Chapter 2
Overview of Basic Methods for Data Science

Li M. Chen

Abstract Data science utilizes all mathematics and computer sciences. In this chapter, we give a brief review of the most fundamental concepts in data science: graph search algorithms, statistical methods especially principal component analysis (PCA), algorithms and data structures, and data mining and pattern recognition. This chapter will provide an overview for machine learning in relation to other mathematical tools. We will first introduce graphs and graph algorithms, which will be used as the foundation of a branch of artificial intelligence called search. The other three branches of artificial intelligence are learning, planning, and knowledge representation. Classification, also related to machine learning, is at the center of pattern recognition, which we will discuss in Chap. 4. Statistical methods especially PCA and regression will also be discussed. Finally, we introduce concepts of data structures and algorithm design including online search and matching.

2.1 "Hardware" and "Software" of Data Science

Data storage, data communication, security, and computing machinery can be considered as the "hardware" in data science. Artificial intelligence (AI), statistical methods, algorithm design, and possible new mathematical methodologies can be viewed as the "software" of data science.

Existing methods play a primary role in data science today. There are three aspects that are important to modern data related problems. We want to give brief statements on each of them. For mathematicians, these are part of the motivation for why we are interested in data science problems and how we start to tackle these problems. For some researchers, the motivation is figuring out what we should do to find the new method to solve a challenging problem.

L.M. Chen (✉)
The University of the District of Columbia, Washington, DC, USA
e-mail: lchen@udc.edu

© Springer International Publishing Switzerland 2015

L.M. Chen et al., *Mathematical Problems in Data Science*,
DOI 10.1007/978-3-319-25127-1_2

2.1.1 Searching and Optimization

Search is one of the most important tasks in AI [34]. General search, in fact, is not a topic of AI, but it belongs to algorithm design and usually relates to graph search. For instance, finding a special object in a graph or finding the best path. When the area in which we are searching is very large, this type of algorithm is not feasible, meaning that we can not complete the search in an allotted time frame. AI search will look for a best or near best way to limit some of the unnecessary paths that unlikely contains the object or target we are looking for.

For instance, A^* search is a type of search based on the so-called breadth first search method in graph theory, which finds the best possible direction to look for the answer.

2.1.2 Decision Making

Everyday, we need to make many decisions. What time should I leave the house for work? What I am going to eat for lunch? Can I come home earlier to avoid traffic? Decision making usually refers to statistical methods and discrete (algorithmic) methods.

Everyone knows the regression method to find a line that best fits sample data in 2D. We usually use the least square method to find such a line, which is a statistical method. Another method is called neural networks that employs the least square method too. It uses multiple layers where each layer performs linear transformation like operations. The process also adds some nonlinear procedures and automated adjustments of parameters based on the least squares.

The decision tree method is a typical algorithmic method [12]. It does not look at the decision for the whole data set. It makes a decision at each individual step. For instance, John has a meeting in the morning, so he leaves home 30 min earlier than usual. We will set up a rule such as "if someone has a meeting in the morning, then he or she needs to leave home 30 min early." When John meets heavy traffic, he will choose to use a toll-road. We then have another if-then rule.

The decision tree method preforms a set of "IF-THEN" statements. The decision is made based on each condition that has occurred. The structure of the method or algorithm is like a binary tree where each node will test if the condition is met. Then, the algorithm will direct John to go left or right in the child node.

2.1.3 Classification

Classification is the placement of certain samples into categories [34, 39]. If the categories are predefined, then we call it supervised classification. Otherwise, we call it unsupervised classification.

The problem of classification is everywhere. For instance, to measure if a person is overweight, we use something called BMI, which is a function of weight and height. When $f(W,H) = (W/H^2) \times 703$, where $W =$ (Weight in Pounds), $H =$ (Height in inches). If the value is above the average, we say the person is overweight. Otherwise, the person is normal. Using this method, people are categorized into two classes: (1) normal and (2) overweight. If we are only interested in people who has the same hight, say 72 inches. So the formula will be $f(W) = (W/(72 \times 72)) \times 703$. The new function that determines the classification is linear. Therefore we call it the linear classifier. Assume the clip-level for the normal class is under 25, so everyone who has BMI less than 25 is healthy (with height = 72 inches). This is also called a supervised classification (since we have predefined a clip-level value).

In another example, we have a picture that we want to separate into different objects. However, we do not know how many objects are in the picture. We can only use the intensity of the image to do the classification. The pixels (picture elements) similar in color or gray scale will be treated as the same object if they are close to each other. This classification is an unsupervised classification.

2.1.4 Learning

For some problems, it is hard to generate rules before we start to process the data such as a function. We generate the model dynamically, meaning that we need to use the samples to obtain the parameters of a classification system. This method usually refers to a learning method. Let us assume that we have satellite data sets from a GPS system. We want to classify the geographical areas into cities, forests, and water. The volume of data is so big that we ask some experts to select 1000 points (locations) and then identify those points as city, forest, or water. Then, we recollect the GPS image data from the database to retrieve the values of the image intensity (with different bandwidths so they have a total of seven bands for seven pictures). We now have data for these 1000 points $< F, v >$. Note that the decision about these 1000 data points is made by experts. We also have the location of the data as well. So we have $(x, y, f_1, ..., f_7, v)$ where v is a value for city, forest, or water. We call this data set the samples S. In order to make a learning system, we split S into two sets: the training set S_{tr} and the testing set S_{te}. We usually request $S_{tr} \cap S_{te} = \emptyset$. However, in statistical learning methods, the researcher usually intends to make S_{tr} and S_{te} overlap for more profound analysis.

We use the training set to find the model for the data and then use the testing set to test the model system we used to see if it is correct. Using the simple example of linear system classification, we can use S_{tr} to find a linear function for finding the PMI of a city population. (This is not a global solution, so we decided to find the parameter or coefficients of the linear equation, such as c1 and c2.) Then, we want to see if this linear model works or not. We can use S_{te} to test and find the correctness of the testing data samples.

