Community Experience Distilled

Learning Data Mining with R

Develop key skills and techniques with R to create and customize data mining algorithms

Bater Makhabel

[PACKT] open source*
community experience distilled
PUBLISHING

Learning Data Mining with R

Develop key skills and techniques with R to create and customize data mining algorithms

Bater Makhabel

[PACKT] PUBLISHING open source
community experience distilled

BIRMINGHAM - MUMBAI

Learning Data Mining with R

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: January 2015

Production reference: 1250115

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78398-210-3

www.packtpub.com

Credits

Author
Bater Makhabel

Reviewers
Jason H.D. Cho
Gururaghav Gopal
Vibhav Kamath
Hasan Kurban

Commissioning Editor
Akram Husain

Acquisition Editors
Richard Gall
Owen Roberts

Content Development Editor
Govindan Kurumangattu

Technical Editors
Tanvi Bhatt
Jalasha D'costa
Pooja Nair
Siddhi Rane

Copy Editors
Roshni Banerjee
Karuna Narayanan
Vikrant Phadkay

Project Coordinator
Shipra Chawhan

Proofreaders
Ameesha Green
Sandra Hopper
Clyde Jenkins

Indexer
Mariammal Chettiyar

Graphics
Valentina D'silva
Disha Haria
Abhinash Sahu

Production Coordinator
Arvindkumar Gupta

Cover Work
Arvindkumar Gupta

About the Author

Bater Makhabel (LinkedIn: *BATERMJ* and GitHub: *BATERMJ*) is a system architect living across Beijing, Shanghai, and Urumqi in China. He received his master's and bachelor's degrees in computer science and technology from Tsinghua University between the years 1995 and 2002. He has extensive experience in machine learning, data mining, natural language processing (NLP), distributed systems, embedded systems, the Web, mobile, algorithms, and applied mathematics and statistics. He has worked for clients such as CA Technologies, META4ALL, and EDA (a subcompany of DFR). He also has experience in setting up start-ups in China.

Bater has been balancing a life of creativity between the edge of computer sciences and human cultures. For the past 12 years, he has gained experience in various culture creations by applying various cutting-edge computer technologies, one being a human-machine interface that is used to communicate with computer systems in the Kazakh language. He has previously collaborated with other writers in his fields too, but *Learning Data Mining with R* is his first official effort.

Acknowledgments

I would like to thank my wife, Zurypa Dawletkan, and my son, Bakhtiyar. They supported me and spent weekends and nights to make this book possible.

I would like to thank Luke Presland. He gave me the opportunity to write this book. A big thank you to Rebecca Pedley and Govindan K; your help on the book was great. Thanks to Jalasha D'costa and the other technical editors and the team for their hard work on the publication version of this book, to make it look good. Also, thanks to all the acquisition editors and technical reviewers.

I also would like to thank my brother, Dr. Bolat Makhabel (LinkedIn: *BOLATMJ*), for providing me with the cover image for this book. He is from a medical science background. The name of the plant in the image is Echinops (the botanical Latin name), Lahsa in Kazakhstan, and Khonrawbas (or Koktiken) in China. This plant is used in the traditional Kazakh medicine and is a part of his research as well.

Although most of my professional knowledge comes from continual practice, its roots are in the firm foundation set up by my university, Tsinghua University, and my teachers, Prof. Dai Meie, Prof. Zhao Yannan, Prof. Wang Jiaqin, Prof. Ju Yuma, and many others. Their spirit is still an inspiration for me to pursue my work in the field of computer science and technology.

I'd like to express my thanks to my wife's parents, Dawletkan Kobegen and Burux Takay, for helping us by taking care of my son.

Lastly, I would also like to express my greatest respect to my sister, Aynur Makhabel, and my brother-in-law, Akimjan Xaymardan, for their valuable virtue.

About the Reviewers

Jason H.D. Cho has received an M.S. degree from University of Illinois at Urbana-Champaign in computer science, and he is currently pursuing his PhD there. He is particularly interested in applying natural language processing and big data to solve medical informatics problem. In particular, he wishes to characterize what patients are concerned about in terms of their health needs in social media. He has also led a group of mentees who have been nominated for one of the top 10 teams in a national primary health care competition (CIMIT). Jason has also reviewed paper submissions in both natural language processing and big data research field.

Gururaghav Gopal is presently working in Paterson securities as a Quant developer, trader, and consultant. Previously, he worked as a data science consultant and was associated with the e-commerce industry. He has also taught graduate and postgraduate students of VIT University, Vellore, India in the field of pattern recognition. He has been a research associate in several research institutes, including IFMR and NAL.

