba, utilizamos clases separadas de un conjunto de datos de clasificación estándar. Al aprender la función de hashing, se supone que se conocen 75 % de las clases y se usan las clases restantes de 25 % para evaluar el esquema de codificación/hash. Llamamos a train75/test75 las imágenes de train/test de 75 % de las clases y train25/test25 las restantes.

Protocolo 1: Retrieval of unseen classes, para indexar train25 y usar test25 como consultas, utilizamos el esquema hash. Usamos las etiquetas de train25 solo para evaluación. Esta configuración es como un enfoque de búsqueda de instancias, excepto que las etiquetas de clase dan la verdad. La división train75 - train25 - test25 es el equivalente supervisado de la división de búsqueda - base de datos - consulta en hash no supervisado.

El rendimiento se midió en métodos de búsqueda aproximados bien conocidos, E2LSH [Lv et al., 2007], SSDH [Kelvin Lin, 2015], y también la búsqueda por fuerza bruta. Todos los experimentos se realizaron en un *workstation* con CPU Intel core i7 3.0Ghz (12 núcleos) y 64 Gb de RAM con cuatro GPU Geforce GTX 1080 con VRAM de 8 Gb cada una.

Realizamos experimentos sobre el modelo propuesto en esta sección (Protocolo 1), utilizando el algoritmo de recuperación de información KNN con k = 100, alimentamos este algoritmo con los espacios reducidos de los conjuntos de datos (MNIST, CIFAR-

10, SHVN y AGNews). En la tabla 5.3, podemos ver que el *mean average precision* (mAP) permanece estable a medida que aumenta la dimensión de los datos. Además, el mAP de las dimensiones reducidas está cerca del mAP de los datos de dimensión originales. Por lo tanto, usar una dimensión más grande no mejorará los resultados de la recuperación de información.

Cuadro 5.3: Mean Average Precision(mAP) y el tiempo acumulado para calcular el mAP para diferentes métodos en los conjuntos de datos MNIST, CIFAR-10, SVHN Y AGNEWS. El número k de los vecinos devueltos se establece en 100.

[HTML]EFEFEG	[HTML]EFEFEG		[HTML]EFEFEGFORIGINAL	Dim.	mAP	Time
MNIST				800	0.90	6.53
[HTML]EFEFEG	[HTML]EFEFEG		[HTML]EFEFEG	[HTML]EFEFEG	[HTML]EFEFEG	[HTML]EFEFEG
[HTML]EFEFEG	[HTML]EFEFEG	CIFAR10	[HTML]EFEFEG4096	[HTML]EFEFEG0.782	[HTML]EFEFEG	[HTML]EFEFEG
SVHN				1152	0.816	35.61
[HTML]EFEFEG	[HTML]EFEFEG		[HTML]EFEFEG	[HTML]EFEFEG	[HTML]EFEFEG	[HTML]EFEFEG
[HTML]EFEFEG	[HTML]EFEFEG	AGNEWS	[HTML]EFEFEG8704	[HTML]EFEFEG0.775	[HTML]EFEFEG	[HTML]EFEFEG

Finalmente, utilizamos Autoencoders con cinco capas, para generar dimensiones reducidas de los conjuntos de datos (MNIST, CIFAR-10, SHVN y AGNews). Las dimensiones (141,141,141 y 211), respectivamente, son las aproximaciones del mejor espacio reducido. Además, comparamos el mejor espacio reducido frente a la dimensión original y una dimensión más pequeña. Nuestro resultado en la Tabla 5.4 muestra que los espacios mejor reducidos obtienen una precisión cercana a la dimensión original.

5.2. Conclusions

Presentamos experimentos exhaustivos que comparan métodos recientes de *deep hashing* que permitieron obtener mejores representaciones del espacio de datos con un menor costo computacional para obtener una mejor precisión al calcular los mejores parámetros de datos. Hemos explorado las correlaciones entre CNN usando la teoría fractal. Para optimizar estos parámetros, utilizamos diferentes métodos de reducción de dimensionalidad que proyectan el conjunto de datos en un espacio de características con dimensionalidad cercana a la dimensionalidad fractal (FD) de los datos en el espacio original. Presentamos una descripción general de estas diferentes técnicas y presentamos nuestro experimento comparativo para representación de datos y rendimiento de recuperación. Los resultados empíricos

Cuadro 5.4: Precisión 1000-NN y tiempo acumulado para diferentes métodos en los conjuntos de datos MNIST, CIFAR-10, SVHN y AGNEWS. El número k de los vecinos devueltos se establece en 1000.

			1000-NN	Time
MNIST	[HTML]EFEF	SSDH(D=48)	0.514	3.802
	[HTML]EFEF	E2LSH(D=800)	0.873	9.531
	[HTML]EFEF	E2LSH(D=141)	0.688	4.152
[HTML]EFEF	[HTML]EFEF	SSDH(D=128)	0.437	3.972
	[HTML]EFEF	E2LSH(D=4096)	0.759	6.85
	[HTML]EFEF	E2LSH(D=141)	0.493	4.223
SVHN	[HTML]EFEF	SSDHI(D=63)	0.332	11.45
	[HTML]EFEF	E2LSH(D=1152)	0.780	19.723
	[HTML]EFEF	E2LSH(D=141)	0.634	12.56
[HTML]EFEF	[HTML]EFEF	SSDH(D=128)	0.384	3.217
	[HTML]EFEF	E2LSH(D=8704)	0.790	13.763
	[HTML]EFEF	E2LSH(D=211)	0.590	3.321

muestran las capacidades de la teoría Fractal para encontrar el subespacio óptimo del conjunto de datos, podemos encontrar una configuración óptima para el proceso de aprendizaje e indexación. Además, pudimos sintonizar métodos de *deep hashing*, basados en la teoría fractal, que nos permite encontrar los valores de parámetros óptimos. Además, podemos estimar estos parámetros en tiempo lineal debido a que depende del cálculo de la dimensión fractal.