We could also randomly reselect S_{tr} and S_{te} to test this set. If we are satisfied, we can use this model. If not, we can select another method. This way is called machine learning [34].

In this chapter, we review basic methods for data science.

2.2 Graph-Theoretic Methods

The graph in mathematics is a general structure for discrete objects. Theoretically, every computerized data storage or data processing mechanism can be deduced into an operational process of graphs. A graph G consists of two sets V and E, where V is the set of vertices and E is the set of pairs of vertices called edges. An edge is said to be incident to the vertices if it joins [3, 12, 17].

2.2.1 Review of Graphs

Graph $G = (V, E)$ is called a simple graph if every pair of vertices has at most one edge that is incident to these two vertices and there is no loop $(a, a) \in E$ for any $a \in V$.

In this book, assume that $G = (V, E)$ is an undirected graph, which means that if $(a, b) \in E$, then $(b, a) \in E$ or $(a, b) = (b, a)$. a and b are also called ends, endpoints, or end vertices of edge (a, b). It is possible for a vertex in a graph to not belong to any edge. V and E are usually finite, and the order of a graph is $|V|$, which is the number of vertices. The size of a graph is linear to $\max\{|V| + |E|\}$, meaning that it requires this much memory to store the graph.

The degree of a vertex is the number of edges that incident with (or link to) it. A loop is an edge that links to the same vertex. A weighted graph means that we assign a number on each edge that may indicate the distance between the two vertices.

If (p, q) is in E, then p is said to be adjacent to q. Let $p_0, p_1, \ldots, p_{n-1}, p_n$ be $n+1$ vertices in V. If (p_{i-1}, p_i) is in E for all $i = 1, \ldots, n$, then $\{p_0, p_1, \ldots, p_{n-1}, p_n\}$ is called a path. If $p_0, p_1, \ldots, p_{n-1}, p_n$ are distinct vertices, then the path is called a simple path.

A simple path $\{p_0, p_1, \ldots, p_{n-1}, p_n\}$ is closed if (p_0, p_n) is an edge in E. A closed path is also called a cycle. Two vertices p and q are connected if there is a path $\{p_0, p_1, \ldots, p_{n-1}, p_n\}$ where $p_0 = p$ and $p_n = q$. G is also called connected if every pair of vertices in G is connected. In this book, it is always assumed that G is connected (unless otherwise specified).

There are three algorithms that are important to data science mentioned in this book [12]. These algorithms are: (1) The breadth first search algorithm, (2) The shortest path algorithm, and (3) The minimum spanning tree algorithm.

The maximum flow and minimum cut algorithms are also related to this topic. The data structures that refer to these two techniques are queues and stacks made by adjacency lists.

2.2.2 Breadth First Search and Depth First Search

Breadth first search is a fast search approach to get a connected component on a graph. It begins at a vertex p and searches for all adjacent vertices. Then, it "inserts" all adjacent points (neighbors) into a queue. "Removing" a vertex from the queue, the algorithm calls the point p and then repeatedly finds p's neighbors until the queue is empty. Marking all the vertices we visited, the marked vertices form a connected component. This technique was introduced in [12].

Algorithm 2.1. Breadth first search technique for all point-connected components

Step 1: Let p_0 be a node in G. Set
$L(p_0) \leftarrow *$ and $QUEUE \leftarrow QUEUE \cup p_0$
i.e., labeling p_0 and p_0 is sent a queue $QUEUE$.

Step 2: If QUEUE is empty, go to Step 4; otherwise,
$p_0 \leftarrow QUEUE$ (top of $QUEUE$). Then,
$L(p_0) \leftarrow 0$.

Step 3: For each p with an edge linking to p_0 ,
do
$QUEUE \leftarrow QUEUE \cup p$ and $L(p) \leftarrow *$. Then, go to Step 2.

Step 4: $S = \{p : L(p) = 0\}$ is a connected part.

Step 5: If p is an un-visited vertex, then let $p_0 = p$. Repeat Step 1.
Otherwise Stop.

Breadth first search only visits a new node twice: once when the node is inserted into the queue and once when the node is removed from the queue. Therefore, this algorithm is a linear time algorithm and uses a fast search approach. Depth first search is similar to breadth first search. Depth first search will find new nodes continuously until no more new nodes can be found. When the algorithm runs through a complete iteration, it saves all visited nodes in a stack. Since we only go with a path to try and find the "deepest" node, there are other branches we may have missed in the first try. Therefore, the algorithm returns to each visited node in the order it was pushed in stack (first in, last out). We check other branches by popping out nodes. After all nodes are popped out, this algorithm will have found all the nodes in a component. This algorithm will visit all edges at most twice. Therefore, it is also a linear algorithm on the edges.

2.2.3 Dijkstra's Algorithm for the Shortest Path

Finding the shortest path in the weighted graphs is a popular topic in graph theory. Two of the most important algorithms are Dijkstra's algorithm and the Bellman–Ford algorithm [12]. Dijkstra's algorithm is faster but cannot take negative edges.

The idea of the shortest path algorithm consists of the following. Beginning at the vertex, this is the departing node. Then, we record the distance from the departing node to all of its neighbors. The record may not be the shortest path involving these notes, but we update the record and update the values on the nodes, a process called relaxation. After that, we extend the notes by one more edge, do a relaxation, and continue to repeat these two steps until we have reached every node in the graph.

The algorithmic technique used in this problem is called dynamic programming. Even though there may be an exponential number of paths from one vertex to another, if we are only interested in the shortest path, we only need to care about the length we travel. We do not have to pass a certain vertex. There are only $n(n-1)/2$ pairs of vertices. If we update the minimum distance while we calculate, this is called dynamic programming.