Gururaghav completed his bachelor's degree in electrical and electronics engineering and master's degree in computer science and engineering. He completed his course from IFMR in financial engineering and risk management, and since then, he has been associated with the finance sector. He has won a few awards and has a few international publications to his name.

He is interested in programming, teaching, and consulting. During his free time, he listens to music.

Vibhav Kamath holds a master's degree in industrial engineering and operations research from Indian Institute of Technology, Bombay, and a bachelor's degree in electronics engineering from College of Engineering, Pune. During his postgraduation, he was intrigued with algorithms and mathematical modeling. He has been involved in analytics since then. Vibhav is currently based in Bangalore, and works for an IT services firm. As a part of his job, he has developed statistical/mathematical models based on techniques such as optimization and linear regression, using the R programming language. He has also worked as a reviewer for two other books on R, *R Graphs Cookbook Second Edition* and *Social Media Mining with R*, both by *Packt Publishing*. In the past, he has also handled data visualization and created dashboards for a leading global bank, using platforms such as SAS, SQL, and Excel/VBA.

In the past, Vibhav has worked on areas such as discrete event simulation and speech processing (both on MATLAB) as part of his academics. He has also worked on robotics in the past, and has built a robot that navigates through a maze, called Micromouse. Apart from analytics and programming, Vibhav has interests in reading and likes both fiction and nonfiction books. He plays table tennis in his free time, follows cricket and tennis, and likes solving puzzles (*Su-doku* and *Kakuro*) when really bored. You can get in touch with him at `vibhav.kamath@hotmail.com` or on LinkedIn at `in.linkedin.com/in/vibhavkamath`.

Hasan Kurban received his master's degree in computer science from Indiana University, Bloomington in 2012. He is currently a PhD student in the School of Informatics and Computing at Indiana University, Bloomington, majoring in computer science and minoring in statistics. His research is focused on data mining, machine learning, and statistics.

I dedicate this book to my parents, Makhabel.Janabel and Gulsahira.Muhatay.
They always love, support, and believe in me.

– Bater Makhabel

Table of Contents

Preface

The necessity to handle many complex statistical analysis projects is hitting statisticians and analysts across the globe. Since there is an increasing interest in data analysis, R offers a free and open source environment that is perfect for both learning and deploying predictive modeling solutions in the real world. With its constantly growing community and plethora of packages, R offers functionality to deal with a truly vast array of problems.

It's been decades since the R programming language was born, and it has become eminent and well known not only within the community of scientists but also in the wider community of developers. It has grown into a powerful tool to help developers produce efficient and consistent source code for data-related tasks. The R development team and independent contributors have created good documentation, so getting started with R programming isn't that hard.

To go further, you can use packages from the official R website. If you want to continually improve your level of expertise, you might read through a set of books that have been published in last couple of years. You should always bear in mind that creating high-level, secure, and internationally compliant code is more complex than the first application created in the beginning.

This book is designed to help you deal with an array of problems that you may encounter during complex statistical projects, which can be difficult. Topics in this book will include learning how to manipulate data with R using code snippets, mining frequent patterns, association, and correlations while working with R programs. This book will also provide for those with only a basic knowledge of R the skills and knowledge to successfully create and customize the most popular data mining algorithms. This will help overcome difficulties encountered and will ensure the most effective use of the R programming language on data mining algorithm development through its rich set of publicly available packages.

Each chapter of this book is intended to stand on its own, so feel free to jump to any chapter where you feel you need to get more in-depth knowledge about a particular topic. If you feel you missed something major, go back and read the earlier chapters. They are constructed in a way to grow your knowledge piece by piece.

Discover how to write code for various predication models, stream data, and time-series data. You will also be introduced to solutions based on the MapReduce algorithm. You will finish this book feeling confident in the ability that you know which data mining algorithm to apply in which situation.

I enjoy working with the R programming language for versatile data mining tasks developments and researches, and I am really happy to share my enthusiasm and expertise with you to help you make use of the language more effectively and comfortably use data mining algorithm developments and applications.

What this book covers

Chapter 1, *Warming Up*, gives you the overview of data mining, the relation of data mining to machine learning, and statistics. It illustrates basic data mining terms such as data definition and preprocessing.

Chapter 2, *Mining Frequent Patterns, Associations, and Correlations*, contains advanced and interesting algorithms required to learn mining frequent patterns, association rules, and correlation rules when working with R programs.