Deep Fractal based Hashing - DAsH

6.1. Consideraciones Iniciales

La creciente disponibilidad de datos en diversos dominios ha creado la necesidad de desarrollar técnicas y métodos para descubrir el conocimiento a partir de volúmenes masivos de datos complejos, motivando a muchos investigadores a trabajar en *databases*, *machine learning*, y comunidades de recuperación de información. Esto ha impulsado el desarrollo de técnicas escalables y eficientes para organizar y recuperar este tipo de datos. Las búsquedas por similitud han sido el enfoque tradicional para la recuperación de la información. Considerando que la similitud es el criterio intuitivo por el cual las personas hacen comparaciones, las comunidades de recuperación de información usan similitud para organizar y buscar datos. Sin embargo, la similitud es una medida intuitiva para la comparación, causa algunas dificultades en representar datos complejos y con la estabilidad de los algoritmos cuando la dimensionalidad de los datos es muy alta (la “maldición de la dimensionalidad”). Aunque muchos trabajos de investigación se han llevado a cabo en el desarrollo de estructuras eficientes de índices y algoritmos de búsqueda por similitud, sólo unos pocos presentan una garantía teórica de estabilidad asintótica para datos de alta dimensión.

Uno de los pocos enfoques que aseguran una solución aproximada con el coste de búsqueda sublineal para datos de alta dimensión es *Locality Sensitive Hashing* (LSH) [Datar et al., 2004a]. LSH se basa en la idea de que la cercanía entre dos objetos suele ser preservada por una operación de proyección aleatoria. En otras palabras, si dos objetos están cerca juntos en su espacio original, entonces estos dos objetos permanecerán cerca después de una operación de proyección escalar. Sin embargo, presenta algunas dificultades para consultas kNN aproximadas, en

particular, relacionadas con la dependencia de los parámetros del dominio de los datos y resultados de calidad. Por lo tanto, en dominios complejos, en particular, en problemas con datos con gran dimensión, una solución aproximada con un sólido análisis teórico puede ser la mejor opción en muchas áreas de aplicación debido a su eficiencia en tiempo y espacio.

Por otra parte, en *Machine Learning* las imágenes se describen a menudo mediante las características visuales manuales. Sin embargo, estas características manuales no pueden revelar el significado semántico de alto nivel (etiquetas o tags) de las imágenes, y a menudo limitan el rendimiento de la recuperación de imágenes [Li et al., 2015]. Así, para obtener esta información semántica tenemos que trabajar con la información de la etiqueta y procesar los datos en un modo supervisado. Inspirado por recientes avances en Red Neuronal Convolutiva (CNN) para problemas de clasificación de imágenes, detección de objetos, y muchas otras tareas de visión [Krizhevsky et al., 2012, Szegedy et al., 2013, Liu et al., 2012], muchos métodos resolvieron el problema de la precisión de la recuperación de similitud utilizando CNN como extractor de características y luego construir un código hash compacto de preservación de similitud para la recuperación rápida de imágenes. De nuevo, *hashing* es ampliamente usado para recuperar imágenes a gran escala así como las búsquedas de video y documentos porque la representación compacta del código hash es esencial para el almacenamiento de datos y es razonable para las búsquedas de consultas [Shen et al.,]. Sin embargo, algunos inconvenientes basados en estos métodos de *hashing* supervisados no se han resuelto completamente, como sigue:

- Existe una compensación entre error de clasificación y error de cuantificación: activaciones de capas inferiores son más generales [Yosinski et al., 2014], así que el entrenamiento es más eficaz. Sin embargo, las capas inferiores tienen mapas de activaciones más grandes (muchos nodos), las cuales son más difíciles de codificar, lo que conduce a un compromiso.
- Existe una dependencia de los valores de los parámetros para esquemas aproximados de búsqueda de similitud basados en LSH, que determinan el número de funciones hash y el número de tablas hash.

En este tesis se propone una nueva técnica de *hashing* supervisada, llamada *Deep fractal based Hashing* (DAsH), diseñado para realizar una búsqueda de similitud aproximada escalable. Las contribuciones de nuestro trabajo son las siguientes. Primero, introducimos y definimos un esquema jerárquico basado en CNN y optimizado usando la teoría fractal. Para superar la limitación de grandes activaciones en capas inferiores de CNN (salida de la última capa convolucional) reducimos su dimensionalidad usando autocodificadores al sub-espacio óptimo. Luego indexamos esta nueva representación con esquema LSH. Segundo, presentamos un nuevo

método, basado en teoría fractal, que nos permite encontrar el número óptimo de funciones hash para un esquema aproximado de búsqueda de similitud basado en LSH.

6.1.1. Teoria del Fractal

Un fractal se caracteriza por la propiedad de auto-similitud, es decir, Es un objeto que presenta aproximadamente las mismas características cuando se analiza en una amplia gama de escalas [Jr. et al., 2010b, Bones et al., 2016]. De la teoría del fractal, la dimensión del fractal de la correlación \mathfrak{D} es particularmente útil para el análisis de datos, ya que puede aplicarse para estimar la dimensión intrínseca de conjuntos de datos reales que muestran un comportamiento fractal, es decir, exactamente o estadísticamente auto-similar [Belussi and Faloutsos, 1995]. Se ha demostrado que, dado un conjunto de N objetos en un conjunto de datos con una función de distancia $d(x, y)$, el número medio de k vecinos dentro de una distancia dada r es proporcional a r elevado a \mathfrak{D} [Arantes et al., 2003]. Así, la cuenta de pares $PC(r)$ de pares de elementos a distancia r sigue la siguiente ley:

$$PC(r) = K_p \times r^{\mathfrak{D}} \quad (6.1)$$

donde, K_p es una constante proporcional, y \mathfrak{D} es la correlación dimensión fractal del conjunto de datos. En consecuencia, un fractal es definido por la propiedad de auto-similitud, que es la característica principal que representa exactamente o estadísticamente la similitud entre las partes de todo el fractal.

6.2. Deep Fractal based Hashing - DAsH

En esta sección, Nosotros proponemos *Deep Fractal based Hashing* (DAsH) diseñado para realizar una búsqueda aproximada escalable mediante un esquema de hash supervisado. Como se presentó en la sección 1, nuestra estrategia es usar la teoría de fractales para encontrar los óptimos sub-espacios para la última capa convolucional de salida de la red CNN, y el numero óptimo de funciones hash para una buena indexación LSH.

La figura 6.1 ilustra la estructura del proceso de entrenamiento. La red consiste de tres tipos de capas: 1) Capas convolucionales con pesos pre-entrenados sobre Imagenet(*transfer-learning*) y *fine-tuning* en el conjunto de datos objetivo; 2) Una capa completamente conectada (*full connected*) con la ultima capa *softmax*; 3) Capa de *autoencoders* los cuales son usados para reducir la dimensionalidad. Red Neuronal Convolutiva (CNN) está entrenado de extremo a extremo con etiquetas verdaderas. Utilizamos la salida de la última capa convolucional porque tiene la representación más general para el aprendizaje, pero tiene un inconveniente con una alta representación dimensional. Para superar el problema de alta dimensionalidad,

reducimos al sub-espacio óptimo usando un autoencoder. Luego indexamos el subespacio óptimo obtenido por el autoencodificador con esquema LSH que, como hemos mencionado, también se sintoniza gracias a la teoría fractal. Al mismo tiempo, usamos otros n-autoencoders para aprender la representación de cada clase. Después, usaremos estos autocodificadores para mejorar el proceso de recuperación.