Algorithm 2.2. Dijkstra's algorithm for the shortest path

Step 1: Let $T = V$. Choose the source point a
$L(a) = 0; L(x) = \infty$ for all $x \in T - \{a\}$.

Step 2: Find all neighbors v of a node u with the value of L at $L(v) = L(u) + w(u, v)$.
$T \leftarrow T - \{v\}$

Step 3: For each x adjacent to v, do
$L(x) = max\{L(x), min\{L(v), L(x) + w(x, v)\}\}$

Step 4: Repeat steps 2–3 until T is empty.

2.2.4 Minimum Spanning Tree

The minimum spanning tree is to find a tree in a connected weighted graph G such that the tree contains all vertices of G and the total weights are at the minimum.

Kruskal's algorithm can be used to find the minimum spanning tree. In this method, a technique called the greedy algorithm is used. The methodology is to pick up the best one (the shortest edge currently available) in each stage to find an ultimate solution.

In Kruskal's algorithm, the tree T initially contains all vertices but no edges. Because of this, it starts as an iterative process, adding an edge to T under the condition that the edge has the minimum weight. We continue to add a minimum edge to T as long as it does not allow for a cycle to appear in T. When T has $|G| - 1$ edges, the process stops.

Algorithm 2.3. Kruskal's Algorithm for finding a minimum spanning tree T

Step 1 Sort the edges based on the weights of the edges from smallest to largest.
Step 2 Set initial tree T to be empty.
Step 3 Select an edge from the sorted edge list and add it to T if such an added edge does not generate a cycle.
Step 4 T would be the minimal spanning tree if $|V| - 1$ edges are added to T.

The principle where we always make a minimum or maximum selection is called the greedy method. In artificial intelligence, the greedy method is constantly applied in applications [34]. In most cases, the method may not obtain an optimal solution but rather an approximation. However, for the minimum spanning tree, the greedy method would reach the optimal solution.

2.3 Statistical Methods

For a set of sample data points in 2D, the method of principal component analysis (PCA) can be used to find the major direction of the data. This is essential to data processing. The method is also related to regression analysis and other methods in numerical analysis and statistical computing [9, 24, 31]. Even though there are many ways to explain PCA, the best one is through statistics.

For a set of sample points, $x_1, x_2 \cdots, x_n$, the mean is the average value of the x_is

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i.$$

We may use a random variable X to represent this data set.

Variance is the measure of how data points differ from each other in a whole. The sample variance of the data set or the random variable X is defined as

$$Var(X) = \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})^2.$$

The square root of the variance is called the standard deviation σ_X, i.e. $Var(X) = \sigma_X^2$. Note that people sometimes use $n - 1$ instead of n in the formula for an unbiased estimate of the population variance.

For simplicity, let $X = \{x_1, x_2 \cdots, x_n\}$ and $Y = \{x_1, x_2 \cdots, x_n\}$. The covariance of X and Y is defined as

$$Cov(X, Y) = \frac{1}{n} \sum_{i=1}^{n} ((x_i - \bar{x})(y_i - \bar{y})).$$

We also can define the correlation of X and Y as follows:

$$Cor(X, Y) = \frac{Cov(X, Y)}{\sigma_X \cdot \sigma_Y}.$$

Now, let's introduce the PCA, an essential technique in data science. Given a set of 2D points, $(x_1, y_1), \cdots, (x_n, y_n)$, we can treat X as a random variable of the first component and Y as the second component of the 2D vector. We define the covariance matrix as the following.

The geometric direction of a set of vector data points can be determined through PCA. The largest eigenvalue of the matrix indicates the major axis by calculating the corresponding eigenvector.

$$M(X, Y) = \begin{pmatrix} Cov(X, X) & Cov(X, Y) \\ Cov(X, Y) & Cov(Y, Y) \end{pmatrix} \tag{2.1}$$

PCA can be applied to find the major axis of the data in 2D. See Fig. 2.1.

In general, this method can be extended to analyze data in m dimensional space. Assume we have m random variables in X_i and n samples for each variable. In other words, we have N sample points $p_i = (x_1, x_2, \cdots, x_m)$, $i = 1, \ldots, N$. We can extend the covariance matrix to be $m \times m$. Therefore, we will have m eigenvalues, $\lambda_1, \ldots, \lambda_m$. We assume that $\lambda_1 \leq, \ldots, \leq \lambda_m$. The eigenvector V_i of λ_i indicates the i-th principal component. This means that most of the sample data is along the side of vector V_i, compared to vector V_j, if $j > i$.

For data storage purposes, we can determine that most of the data is covered by the first few principal components. We can use linear transformation to attach this to the original data and store the transformed data in a much lower dimension to save space. This is why PCA is one of the most effective methods in many applications in BigData and data science today.

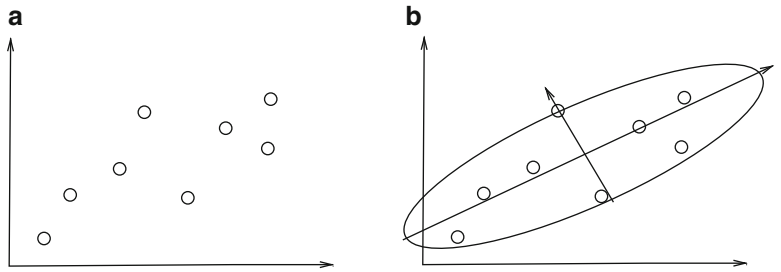

Fig. 2.1 Principal component analysis: (**a**) original data points, and (**b**) eigenvectors of the covariance matrix

2.4 Classification, Clustering, and Pattern Recognition

Classification usually refers to the supervised classification. In this method, we want to know the categories of data points. We usually collect some samples whose classifications are known. It is also called pattern recognition meaning that we want to put a new sample into its pattern category.