Chapter 3, *Classification*, helps you learn the classic classification algorithms written in the R language, covering various classification algorithms for different types of datasets.

Chapter 4, *Advanced Classification*, teaches you more classification algorithms, such as the Bayesian Belief Network, SVM, and k-Nearest Neighbors algorithm.

Chapter 5, *Cluster Analysis*, helps you learn how to implement the popular and classic algorithms for clustering, such as k-means, CLARA, and spectral algorithms.

Chapter 6, *Advanced Cluster Analysis*, shows the implementation of advanced algorithms for clustering that are related to hot topics in current industries, including EM, CLIQUE, DBSCAN, and so on.

Chapter 7, *Outlier Detection*, demonstrates the classic and popular algorithms used to detect outliers in real-world cases.

Chapter 8, *Mining Stream, Time-series, and Sequence Data*, explains these three hot topics with the most popular, classic, and top-ranking algorithms.

Chapter 9, *Graph Mining and Network Analysis*, shows you the overview of graphs and social mining algorithms, along with other interesting topics.

Chapter 10, *Mining Text and Web Data*, helps you learn the popular algorithms applied in domains with interesting applications.

Appendix, *Algorithms and Data Structures*, contains a list of algorithms and data structures to help you on your data mining journey.

What you need for this book

Any modern PC with Windows, Linux, or Mac OS should be sufficient to run the code samples given in this book. All of the software used in the book is open source and freely available on the Web, at `http://www.r-project.org/`.

Who this book is for

This book is intended for budding data scientists, quantitative analysts, and software engineers with only basic exposure to R and statistics. This book assumes familiarity with only the very basics of R, such as the main data types, simple functions, and how to move data around. No prior experience with data mining packages is necessary. However, you should have basic understanding of data mining concepts and processes.

Even if you are brand new to data mining, you will be able to master both the basic and the advanced implementations of data mining algorithms. You will learn how to select and apply the appropriate algorithms from various data mining algorithms to some specific datasets out of most of the datasets available for the real world.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and explanations of their meanings.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "We can include other contexts through the use of the `include` directive."

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Clicking on the **Next** button moves you to the next screen."

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you. You can also find the code files for this book at `https://github.com/batermj/learning-data-mining-with-r`.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

1
Warming Up

In this chapter, you will learn basic data mining terms such as data definition, preprocessing, and so on.

The most important data mining algorithms will be illustrated with R to help you grasp the principles quickly, including but not limited to, classification, clustering, and outlier detection. Before diving right into data mining, let's have a look at the topics we'll cover:

- Data mining
- Social network mining
- Text mining
- Web data mining
- Why R
- Statistics
- Machine learning
- Data attributes and description
- Data measuring
- Data cleaning
- Data integration
- Data reduction
- Data transformation and discretization
- Visualization of results

In the history of humankind, the results of data from every aspect is extensive, for example websites, social networks by user's e-mail or name or account, search terms, locations on map, companies, IP addresses, books, films, music, and products.

Data mining techniques can be applied to any kind of old or emerging data; each data type can be best dealt with using certain, but not all, techniques. In other words, the data mining techniques are constrained by data type, size of the dataset, context of the tasks applied, and so on. Every dataset has its own appropriate data mining solutions.

New data mining techniques always need to be researched along with new data types once the old techniques cannot be applied to it or if the new data type cannot be transformed onto the traditional data types. The evolution of stream mining algorithms applied to Twitter's huge source set is one typical example. The graph mining algorithms developed for social networks is another example.

The most popular and basic forms of data are from databases, data warehouses, ordered/sequence data, graph data, text data, and so on. In other words, they are federated data, high dimensional data, longitudinal data, streaming data, web data, numeric, categorical, or text data.

Big data

Big data is large amount of data that does not fit in the memory of a single machine. In other words, the size of data itself becomes a part of the issue when studying it. Besides volume, two other major characteristics of big data are variety and velocity; these are the famous three Vs of big data. Velocity means data process rate or how fast the data is being processed. Variety denotes various data source types. Noises arise more frequently in big data source sets and affect the mining results, which require efficient data preprocessing algorithms.

As a result, distributed filesystems are used as tools for successful implementation of parallel algorithms on large amounts of data; it is a certainty that we will get even more data with each passing second. Data analytics and visualization techniques are the primary factors of the data mining tasks related to massive data. The characteristics of massive data appeal to many new data mining technique-related platforms, one of which is RHadoop. We'll be describing this in a later section.