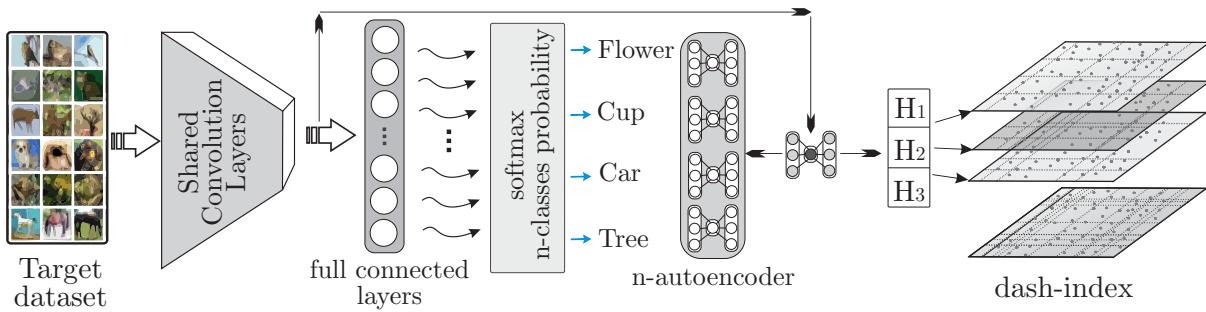


Figura 6.1: DAsH. Proceso de entrenamiento e indexación.

Como se mostró en [cit, 2007] un algoritmo de reducción de dimensionalidad exitoso proyecta los datos en un espacio de características con dimensionalidad cercana a la dimensionalidad fractal (FD) de los datos en el espacio original y conserva las propiedades topológicas. Por lo tanto, para encontrar la dimensionalidad objetivo (m) que necesitan las redes de autoencodificadores seguimos la siguiente heurística. Empezamos con el valor $m_1 = 2^2$, calculamos la FD del nuevo espacio con sólo eso, luego incrementamos el valor en $m_2 = 2^3$, recalculamos la FD, y continuamos haciendo esto hasta algún t ($m_t = 2^t$) donde podemos ver una mejora en la dimensión fractal, lo que significa que más características no cambian la dimensionalidad fractal del conjunto de datos.

El segundo paso de nuestro procedimiento es la recuperación de la imagen vía DAsH. Procesamos la imagen de consulta enviándola a través de CNN con el objetivo de obtener las clases n más fuertes. En contraste con los algoritmos de aprendizaje de similitud existentes que aprenden la similitud de la característica de bajo nivel, nuestra similitud es la combinación de similitud semántica y nivel de hashing. Así que, la similitud de nivel semántico se calcula en primer lugar. Después de la comprobación de relevancia semántica, obtendremos las nuevas consultas (q_1, q_2, \dots, q_n) usando los n autoencoders más fuertes. La consulta es transformada en una nueva consulta de objetos (q_1, q_2, \dots, q_n) los cuales son escogidos para localizar los apropiados *buckets*. Una vez que los *buckets* son localizados, el conjunto de candidatos relevantes son formados. Luego, los elementos del conjunto candidato se analizan exhaustivamente con el fin de recuperar sólo los objetos que satisfacen la condición de consulta. Este proceso es realizado para cada una de las L tablas hash . Este proceso es ilustrado en la Figura 6.2.

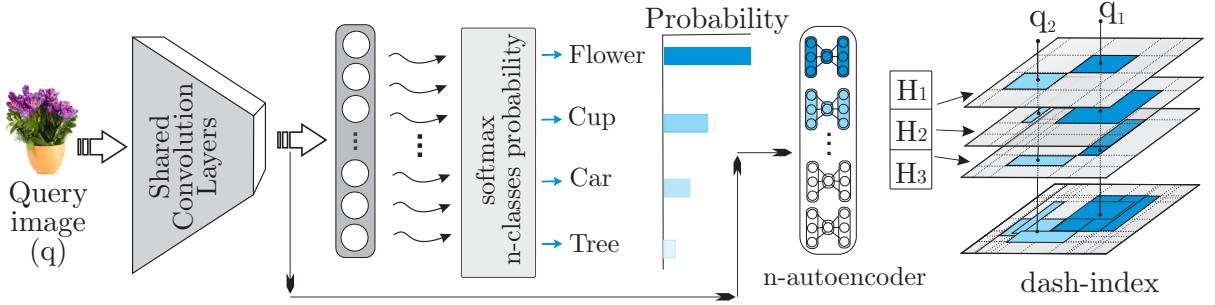


Figura 6.2: DAsH. Proceso de recuperación.

6.2.1. Usando Fractales para estimar los parámetros LSH

Para afinar los parámetros LSH se utilizó una propiedad de la correlación de la dimensión fractal \mathfrak{D} , que puede describir estadísticamente un conjunto de datos. Además, la correlación de la dimensión fractal \mathfrak{D} se puede estimar en tiempo lineal como se representa en [Traina Jr et al., 2010].

Estamos interesados en averiguar la escala de resolución $\log(r)$ en la que hay aproximadamente k objetos. Considerando la línea con pendiente \mathfrak{D} pasando en un punto definido como $\langle \log(r), \log(Pares(k)) \rangle$ la constante K_d usando la Ecuación 6.1 :

$$\begin{aligned} \log(PC(r)) &= \mathfrak{D} \times \log(r) + K_p \\ K_p &= \log(Pairs(k)) - \mathfrak{D} \times \log(r) \end{aligned} \quad (6.2)$$

Considerando otro punto $\langle \log(R), \log(Pares(N)) \rangle$, la constante K_d se define como:

$$K_d = \log(Pairs(N)) - \mathfrak{D} \cdot \log(R) \quad (6.3)$$

Ahora, combinando Ecuaciones 6.2, 6.3, podemos definir el radio r como:

$$r = R \cdot \exp\left(\frac{\log(Pairs(k)) - \log(Pairs(N))}{\mathfrak{D}}\right) \quad (6.4)$$