Clustering is called unsupervised classification. It is a process where we do not know the pattern. This means that we can only partition the data into several categories [28, 34, 39].

In this section, we introduce the most popular classification methods: k-Nearest Neighbor Method (kNN), and k-Means Method. The methods discussed in this section will be used in image data analysis in Chap. 5 [23, 26, 27, 32, 37].

2.4.1 k-Nearest Neighbor Method

The kNN method is a supervised classification method. It is often used in pattern recognition.

Let us take a sample subset of data points $S = \{x_1, \cdots, x_n\}$. For each x_i we know its classification. If x is a new data point, the shortest distance $d(x, x_i)$, $i = 1, \ldots, n$, indicates the class containing x_i that should include x. This is also called the nearest neighbor method.

kNN is a generalization of the nearest neighbor method. We find the k nearest neighbors in S for x of x in S, then we determine that x will be contained in the class that also contains most of those neighbors [22, 34, 39].

2.4.2 k-Means Method

The k-means method is an unsupervised classification method. Given a set of vectors, if we want to classify the data set into two categories, we can find two centers such that the total summation of the distances of each data point to its center will be minimum.

In Fig. 2.2, we first select two initial vectors as the "centers"(called sites) of the two categories. Second, for each element P in the set, we put it into one of the categories and record the distance from the center. Third, we update the two center locations such that the total summation of the distance becomes smaller. The algorithm will halt if there is no improvement when we move the two centers.

The k-means algorithm can be explained below: Based on the initial k sites, partition the space using the Voronoi diagram. Then, we can move the center to the geometric centroid of the new partition before we recalculate the results. This is an iterated process. The only problem is with the local minimum, meaning that this algorithm may converge to the local minimum for best result. In other words, the algorithm will stop after a local minimum is attained [39].

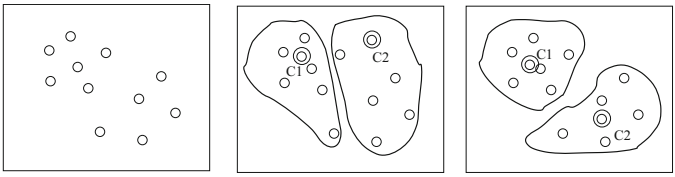

Fig. 2.2 The k mean algorithm

We present the k-means algorithm below:

Algorithm 2.4. The k-means algorithm

Input: For a set of data points $X = \{x_1, \cdots, x_n\}$, each x_i is an m-dimensional real vector.

Output: k-means tries to find $P = \{c_1, c_2, \cdots, c_k\}$, where each c_i is a vector. Then, all the data points will be partitioned into k subsets of X, where S_i associated with c_i and satisfies the following:

$$\min \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in S_i} \left\| \mathbf{x}_j - c_i \right\|^2. \tag{2.2}$$

Step 1 Randomly select k vectors $c_1, c_2, \cdots, c_k$ as the initial centers.

Step 2 For each data point x_i, calculate the distance $d(x_i, c_j)$, setting x_i as S_t if $d(x_i, c_t)$ is the smallest.

Step 3 Calculate the new geometric center of S_j by

$$c_j = \frac{1}{|S_j|} \sum_{x_i \in S_i} x_i$$

Step 4 Repeat Step 2 until the total square of the distances in (2.1) can no longer be improved.

2.5 Numerical Methods and Data Reconstruction in Science and Engineering

The most popular problem in data reconstruction is fitting a curve or a surface such as Bezier polynomials and B-Splines. Data fitting has two meanings: interpolation and approximation. Interpolation means fitting the data exactly on the sample values while approximation means setting a fitted function near the sample values. Other methods include data reconstruction from projection, such as CT scans in medical imaging, and discrete fitting, which can be found in [8, 20].

The most useful feature in data reconstruction is finding the missing data elements of massive data collections in data science. IT is especially important to fill the gaps in financial marketing data. For instance, during holidays we do not have actual New York stock exchange information. However, the industry is not on vacation. Therefore, the data information could be restored by filling in the predicted information. This technique has been used for a long time. See Chap. 9.

In this section, we will not do a comprehensive review of numerical and computational methods for data fitting. We will only discuss a principle of data fitting using the least squares method. For more information on the numerical method, we recommend the book Numerical Recipes and [8, 31].

Assume we have n sample data points in (x_i, y_i). These points are on a straight line, but the sampling process may have some observation errors. Based on the equation of a line (2.2), we have:

$$y_i = a + bx_i \, , i = 1, \cdots, n \tag{2.3}$$

However, since there are observation or sampling errors, this equation may not always hold true for all sample points. In other words, $e_i = y_i - (a + bx_i)$ is not always zero. What we can do is find a and b such that the summation of the square of errors $((e_i)^2)$ is minimized. So, it is called the least squares method .

$$\text{Minimize } E = \Sigma_{i=1}^{n} (e_i)^2 \tag{2.4}$$

where E is the function of a and b so that $E = E(a, b)$. To get the minimum $E(a, b)$, according to the extreme value theorem in calculus, the following two equations must hold:

$$\frac{\partial E}{\partial a} = \frac{\partial}{\partial a}(\Sigma(y_i - (a + bx_i))^2) = \frac{\partial}{\partial a}(\Sigma - 2(y_i - (a + bx_i))$$
$$\frac{\partial E}{\partial b} = \frac{\partial}{\partial b}(\Sigma(y_i - (a + bx_i))^2) = \frac{\partial}{\partial b}(\Sigma - 2x_i(y_i - (a + bx_i)).$$

Thus, we have a system of linear equations:

$$na + (\Sigma x_i) \cdot b = \Sigma y_i$$
$$(\Sigma x_i) \cdot a + (\Sigma x_i^2) \cdot b = \Sigma (x_i \cdot y_i).$$

We can easily solve these equations. In Chap. 9, we give some detailed descriptions of curve fitting and its application to financial industry.