Some data types that are important to big data are as follows:

- The data from the camera video, which includes more metadata for analysis to expedite crime investigations, enhanced retail analysis, military intelligence, and so on.
- The second data type is from embedded sensors, such as medical sensors, to monitor any potential outbreaks of virus.

- The third data type is from entertainment, information freely published through social media by anyone.
- The last data type is consumer images, aggregated from social medias, and tagging on these like images are important.

Here is a table illustrating the history of data size growth. It shows that information will be more than double every two years, changing the way researchers or companies manage and extract value through data mining techniques from data, revealing new data mining studies.

Year	**Data Sizes**	**Comments**
N/A		1 MB (Megabyte) = 2^{20}. The human brain holds about 200 MB of information.
N/A		1 PB (Petabyte) = 2^{50}. It is similar to the size of 3 years' observation data for Earth by NASA and is equivalent of 70.8 times the books in America's Library of Congress.
1999	1 EB	1 EB (Exabyte) = 2^{60}. The world produced 1.5 EB of unique information.
2007	281 EB	The world produced about 281 Exabyte of unique information.
2011	1.8 ZB	1 ZB (Zetabyte)= 2^{70}. This is all data gathered by human beings in 2011.
Very soon		1 YB(Yottabytes)= 2^{80}.

Scalability and efficiency

Efficiency, scalability, performance, optimization, and the ability to perform in real time are important issues for almost any algorithms, and it is the same for data mining. There are always necessary metrics or benchmark factors of data mining algorithms.

As the amount of data continues to grow, keeping data mining algorithms effective and scalable is necessary to effectively extract information from massive datasets in many data repositories or data streams.

The storage of data from a single machine to wide distribution, the huge size of many datasets, and the computational complexity of the data mining methods are all factors that drive the development of parallel and distributed data-intensive mining algorithms.

Data source

Data serves as the input for the data mining system and data repositories are important. In an enterprise environment, database and logfiles are common sources. In web data mining, web pages are the source of data. The data that continuously fetched various sensors are also a typical data source.

> Here are some free online data sources particularly helpful to learn about data mining:
>
> - **Frequent Itemset Mining Dataset Repository**: A repository with datasets for methods to find frequent itemsets (`http://fimi.ua.ac.be/data/`).
> - **UCI Machine Learning Repository**: This is a collection of dataset, suitable for classification tasks (`http://archive.ics.uci.edu/ml/`).
> - **The Data and Story Library at statlib**: DASL (pronounced "dazzle") is an online library of data files and stories that illustrate the use of basic statistics methods. We hope to provide data from a wide variety of topics so that statistics teachers can find real-world examples that will be interesting to their students. Use DASL's powerful search engine to locate the story or data file of interest. (`http://lib.stat.cmu.edu/DASL/`)
> - **WordNet**: This is a lexical database for English (`http://wordnet.princeton.edu`)

Data mining

Data mining is the discovery of a *model* in data; it's also called exploratory data analysis, and discovers useful, valid, unexpected, and understandable knowledge from the data. Some goals are shared with other sciences, such as statistics, artificial intelligence, machine learning, and pattern recognition. Data mining has been frequently treated as an algorithmic problem in most cases. Clustering, classification, association rule learning, anomaly detection, regression, and summarization are all part of the tasks belonging to data mining.

The data mining methods can be summarized into two main categories of data mining problems: feature extraction and summarization.

Feature extraction

This is to extract the most prominent features of the data and ignore the rest. Here are some examples:

- **Frequent itemsets**: This model makes sense for data that consists of *baskets* of small sets of items.
- **Similar items**: Sometimes your data looks like a collection of sets and the objective is to find pairs of sets that have a relatively large fraction of their elements in common. It's a fundamental problem of data mining.

Summarization

The target is to summarize the dataset succinctly and approximately, such as clustering, which is the process of examining a collection of *points* (data) and grouping the points into *clusters* according to some measure. The goal is that points in the same cluster have a small distance from one another, while points in different clusters are at a large distance from one another.