Usando la última ecuación 6.4 descubrimos que el número óptimo de funciones hash m para un índice basado en LSH configurado para recuperar los k vecinos más cercanos es proporcional al número de pares en una distancia r . Esto tiene sentido, porque un promedio de k vecinos están dentro de una distancia dada r . Entonces, definimos:

$$m \approx \log(PC(r)) \quad (6.5)$$

combinando las ecuaciones 6.5 y 6.1 obtendremos que $m \approx \mathfrak{D} \cdot \log(r)$. Experimentalmente, confirmamos que el valor óptimo de m es:

$$m = (\lceil \mathfrak{D} + 1 \rceil) \cdot \log(r) \quad (6.6)$$

6.3. Experimentos

En esta sección, estamos interesados en responder a la siguiente pregunta: (a) ¿Qué tan preciso es nuestro modelo en la estimación de los parámetros LSH utilizando la dimensión fractal?; (B) ¿Cómo mejora nuestro método DAsH las otras implementaciones de LSH en términos de *consultas de rendimiento y precisión*? El rendimiento del método DAsH se comparó con los de los dos métodos más conocidos, llamado Multi-probe LSH [Lv et al., 2007] y LSH-Forest [Bawa et al., 2005]. Todos los experimentos se realizaron en un workstation con Intel core i7 3.0Ghz (12 núcleos) CPU y 64Gb de RAM con cuatro tarjetas gráficas Geforce GTX 1080 GPU de 8Gb VRAM cada una.

Primero realizamos experimentos en ocho conjuntos de datos ampliamente utilizados usando características hechas a mano (AUDIO, CITIES, EIGENFACES, HISTOGRAMS, MGCOUNTY, RANDOMWALK, SYNTH16D, SYNTH6D, VIDEO)¹ para evaluar nuestro método propuesto para estimar los parámetros LSH. Además de las características hechas a mano, también mostramos la eficacia de nuestros métodos cuando las características son extraídas por las Redes Neuronales Convolucionales profundas (CNN), realizamos este experimento en tres conjuntos de datos (MNIST², CIFAR-10³, SVHN⁴) para evaluarlos en términos de **consultas de rendimiento** y **mean average precision (mAP)**. A continuación se describen los detalles de los experimentos y resultados.

6.3.1. Experimento 1: Sintonización de parámetros LSH

Los métodos basados en LSH reportan resultados eficientes cuando los valores m (número de funciones hash) son elegidos. Para evaluar la eficacia del enfoque presentado para afinar los parámetros LSH utilizando la dimensión fractal, hemos trabajado en una variedad de datos sintéticos y reales. Tabla 6.1 resume las principales características y parámetros de los conjuntos de datos, incluyendo el número de elementos N , el número de atributos d , su dimensión intrínseca (fractal) D y los parámetros LSH usando dos aproximaciones: el algoritmo de Andoni⁵ y nuestra propuesta basada en dimension fractal. Los resultados del experimento para el número de funciones de hash m muestran que las estimaciones dadas por la ecuación 6.6 son comparables con las obtenidas con el algoritmo E2LSH propuesto por Andoni utilizando hasta 10 veces menos tiempo.

¹https://github.com/joselhuillca/fractal_dataset

²<http://yann.lecun.com/exdb/mnist/>

³<https://www.cs.toronto.edu/~kriz/cifar.html>

⁴<http://ufldl.stanford.edu/housenumbers/>

⁵<http://www.mit.edu/~andoni/LSH/>

Cuadro 6.1: Optimal LSH Params using exhaustive e2lsh and the fractal based method.

	dataset		fractal params			e2lsh		r	HTML
	N	d	D	log(R)	log(CR)	m	time(s)		
audio	54387	192	6.49	-1.30	17.00	[HTML]f2f2f218	[HTML]f2f2f264.20	0.14	[HTM
cities	5507	2	2.36	2.30	16.00	[HTML]f2f2f28	[HTML]f2f2f216.45	4.16	[HTM
eigenfaces	11900	16	4.25	-1.40	18.00	[HTML]f2f2f212	[HTML]f2f2f242.28	0.13	[HTM
histograms	4247	256	2.50	-0.81	16.01	[HTML]f2f2f26	[HTML]f2f2f215.62	0.22	[HTM
mgcounty	27282	2	1.81	0.70	19.00	[HTML]f2f2f24	[HTML]f2f2f287.96	0.27	[HTM
randomwalk	10000	1024	5.52	2.80	15.00	[HTML]f2f2f216	[HTML]f2f2f250.60	10.16	[HTM
synth16d	10000	16	8.36	-1.40	17.00	[HTML]f2f2f220	[HTML]f2f2f227.20	0.18	[HTM
synth6d	10000	6	4.95	-1.40	17.00	[HTML]f2f2f212	[HTML]f2f2f228.16	0.14	[HTM
video	79094	50	7.73	-1.40	21.00	[HTML]f2f2f216	[HTML]f2f2f21205.73	0.13	[HTM

6.3.2. Rendimiento de Recuperación

El objetivo de este experimento es medir el tiempo total dedicado a recuperar los objetos vecinos k -nearest. Las estructuras de datos que se compararon se probaron con valores específicos para las consultas. Por lo tanto, usamos $k = 1000$ cuando calculamos la métrica de Mean Average Precision (mAP) y $k = 25$ cuando calculamos la métrica de precisión ($P(%)$).

Cuadro 6.2: Mean Average Precision(mAP), precision, and cumulative time spent to compute mAP for different methods on the MNIST, SVHN and CIFAR-10 datasets.

	MNIST			HTML
	mAP	P (%)	time(s)	
MpLSH [Lv et al., 2007]	0.86	0.95	[HTML]c8c8c819.58	
ITQ [Gong et al., 2013]	0.87	0.95	1297.32	
LOPQ [Kalantidis and Avrithis, 2014]	0.86	0.90	119.21	
LSH-F [Bawa et al., 2005]	0.85	0.96	217.33	
DAsH (Ours)	[HTML]c8c8c80.93	[HTML]c8c8c80.98	[HTML]f2f2f220.82	[HTML]

La Tabla 6.2 muestra la comparación en términos de mean average precision (mAP) y tiempo de consulta para todos los métodos basados en hash. El método DAsH fue significativamente mejor que otros métodos; proporcionando hasta un mAP 10% mejor, manteniendo un excelente tiempo de recuperación. Este resultado demuestra el potencial de impulsar las operaciones de consulta con el diseño de estructuras de índices especializados.