2.6 Algorithm Design and Computational Complexity

Algorithms are at the heart of computer science [12]. Everything related to data processing can be viewed as a type of algorithm design. However, in some cases, such as linear regression, the mathematical part of the algorithm dominates the algorithm. Therefore, we do not really think of it as a typical algorithm, which means that the procedure has some complexity and is not straightforward.

The Euclidean algorithm that is used to find the greatest common divisor (GCD) is referred to as the first algorithm since the procedure is more important in yielding the results than the mathematical formula derivation.

Many mathematical algorithms for data processing do not care for data structure or the format of the data stored in computer memory or computer disks. This is because we used to focus on the mathematical part of algorithm design. However, in data science, as we have explained, data is stored in different forms and different places. Data structure becomes the key to efficient processing.

2.6.1 Concepts of Data Structures and Databases

Data structures are fundamental to data science. It is the way different data types are held in storage or memory for fast processing. The simplest data structure is the array. We will first introduce the concept of the queue, stack, linked list, and tree. Then, we will focus on complex data structures such as quadtrees, octrees, and R-trees. Those data structures are often used in geometric data storage and massive data collections.

Databases are special systems that hold a large amount of data. A database system such as Oracle or MySQL contains functions that can be called to store and obtain data samples in the system. A data structure will be implemented in a database system. For instance, Oracle is a relational database. It means that a table or an array is the basic data structure.

A record is a row of data in the array (or the table) that stores all information about an individual. For instance, we store all information about a student at a university in a record. The record contains the student's name, ID number, email address, etc. A column of the array (or the table) stores all information about a property of the data set, such as the student ID number for all students at the university.

A database system not only contains data, but also includes functions that can retrieve the data. For example, if we only need 100 students midterm test scores, we can get this data by asking the system. This is called a query. This query is based on the relation of the table. (A relation in mathematics is an algebraic structure that is a collection of select vectors.) Data science must be able to deal with different types of database systems.

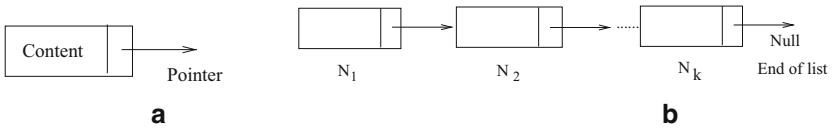

Fig. 2.3 A linked list: (**a**) a node configuration, (**b**) an example of a linked list

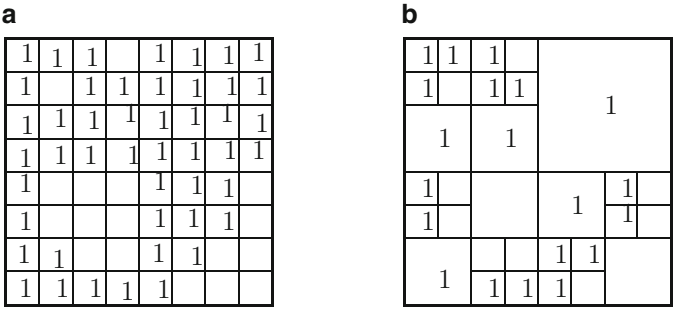

Fig. 2.4 An example of Quad-trees: (**a**) the original image, and (**b**) the quad-tree representation of (a). [7, 9]

2.6.2 Queues, Stacks, and Linked Lists

Queues, Stacks, and Linked Lists are efficient data structures to data science. To understand these data structures is essential to process massive data. A queue is a special data arrangement where the first entry is removed first; a stack performs differently in that the last data entry is removed first. A linked list is a collection of nodes that contains data and a pointer to the next node. See Fig. 2.3.

A tree is similar to a linked list. A node in a tree may contain two or more pointers. A binary tree is a tree where each node only has two pointers.

2.6.3 Quadtrees, Octrees, and R-trees

Quadtrees, Octrees, and *R*-trees are usually used for geometric data storage and retrieval.

The quadtree is for 2D data sets or objects. The root of a quadtree contains a 4-subtree. If we partition a 2D square into 4 small, identical subsquares, then each subtree will represent a subsquare. When storing a binary image, we can save space when a subsquare contains no information (it is blank) [20, 30, 33].

The quadtree method partitions a 2D space into four equal quadrants, subquadrants, and so on. We stop at a node called a leaf if all the elements represented by the node have the same value. We continue decomposing if this is not the case. See Fig. 2.4 [9].

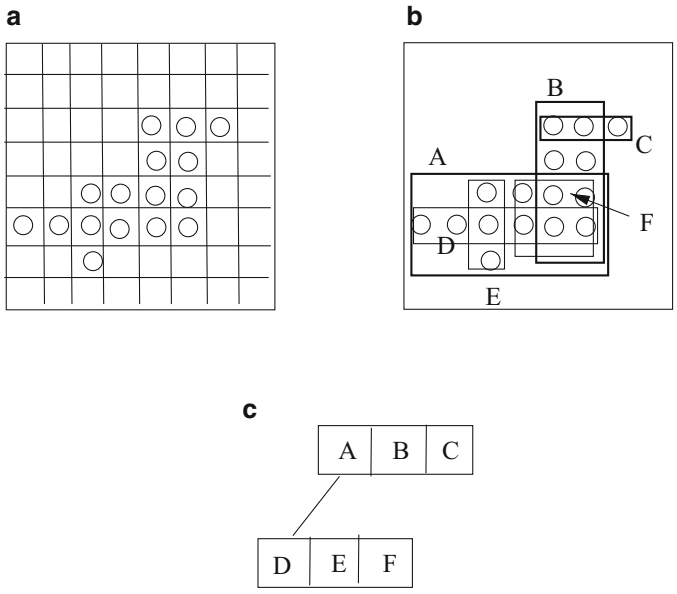

Fig. 2.5 *R*-Trees: (**a**) original data, (**b**) data points contained in rectangular boxes, and (**c**) tree hierarchy of boxes

The depth or height of the tree is at most $\log_2 n$ if we are dealing with an $n \times n$ region. We can save space since one leaf may represent many pixels in real images.

The octree uses the same principle as the quadtree, but it is used for 3D data storage. This is because we can split a 3D cube into 8 subcubes. Therefore, octrees are used for spatial data representation [36].

Each node in an octree represents a cubic region. We subdivide it into eight octants that will be represented by eight subnodes. When the values of the subregion are the same. The corresponding node will be a leaf. Otherwise, we keep dividing the region.

R-trees are mainly used in query and search for the geometric data analysis. The idea is to use a rectangle to cover all data points in a 2D space and then use smaller subrectangles to cover the points inside the parent rectangle. The subrectangles in the same level can overlap each other. See Fig. 2.5. The dimensions of each rectangle are not fixed. This allows the most flexibility in the partition of data in a space.

A popular problem in data science is as follows: Given a point location, find all of the rectangles or leaves that contain a chosen point [13, 42]. The idea of *R*-trees is to use the bounding rectangular boxes to decide whether or not we need to search inside the subtree. Therefore, most of the nodes in the tree are never read during a search. There are many applications that use *R*-trees in wireless networking. For instance, in queries for wireless broadcast systems, we need to quickly find a group of users in a heavy rain area to tell them that there may be a possibility of flooding. This is especially useful in social network systems. Social networking categorizes

individuals into specific groups . Let us take a social network, such as *facebook*, that does not usually contain location parameters for users. When a local social event needs to find attendees in a hurry, they can ask *facebook* to send messages to local members. *R*-tree is the best way to identify the people to send this message.

2.6.4 NP-Hard Problems and Approximation of Solutions

Some problems are easy to solve such as sorting and searching problems. However, for other problems, even though we can find algorithms to solve the problem, there is no quick or fast algorithm to complete the task. Many of these require exponential time. If we want to have a precise answer, we may need to spend 100 years to solve this using a single machine. Cloud computing provides us the possibility of solving a difficult problem in a relatively short period of time. For instance, in RSA public cryptography, we should use the string (called key) with 2024 bits for encryption. This is because in distributed computing, a host computer links with thousands of other computers, which can break the encryption code if the key is shorter than a certain length. Both Bigdata and data science must consider security problems. However, this is not the main topic of this book.

Roughly speaking, an NP-hard problem would take years to run for a data set with a large input size. A problem is said to be an NP-problem if we can check the answer quickly, e.g. within $O(n)$ time. Many problems are found to have such a property. For instance, the traveling salesmen problem has this property. This problem involves finding a route for a traveler who plans to visit one city in each of the 50 United States exactly once. The question is, is there a route such that the total distance travelled is smaller than a given number K?

If we know the sequence of the list of 50 cites, then we can easily check to see if the answer is correct. If we do not know the answer, then we may need to spend 10 years in finding it [1, 12].

It was proven that the traveling salesmen problem is the hardest of the NP-problems, called an NP-complete problem. It means that if the traveling salesmen problem can be solved in polynomial time, then all NP-problems can be solved in polynomial time with regular computers. There is a famous unsolved problem in mathematics and computer science called the $P = ?NP$ problem. Here, P means that the problem can be solved in polynomial time by regular computers (for example, a Turing machine) and NP means that the problem can be solved in polynomial time by a non-deterministic Turing machine. NP-hard problems mean the problems that are harder than NP-complete problems [12, 21, 25].

Some data science problems are NP-hard. For these problems that are difficult to solve, we may need to find an approximation of the solution. In [40], there are many methods that can be used in finding a near best answer for NP-hard problems.

2.7 Online Searching and Matching

Searching and matching are two basic tasks over the Internet. Search is used to find an object in a set or space. For instance, if we are searching for a number in an integer set, we know the binary search method is the fastest for a sorted array of integers. Matching is usually try to fine a substring in a text that matches a given pattern that is also a string. Today, the image especially the face image matching is very popular. We will discuss the image problems in Chap. 5 [9, 11, 15, 16, 20].

2.7.1 Google Search

Google search is an excellent example for online search using mathematical algorithms [4, 9, 29]. We say that the algorithm is a mathematical algorithm since it uses some deep knowledge of mathematics not only the complexity from computer science point of view.

Google search (also called *PageRank*) seeks to establish a link graph and then calculates the importance of each web node (page). To explain this method, we start with the adjacency matrix of the page link graph, which is a directed graph. This matrix, shown in Fig. 2.6, is as follows [9]:

$$M = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

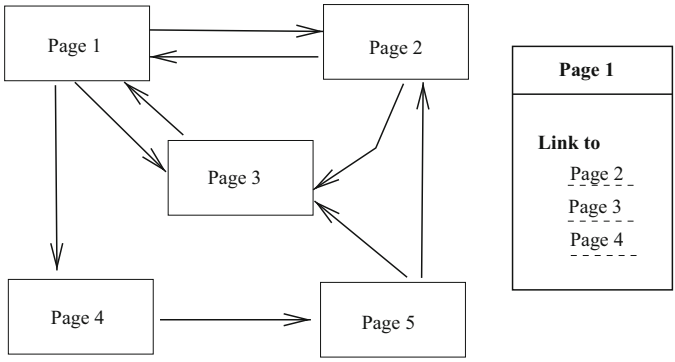

Fig. 2.6 Example of a Web link graph [9]

Its weight graph with the distribution of the average contribution to each of outgoing nodes is shown below. For instance, the first page "Page 1" gives $1/3$ of the contributions to each of its pointed neighbors: Page2, Page3, and Page4. We have the weight matrix:

$$W = \begin{bmatrix} 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \end{bmatrix}$$

The transpose of W will be the matrix we are interested in, denoted by M_{PR}. Such a representation is very intuitive.