6.4. Conclusions

En este trabajo, presentamos un nuevo esquema para resolver la búsqueda aproximada de similitud por hashing supervisado llamado Deep Fractal base Hashing. Nuestro enfoque muestra el potencial de impulsar las operaciones de consulta cuando se diseña una estructura de índice especializada de extremo a extremo. Debido a las habilidades de la teoría de Fractal para encontrar el sub-espacio óptimo del conjunto de datos y el número óptimo de funciones hash para LSH Index, podemos encontrar una configuración óptima para el aprendizaje y el proceso de indexación. Además,

hemos definido un nuevo método, basado en la teoría fractal, que nos permite encontrar el número óptimo de funciones hash para el índice LSH. Podemos estimar estos parámetros en tiempo lineal debido a que depende del cálculo de la dimensión fractal .

Realizamos estudios de desempeño en muchos conjuntos de datos reales y sintéticos. Los resultados empíricos para los parámetros LSH muestran que nuestro método basado en la teoría fractal es comparable con los obtenidos con el algoritmo de fuerza bruta utilizando hasta $10X$ menos de tiempo. Por otra parte, en el rendimiento de recuperación, el método DAsH fue significativamente mejor que otros métodos aproximados, proporcionando hasta un 8 % mejor precisión, manteniendo excelentes tiempos de recuperación.

CAPÍTULO

7

Conclusiones y Trabajos Futuros

Bibliografía

[cit, 2007] (2007). Chapter 7 Fractal dimension. *Mathematics in Science and Engineering*, 209:167 – 218. L-System Fractals.

[Aggarwal, 2015] Aggarwal, C. C. (2015). *Data mining: the textbook*. Springer.

[Andoni and Indyk, 2008] Andoni, A. and Indyk, P. (2008). Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*, 51(1):117–122.

[Arantes et al., 2003] Arantes, A. S., Vieira, M. R., Traina, A. J. M., and Traina, C. (2003). The fractal dimension making similarity queries more efficient. In *Proceedings of the II ACM SIGKDD Workshop on Fractals, Power Laws and Other Next Generation Data Mining Tools*. ACM.

[Babenko, 2015] Babenko, A., L. V. (2015). Aggregating local deep features for image retrieval. *Proceedings of the IEEE International Conference on Computer Vision*, pages 1269–1277.

[Babenko, 2014] Babenko, A., S. A. C. A.-L. V. (2014). Neural codes for image retrieval. *Proceedings of European Conference on Computer Vision*, pages 584–599.

[Barbará and Chen, 2003] Barbará, D. and Chen, P. (2003). Using self-similarity to cluster large data sets. *Data Mining and Knowledge Discovery*, 7(2):123–152.

[Barbará et al., 2004] Barbará, D., Chen, P., and Nazeri, Z. (2004). *Self-Similar Mining of Time Association Rules*, pages 86–95. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Bawa et al., 2005] Bawa, M., Condie, T., and Ganesan, P. (2005). LSH forest: self-tuning indexes for similarity search. In *Proceedings of the 14th international conference on World Wide Web*, pages 651–660, Chiba, Japan.

[Beckmann et al., 1990] Beckmann, N., Kriegel, H.-P., Schneider, R., and Seeger, B. (1990). The R*-tree: an efficient and robust access method for points and rectangles. *SIGMOD Record*, 19(2):322–331.

[Bellman, 1961] Bellman, R. E. (1961). *Adaptive control processes - A guided tour*, volume 1961, chapter (A RAND Corporation Research Study), pages 255–260. Princeton University Press.

[Belussi and Faloutsos, 1995] Belussi, A. and Faloutsos, C. (1995). Estimating the selectivity of spatial queries using the ‘correlation’ fractal dimension. In *Proceedings of the 21th International Conference on Very Large Data Bases*, VLDB ’95, pages 299–310, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Belussi and Faloutsos, 1998] Belussi, A. and Faloutsos, C. (1998). Estimating the selectivity of spatial queries using thecorrelation’fractal dimension. Technical report.

[Bentley, 1979] Bentley, J. L. (1979). Multidimensional Binary Search Trees in Database Applications. *Transactions on Software Engineering*, 5(4):333–340.

[Berchtold et al., 1996] Berchtold, S., Keim, D. A., and Kriegel, H.-P. (1996). The X-tree: An Index Structure for High-Dimensional Data. In *Proceedings of the International Conference on Very Large Data Bases*, pages 28–39, San Francisco, CA, USA.

[Bettaiah and Ranganath, 2014] Bettaiah, V. and Ranganath, H. S. (2014). An analysis of time series representation methods: Data mining applications perspective. In *Proceedings of the 2014 ACM Southeast Regional Conference*, ACM SE ’14, pages 16:1–16:6, New York, NY, USA. ACM.

[Blott and Weber, 2008] Blott, S. and Weber, R. (2008). What’s wrong with high-dimensional similarity search. In *Proceedings of the International Conference on Very Large Data Bases*, pages 3–3, Aukland, New Zealand.

[Böhm et al., 2001] Böhm, C., Berchtold, S., and Keim, D. A. (2001). Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3):322–373.

[Bones et al., 2016] Bones, C. C., Romani, L. A. S., and de Sousa, E. P. M. (2016). Clustering multivariate data streams by correlating attributes using fractal dimension. *JIDM*, 7(3):249–264.

[Bottou, 2012] Bottou, L. (2012). *Stochastic Gradient Tricks*, volume 7700, page 430–445. Springer.

- [Boureau, 2010] Boureau, Y.L., P. J. L. Y. (2010). A theoretical analysis of feature pooling in visual recognition. *Proceedings of the 27th International Conference on Machine Learning*, pages 111–118.
- [Buduma, 2016] Buduma, N. (2016). *Fundamentals of Deep Learnin*. O'Reilly Media, Inc.
- [Chakrabarti and Faloutsos, 2002] Chakrabarti, D. and Faloutsos, C. (2002). F4: Large-scale automated forecasting using fractals. In *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, CIKM '02, pages 2–9, New York, NY, USA. ACM.
- [Chávez et al., 2001] Chávez, E., Navarro, G., Baeza-Yates, R., and Marroquín, J. L. (2001). Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321.
- [Chen, 2014a] Chen, L.C., P. G. K. I.-M. K. Y. A. (2014a). Semantic image segmentation with deep convolutional nets and fully connected crfs. arxiv preprint arxiv. page 1412.7062.
- [Chen, 2014b] Chen, X., Y. A. (2014b). Articulated pose estimation by a graphical model with image dependent pairwise relations. *Advances in Neural Information Processing Systems*, pages 1736–1744.
- [Ciaccia et al., 1997] Ciaccia, P., Patella, M., and Zezula, P. (1997). M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Proceedings of the International Conference on Very Large Data Bases*, pages 426–435, San Francisco, CA, USA.
- [Clarkson, 2006] Clarkson, K. L. (2006). *Nearest-Neighbor Methods for Learning and Vision: Theory and Practice*, chapter Nearest-Neighbor Searching and Metric Space Dimensions, pages 15–59. MIT Press.
- [Datar et al., 2004a] Datar, M., Immorlica, N., Indyk, P., and Mirrokni, V. S. (2004a). Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Twentieth annual Symposium on Computational Geometry*, pages 253–262, Brooklyn, New York, USA.
- [Datar et al., 2004b] Datar, M., Immorlica, N., Indyk, P., and Mirrokni, V. S. (2004b). Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the annual symposium on computational geometry*, pages 253–262, New York, NY, USA.
- [Del Corso et al., 2005] Del Corso, G. M., Gullí, A., and Romani, F. (2005). Ranking a stream of news. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, pages 97–106, New York, NY, USA. ACM.

[Deng, 2009] Deng, J., D. W. S. R.-L. L. K. F.-F. L. (2009). Imagenet: A large-scale hierarchical image database. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, page 248–255.

[Di Ieva, 2016] Di Ieva, A. (2016). *The Fractal Geometry of the Brain*. Springer Series in Computational Neuroscience. Springer New York.

[Dong et al., 2008] Dong, W., Wang, Z., Josephson, W., Charikar, M., and Li, K. (2008). Modeling LSH for performance tuning. In *Proceedings of the International Conference on Information and Knowledge Engineering*, pages 669–678, Napa Valley, California, USA.

[Duda et al., 2001] Duda, R. O., Hart, P. E., and Stork, D. G. (2001). *Pattern classification*. J. Wiley Sons, New York, Chichester, Weinheim. La page de titre porte en plus A Wiley-Interscience publication.

[Elena E. and Matteucci, 2001] Elena E., K. K. and Matteucci, M. (2001). Wekwek: A study in fractal dimension and dimensionality reduction. *ACM*.

[Faloutsos and Kamel, 1994] Faloutsos, C. and Kamel, I. (1994). Beyond uniformity and independence: Analysis of r-trees using the concept of fractal dimension. In Vianu, V., editor, *PODS*, pages 4–13. ACM Press.

[Faloutsos et al., 1994a] Faloutsos, C., Ranganathan, M., and Manolopoulos, Y. (1994a). Fast subsequence matching in time-series databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data Conference*, pages 419–429, New York, NY, USA.

[Faloutsos et al., 1994b] Faloutsos, C., Ranganathan, M., and Manolopoulos, Y. (1994b). Fast subsequence matching in time-series databases. *SIGMOD Rec.*, 23(2):419–429.

[Fan, 2015] Fan, X., Z. K. L. Y.-W. S. (2015). Combining local appearance and holistic view: Dual-source deep neural networks for human pose estimation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1347–1355.

[Fayyad et al., 1996] Fayyad, U. M., Piatetsky-Shapiro, G., and Smyth, P. (1996). From data mining to knowledge discovery: an overview. In *Advances in knowledge discovery and data mining*, pages 1–34. American Association for Artificial Intelligence.

[Gionis et al., 1999] Gionis, A., Indyk, P., and Motwani, R. (1999). Similarity Search in High Dimensions via Hashing. In *Proceedings of the International Conference on Very Large Data Bases*, pages 518–529, San Francisco, CA, USA.

- [Girshick, 2014] Girshick, R., D. J. D. T.-M. J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 580—587.
- [Girshick, 2015] Girshick, R. (2015). Fast r-cnn. *Proceedings of the IEEE International Conference on Computer Vision*, pages 1440—1448.
- [Gong, 2014] Gong, Y., W. L. G. R.-L. S. (2014). Multi-scale orderless pooling of deep convolutional activation features. *Proceedings of European Conference on Computer Vision*, pages 392—407.
- [Gong et al., 2013] Gong, Y., Lazebnik, S., Gordo, A., and Perronnin, F. (2013). Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(12):2916–2929.
- [Gower, 1982] Gower, J. C. (1982). Euclidean distance geometry. *The Mathematical Scientist*, 7:1–14.
- [Guttman, 1984] Guttman, A. (1984). R-trees: a dynamic index structure for spatial searching. *SIGMOD Record*, 14:47–57.
- [Han et al., 2011] Han, J., Pei, J., and Kamber, M. (2011). *Data mining: concepts and techniques*. Elsevier.
- [Hariharan, 2014] Hariharan, B., A. P. G. R.-M. J. (2014). Simultaneous detection and segmentation. *Proceedings of European Conference on Computer Vision*, pages 297—312.
- [He, 2014] He, K., Z. X. R. S.-S. J. (2014). : Spatial pyramid pooling in deep convolutional networks for visual recognition. *Proceedings of European Conference on Computer Vision*, pages 346—361.
- [Hjaltason and Samet, 2003] Hjaltason, G. R. and Samet, H. (2003). Index-driven similarity search in metric spaces. *ACM Transactions on Database Systems*, 28(4):517–580.
- [Indyk and Motwani, 1998] Indyk, P. and Motwani, R. (1998). Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, New York, NY, USA.
- [Jr. et al., 2010a] Jr., C. T., Traina, A. J. M., and Faloutsos, C. (2010a). Fast feature selection using fractal dimension - ten years later. *JIDM*, 1(1):17–20.
- [Jr. et al., 2010b] Jr., C. T., Traina, A. J. M., Wu, L., and Faloutsos, C. (2010b). Fast feature selection using fractal dimension. *JIDM*, 1(1):3–16.

[Kalantidis and Avrithis, 2014] Kalantidis, Y. and Avrithis, Y. (2014). Locally optimized product quantization for approximate nearest neighbor search. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2329–2336.

[Kelvin Lin, 2015] Kelvin Lin, H.-F. Yang, J.-H. H. C.-S. C. (2015). Deep learning of binary hash codes for fast image retrieval. In *CVPR Workshop (CVPRW) on Deep Learning in Computer Vision*, CVPRW’15. IEEE.

[Krizhevsky et al.,] Krizhevsky, A., Nair, V., and Hinton, G. Cifar-10 (canadian institute for advanced research).

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems*, NIPS’12, pages 1097–1105, USA. Curran Associates Inc.