On the other hand, Page and Brin have used the following formula to rank the importance of each web page [4] called the PageRank of a page A:

$$PR(A) = \frac{(1-d)}{N} + d(PR(T_1)/C(T_1) + \ldots + PR(T_n)/C(T_n)), \tag{2.5}$$

where N is the total number of pages considered, $PR(T_i)$ is the PageRank of page T_i (which links to page A), $C(T_i)$ is the number of links going out of page T_i, and $d \in [0, 1]$ is a damping factor which is usually set to 0.85.

A simple algorithm runs the above formula in an iterative manner. It stops when an error limit is met.

It is astonishing when we check the relationship between the matrix M_{PR} and $PR(A)$. Let vector $x = (1/N, \cdots, 1/N)$. x^T be the transpose of x. We will then get another vector $M_{PR}x^T$ and so on for $M_{PR}^k x^T$. We know that $M_{PR}^k x^T$ will converge to a vector when k is big enough. Then

$$M_{PR}^{k+1} x^T = M_{PR}[(M_{PR})^k x^T] = (M_{PR})^k x^T.$$

Therefore, in such a case, $y = (M_{PR})^k x^T$ is the eigenvector of M_{PR}. After we add the dumping factor to the matrix, we have

$$G = \begin{bmatrix} \frac{(1-d)}{N} \\ \cdots \\ \frac{(1-d)}{N} \end{bmatrix} + d \cdot M_{PR}. \tag{2.6}$$

The eigenvector of G will be approximately

$$\begin{bmatrix} PR(A_1) \\ \cdots \\ PR(A_N) \end{bmatrix}.$$

2.7.2 Matching

Matching usually means searching for an object that may contain several elements in a set. The set may hold a structure, meaning that the elements in the set have relations and connections. Finding a substring in a DNA sequence is a good example. More profound research in finding a protein structure is a geometric problem.

The most useful matching technique today is still string matching, it is to find a certain substring from a big text file. A fast algorithm called the KMP algorithm is often used [12].

If the set is a random set, then the search will be trivial since we can only compare the elements from the set one by one. When the set contains a structure, such as an order, the search may use properties of the structure. In space, the set could be a topological structure such as components, a geometric structure such as distance metrics, or an algebraic relation such as rules.

Matching could also mean that we can find a partial or best match. For instance, finger print matching [20, 39].

2.7.3 High Dimensional Search

Recent research shows great interest in high dimensional search. This search is usually related to the nearest neighbor method. The fast algorithm is based on preexisting data structures. To find a new data point in a set of points, usually in the form of an n-dimensional vector, we want to find the point that is closest to the new point. The R-tree data structure was previously implemented in order to perform a fast search. This application is important in wireless networking. For instance, we have a number of wireless tower stations that cover cellular phone user communication. When a new phone user joins in the area, we need to locate the new phone and appoint a station to communicate with it. To search the cell phone without calculating its distance from all the stations, we will need a special data structure. The most popular today is the R-tree.

Another hot research topic due to the need of BigData applications is called subspace recovery. The following is an interesting question that has recently received much attention:

Thinking about a simple problem, there are 100 sample points on a 2D plane. We want to find a line that contains most of these sample points. Is this problem NP hard? The decision problem can be described as follows: Is there a line that contains at least 50 points? What about a curve instead of a line? This problem is highly related to manifold learning where we try to determine a cloud point set that represents a manifold such as a curve. The data points are not only 50–100, but they are also 50–100 GB. How do we use cloud computing technology to solve this problem?

The general version of this problem is called robust subspace recovery, which relates to dimension reduction [14, 18]: Given a collection of m points in R^n, if many but not necessarily all of these points are contained in a d-dimensional subspace T, can we find m? The points contained in T are called inliers and the remaining points are outliers. This problem has received considerable attention in computer science and statistics [14, 35, 38, 41]. However, efficient algorithms from computer science are not robust to adversarial outliers, and the estimators from robust statistics are difficult to compute in higher dimensions [2].

The problem is finding a T dimensional space that contains most of the points or a given ratio of inner points. How do we determine T in the fastest way? Does T have some sort of boundary?

Geometric search, especially high dimensional search, is a hot topic related to data science, along with incomplete data search, which is related to artificial intelligence.

2.8 Remarks: Relationship Among Data Science, Database, Networking, and Artificial Intelligence

In short, data science deals with massive data sets in different formats. The data sets are stored in databases or obtained over the Internet. Data science mainly uses statistical and artificial intelligence methods today. Fast algorithms and parallel and distributed algorithms are important in applications. Today, in terms of cloud computing technology, some software systems such as Hadoop and Spark are available to use [15, 22, 28]. These are essentially developed based on the principle of distributed computing. In the following chapters, we will study detailed methods for incomplete relations among data sets and machine learning. We will also discuss topological methods [5, 6, 10, 19] and advanced statistical methods, in Part II of this book, and other advanced topics and problems in Part III.