[Krizhevsky, 2012] Krizhevsky, A., S.-I. H. G. (2012). Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, pages 1097—1105.

[Kulis and Grauman, 2009] Kulis, B. and Grauman, K. (2009). Kernelized locality-sensitive hashing for scalable image search. In *Proceedings of the International Conference on Computer Vision*, pages 2130 –2137, Kyoto, Japan.

[Le et al., 2015] Le, Q. V., Brain, G., and Inc, G. (2015). A tutorial on deep learning part 2: Autoencoders, convolutional neural networks and recurrent neural networks.

[LeCun, 1998] LeCun, Y., B.-L. B. Y.-H. P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, (86):2278—2324.

[LeCun and Cortes, 2010] LeCun, Y. and Cortes, C. (2010). MNIST handwritten digit database.

[Li et al., 2015] Li, Z., Liu, J., Tang, J., and Lu, H. (2015). Robust structured subspace learning for data representation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 37(10):2085–2098.

[Lin, 2013] Lin, M., C.-Q. Y. S. (2013). Network in network. arxiv preprint arxiv. page 1312.4400.

[Lin, 2015] Lin, G., S.-C. R. I.-e. a. (2015). Efficient piecewise training of deep structured models for semantic segmentation. arxiv preprint arxiv. page 1504.01013.

- [Liu, 2015a] Liu, Y., G.-Y. W. S.-L. M. (2015a). Deepindex for accurate and efficient image retrieval. : *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*, pages 43—50.
- [Liu, 2015b] Liu, Z., L.-X. L. P.-L. C. T.-X. (2015b). Semantic image segmentation via deep parsing network. *Proceedings of the IEEE International Conference on Computer Vision*, pages 1377–1385.
- [Liu et al., 2012] Liu, W., Wang, J., Ji, R., Jiang, Y.-G., and Chang, S.-F. (2012). Supervised hashing with kernels. In *CVPR*, pages 2074–2081. IEEE Computer Society.
- [Long, 2015] Long, J., S.-E. D. T. (2015). Fully convolutional networks for semantic segmentation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440.
- [Lv et al., 2007] Lv, Q., Josephson, W., Wang, Z., Charikar, M., and Li, K. (2007). Multi-probe LSH: efficient indexing for high-dimensional similarity search. In *Proceedings of the International Conference on Very Large Data Bases*, pages 950–961, Vienna, Austria.
- [Manning and Schutze, 2008] Manning, C. D., R. P. and Schutze, H. (2008). Introduction to information retrieval. *Cambridge University Press, New York, NY, USA*.
- [Navarro, 2002] Navarro, G. (2002). Searching in metric spaces by spatial approximation. *The VLDB Journal*, 11(1):28–46.
- [Navarro et al., 2007] Navarro, G., Paredes, R., and Chávez, E. (2007). t-Spanners for metric space searching. *Data and Knowledge Engineering*, 63:820–854.
- [Netzer et al., 2011] Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Y Ng, A. (2011). Reading digits in natural images with unsupervised feature learning.
- [Ng et al., 2016] Ng, A. Y., Coates, A., Le, Q., Lee, H., and Maas, A. (2016). Ufldl tutorial - deep learning.
- [Ocsa et al., 2007] Ocsa, A., Bedregal, C., and Cuadros-Vargas, E. (2007). A new approach for similarity queries using neighborhood graphs. In *Proceedings of the Brazilian Symposium on Databases*, pages 131–142, João Pessoa, Paraíba, Brasil.
- [Ocsa and Sousa, 2010] Ocsa, A. and Sousa, E. P. M. (2010). An Adaptive Multi-level Hashing Structure for Fast Approximate Similarity Search. *Journal of Information and Data Management*, 1(3):359–374.

[Oquab, 2015] Oquab, M., B. L. L. I.-S. J. (2015). Is object localization for free?—weakly-supervised learning with convolutional neural networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 685—694.

[Ravichandran and Hovy, 2005] Ravichandran, D., P. P. and Hovy, E. (2005). Randomized algorithms and nlp: Using locality sensitive hash function for high speed noun clustering. *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics, ACL'05*, page 622–629.

[Razavian, 2014] Razavian, A.S., S. J. M. A.-C. S. (2014). A baseline for visual instance retrieval with deep convolutional networks. arxiv preprint arxiv. *Proceedings of European Conference on Computer Vision*, page 1412.6574.

[Ren, 2015] Ren, S., H. K. G. R.-S. J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in Neural Information Processing Systems*, pages 91—99.

[Sablayrolles et al., 2016a] Sablayrolles, A., Douze, M., Jégou, H., and Usunier, N. (2016a). How should we evaluate supervised hashing?

[Sablayrolles et al., 2016b] Sablayrolles, A., Douze, M., Jégou, H., and Usunier, N. (2016b). How should we evaluate supervised hashing? *CoRR*.

[Scherer, 2010] Scherer, D., M. A. B. S. (2010). Evaluation of pooling operations in convolutional architectures for object recognition. *International Conference on Artificial Neural Networks*, pages 92—101.

[Schroeder and Fractals,] Schroeder, M. and Fractals, C. Power laws: Minutes from an infinite paradise. 1991.

[Schroeder, 1991] Schroeder, M. R. (1991). *Fractals, Chaos, Power Laws: minutes from an infinite paradise*. W. H. Freeman and Company.

[Sellis et al., 1987] Sellis, T. K., Roussopoulos, N., and Faloutsos, C. (1987). The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. In *Proceedings of the International Conference on Very Large Data Bases*, pages 507–518, San Francisco, CA, USA.

[Sermanet, 2013] Sermanet, P., E. D. Z. X.-M. M. F.-R. L.-Y. (2013). Overfeat: Integrated recognition, localization and detection using convolutional networks. arxiv preprint arxiv. page 1312.6229.

[Sharif Razavian, 2014] Sharif Razavian, A., A. H. S.-J. C. S. (2014). Cnn features off-the-shelf: an astounding baseline for recognition. : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 806–813.

[Shen et al.,] Shen, F., Shen, C., Liu, W., and Shen, H. T. In *CVPR*.