References

1. A.V. Aho, J.E. Hopcroft, J.D. Ullman, *The Design and Analysis of Computer Algorithms* (Addison-Wesley, Boston, 1974)
2. M. Belkin, P. Niyogi, Laplacian eigenmaps for dimensionality reduction and data representation. Neural Comput. **15**(6), 1373–1396 (2003)
3. B. Bollobas, *Random Graphs* (Academic, London, 1985)
4. S. Brin, L. Page, The anatomy of a large-scale hypertextual Web search engine. Comput. Netw. ISDN Syst. **30**, 107–117 (1998)
5. G. Carlsson, A. Zomorodian, Theory of multidimensional persistence. Discret. Comput. Geom. **42**(1), 71–93 (2009)
6. L. Cayton, Algorithms for manifold learning. Technical Report CS2008-0923, UCSD (2005)
7. L. Chen, *Discrete Surfaces and Manifolds: A Theory of Digital-Discrete Geometry and Topology* (SP Computing, Rockville, 2004)
8. L. Chen, *Digital Functions and Data Reconstruction* (Springer, New York, 2013)

9. L.M. Chen, Digital and Discrete Geometry: Theory and Algorithms, NY Springer (2014)
10. L. Chen, Y. Rong, Digital topological method for computing genus and the Betti numbers. Topol. Appl. **157**(12), 1931–1936 (2010)
11. L. Chen, H. Zhu, W. Cui, Very fast region-connected segmentation for spatial data: case study, in *IEEE Conference on System, Man, and Cybernetics* (2006). pp. 4001–4005
12. T.H. Cormen, C.E. Leiserson, R.L. Rivest, *Introduction to Algorithms* (MIT Press, Cambridge, 1993)
13. M. Demirbas, H. Ferhatosmanoglu, Peer-to-peer spatial queries in sensor networks, in *Third International Conference on Peer-to-Peer Computing*, Linkoping (2003)
14. D.L. Donoho, C. Grimes. Hessian Eigenmaps: new locally linear embedding techniques for high-dimensional data. Technical Report TR-2003-08, Department of Statistics, Stanford University (2003)
15. Afrati Foto and Jeffrey Ullman, (2009) Optimizing Joins in a Map-Reduce Environment. Technical Report. Stanford InfoLab. (2009)
16. K. Fukunaga, L.D. Hostetler, The estimation of the gradient of a density function, with applications in pattern recognition. IEEE Trans. Inf. Theory **21**(1), 32–40 (1975)
17. F. Harary, *Graph Theory* (Addison-Wesley, Reading, 1969)
18. M. Hardt, A. Moitra. Algorithms and hardness for robust subspace recovery, in *COLT*, pp. 354–375 (2013)
19. R. Ghrist, Barcodes: the persistent topology of data. Bull. Am. Math. Soc. **45**(1), 61–75 (2008)
20. R.C. Gonzalez, R. Wood, *Digital Image Processing* (Addison-Wesley, Reading, 1993)
21. J. Goodman, J. O'Rourke, *Handbook of Discrete and Computational Geometry* (CRC, Boca Raton, 1997)
22. J. Han, M. Kamber, *Data Mining: Concepts and Techniques* (Morgan Kaufmann, San Francisco, 2001)
23. H. Homann, *Implementation of a 3D Thinning Algorithm* (Oxford University, Wolf-son Medical Vision Lab., Oxford, 2007)
24. F.V. Jensen, *Bayesian Networks and Decision Graphs* (Springer, New York, 2001)
25. T. Kanungo, D.M. Mount, N. Netanyahu, C. Piatko, R. Silverman, A.Y. Wu, A local search approximation algorithm for k-means clustering. Comput. Geom. Theory Appl. **28**, 89–112 (2004)
26. R. Klette, A. Rosenfeld, *Digital Geometry, Geometric Methods for Digital Picture Analysis*. Computer Graphics and Geometric Modeling (Morgan Kaufmann, San Francisco, 2004)
27. T.C. Lee, R.L. Kashyap, C.N. Chu, Building skeleton models via 3-D medial surface/axis thinning algorithms. Comput. Vis. Graphics Image Process. **56**(6), 462–478 (1994)
28. T.M. Mitchell, *Machine Learning* (McGraw Hill, New York, 1997)
29. L. Page, S. Brin, R. Motwani, T. Winograd, The PageRank citation ranking: bringing order to the web. Technical Report. Stanford InfoLab. (1999)
30. T. Pavilidis, *Algorithms for Graphics and Image Processing* (Computer Science Press, Rockville, 1982)
31. W.H. Press, et al., *Numerical Recipes in C: The Art of Scientific Computing*, 2nd edn. (Cambridge University Press, Cambridge, 1993)
32. X. Ren, J. Malik, Learning a classification model for segmentation, in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 10–17 (2003)
33. A. Rosenfeld, and A.C. Kak, *Digital Picture Processing*, 2nd edn. (Academic, New York, 1982)
34. S. Russell, P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd edn. (Pearson, Boston, 2009)
35. L.K. Saul, S.T. Roweis, Think globally, fit locally: unsupervised learning of low dimensional manifolds. J. Mach. Learn. Res. **4**, 119–155 (2003)
36. H. Samet, *The Design and Analysis of Spatial Data Structures* (Addison Wesley, Reading, 1990)
37. J. Shi, J. Malik, Normalized cuts and image segmentation. IEEE Trans. Pattern Anal. Mach. Intell. **22**(8), 888–905 (2000)

38. J.B. Tenenbaum, V. de Silva, J.C. Langford, A global geometric framework for nonlinear dimensionality reduction. Science **290**, 2319–2323 (2000)
39. S. Theodoridis, K. Koutroumbas, *Pattern Recognition* (Academic, Boston, 2003)
40. D.P. Williamson, D.B. Shmoys, *The Design of Approximation Algorithms* (Cambridge University Press, Cambridge, 2011)
41. Z. Zhang, H. Zha, Principal manifolds and nonlinear dimension reduction via local tangent space alignment. SIAM J. Sci. Comput. **26**(1), 313–338 (2005)
42. B. Zheng, W.-C. Lee, D.L. Lee. Spatial queries in wireless broadcast systems. Wirel. Netw. **10**(6), 723–736 (2004)

http://www.springer.com/978-3-319-25125-7

Mathematical Problems in Data Science
Theoretical and Practical Methods
Chen, L.M.; Su, Z.; Jiang, B.
2015, XV, 213 p. 64 illus., 42 illus. in color., Hardcover
ISBN: 978-3-319-25125-7