- [Simonyan, 2014] Simonyan, K., Z. A. (2014). Very deep convolutional networks for largescale image recognition. arxiv preprint arxiv. page 1409.1556.
- [Skopal et al., 2005] Skopal, T., Pokorný, J., and Snásel, V. (2005). Nearest Neighbours Search Using the PM-Tree. In *In Proceedings of Database Systems for Advanced Applications*, pages 803–815, Beijing, China.
- [Slaney and Casey, 2008] Slaney, M. and Casey, M. (2008). Locality-Sensitive Hashing for Finding Nearest Neighbors. *IEEE Signal Processing Magazine*, 25(1):128–131.
- [Smola and Vishwanathan, 2008] Smola, A. J. and Vishwanathan, S. (2008). *Introduction to Machine Learning*. Cambridge University Press.
- [Sun, 2014] Sun, S., Z. W. L. H.-T. Q. (2014). Search by detection: Object-level feature for image retrieval. : *Proceedings of International Conference on Internet Multimedia Computing and Service*, page 46.
- [Sutherland, 1975] Sutherland, W. (1975). *Introduction to Metric and Topological Spaces*. Open university set book. Clarendon Press.
- [Szegedy et al., 2013] Szegedy, C., Toshev, A., and Erhan, D. (2013). Deep neural networks for object detection. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 26*, pages 2553–2561. Curran Associates, Inc.
- [Szegedy, 2015] Szegedy, C., L. W. J. Y.-S. P. R.-S. A.-D. E. D. V. V. R. A. (2015). Going deeper with convolutions. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1—9.
- [Tan et al., 2005] Tan, P.-N., Steinbach, M., and Kumar, V. (2005). *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
- [Tao et al., 2010] Tao, Y., Yi, K., Sheng, C., and Kalnis, P. (2010). Efficient and accurate nearest neighbor and closest pair search in high-dimensional space. *ACM Transactions on Database Systems*, 35(3):1–46.
- [Tompson, 2014] Tompson, J.J., J. A. L. Y.-B. C. (2014). Joint training of a convolutional network and a graphical model for human pose estimation. *Advances in Neural Information Processing Systems*, pages 1799–1807.
- [Toshev, 2014] Toshev, A., S. C. (2014). Deeppose: Human pose estimation via deep neural networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1653–1660.

[Traina et al., 2001] Traina, A., Traina, C., Papadimitriou, S., and Faloutsos, C. (2001). Tri-plots: Scalable tools for multidimensional data mining. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 184–193, New York, NY, USA. ACM.

[Traina et al., 2005] Traina, C., Sousa, E. P. M., and Traina, A. J. M. (2005). Using fractals in data mining. *New Generation of Data Mining Applications*, 1:599–630.

[Traina et al., 2002] Traina, Jr., C., Traina, A., Filho, R. S., and Faloutsos, C. (2002). How to improve the pruning ability of dynamic metric access methods. In *Proceedings of the International Conference on Information and Knowledge Engineering*, pages 219–226, New York, NY, USA.

[Traina et al., 2010] Traina, Jr., C., Traina, A. J. M., and Faloutsos, C. (2010). Fast Feature Selection using Fractal Dimension - Ten Years Later. *Journal of Information and Data Management*, 1(1):17–20.

[Traina C. et al., 2002] Traina C., J., Traina, A., Faloutsos, C., and Seeger, B. (2002). Fast Indexing and Visualization of Metric Data Sets using Slim-Trees. *IEEE Transactions on Knowledge and Data Engineering*, 14(2):244–260.

[Traina Jr et al., 2010] Traina Jr, C., Traina, A., Wu, L., and Faloutsos, C. (2010). Fast feature selection using fractal dimension. *Journal of Information and data Management*, 1(1):3.

[Traina Jr et al., 1999] Traina Jr, C., Traina, A. J., and Faloutsos, C. (1999). Distance exponent: A new concept for selectivity estimation in metric trees. Technical report, DTIC Document.

[Uijlings, 2013] Uijlings, J.R., v. d. S. K.-G. T. S.-A. (2013). Selective search for object recognition. *International Journal of Computer Vision*, 104:154—171.

[Vieira et al., 2004] Vieira, M. R., Jr., C. T., Chino, F. J. T., and Traina, A. J. M. (2004). DBM-Tree: A Dynamic Metric Access Method Sensitive to Local Density Data. In *Proceedings of the Brazilian Symposium on Databases*, pages 163–177, Brasília, Brasil.

[Volnyansky and Pestov, 2009] Volnyansky, I. and Pestov, V. (2009). Curse of Dimensionality in Pivot Based Indexes. In *Proceedings of the International Workshop on Similarity Search and Applications*, Washington, DC, USA.

[Wan, 2014] Wan, J., W. D. H. S.-W. P. Z.-J. Z.-Y. L. J. (2014). Deep learning for content-based image retrieval: A comprehensive study. : *Proceedings of the 22nd ACM International Conference on Multimedia*, ACM, pages 157–166.

- [Wang et al., 2010] Wang, J., Kumar, S., and Chang, S.-F. (2010). Semi-Supervised Hashing for Scalable Image Retrieval. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3424 –3431, San Francisco, USA.
- [Xiaoyue Wang and Keogh, 2013] Xiaoyue Wang, Abdullah Mueen, H. D.-G. T. P. S. and Keogh, E. (2013). Experimental comparison of representation methods and distance measures for time series data. *Data Min Knowl Disc*, 26:275–309.
- [Y et al., 2010] Y, L.-Q., W, L.-Z., L, B., Z, J.-H., and Z, X.-D. (2010). Combined static and dynamic immutability analysis of java program. *Chinese Journal of Computers*. Vol. 33, pages 736–746.
- [Yianilos, 1993] Yianilos, P. N. (1993). Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings - Annual Symposium on Discrete Algorithms*, pages 311–321, Philadelphia, PA, USA.
- [Yosinski et al., 2014] Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? *CoRR*, abs/1411.1792.
- [Zeiler, 2013] Zeiler, M. (2013). *Hierarchical convolutional deep learning in computer vision*. PhD thesis, NEW YORK UNIVERSITY.
- [Zhang et al., 2011] Zhang, D., Agrawal, D., Chen, G., and Tung, A. (2011). HashFile: An efficient index structure for multimedia data. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 1103 –1114, Hannover, Germany.
- [Zhang, 2015] Zhang, Y., S. K. V. R.-P. G. L. H. (2015). Improving object detection with deep convolutional networks via bayesian optimization and structured prediction. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 249—258.
- [Zheng, 2015] Zheng, S., J. S. R.-P.-B. V. V. S. Z.-D. D. H. C. T. P. (2015). Conditional random fields as recurrent neural networks. *Proceedings of the IEEE International Conference on Computer Vision*, pages 1529–1537.
- [Zhu, 2015] Zhu, Y., U. R. S.-R.-F. S. (2015). segdepm: Exploiting segmentation and context in deep neural networks for object detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4703—4711.