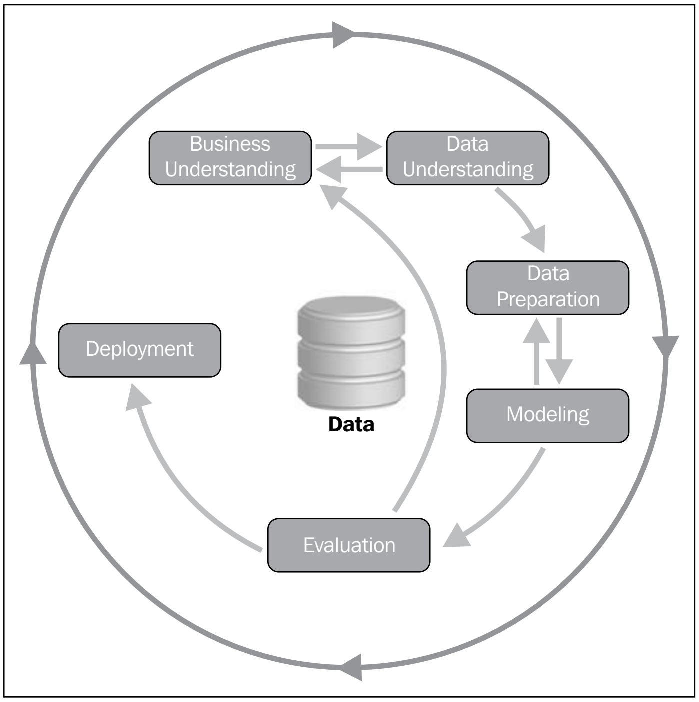
The data mining process

There are two popular processes to define the data mining process in different perspectives, and the more widely adopted one is CRISP-DM:

- **Cross-Industry Standard Process for Data Mining** (**CRISP-DM**)
- **Sample, Explore, Modify, Model, Assess** (**SEMMA**), which was developed by the SAS Institute, USA

CRISP-DM

There are six phases in this process that are shown in the following figure; it is not rigid, but often has a great deal of backtracking:



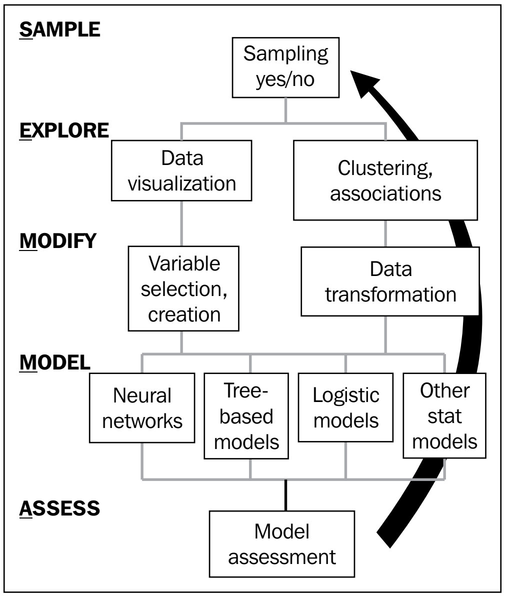
Let's look at the phases in detail:

- **Business understanding**: This task includes determining business objectives, assessing the current situation, establishing data mining goals, and developing a plan.
- **Data understanding**: This task evaluates data requirements and includes initial data collection, data description, data exploration, and the verification of data quality.
- **Data preparation**: Once available, data resources are identified in the last step. Then, the data needs to be selected, cleaned, and then built into the desired form and format.

- **Modeling**: Visualization and cluster analysis are useful for initial analysis. The initial association rules can be developed by applying tools such as generalized rule induction. This is a data mining technique to discover knowledge represented as rules to illustrate the data in the view of causal relationship between conditional factors and a given decision/outcome. The models appropriate to the data type can also be applied.
- **Evaluation** :The results should be evaluated in the context specified by the business objectives in the first step. This leads to the identification of new needs and in turn reverts to the prior phases in most cases.
- **Deployment**: Data mining can be used to both verify previously held hypotheses or for knowledge.

SEMMA

Here is an overview of the process for SEMMA:



Let's look at these processes in detail:

- **Sample**: In this step, a portion of a large dataset is extracted
- **Explore**: To gain a better understanding of the dataset, unanticipated trends and anomalies are searched in this step
- **Modify**: The variables are created, selected, and transformed to focus on the model construction process

- **Model**: A variable combination of models is searched to predict a desired outcome
- **Assess**: The findings from the data mining process are evaluated by its usefulness and reliability

Social network mining

As we mentioned before, data mining finds a *model* on data and the mining of social network finds the *model* on graph data in which the social network is represented.

Social network mining is one application of web data mining; the popular applications are social sciences and bibliometry, PageRank and HITS, shortcomings of the coarse-grained graph model, enhanced models and techniques, evaluation of topic distillation, and measuring and modeling the Web.

Social network

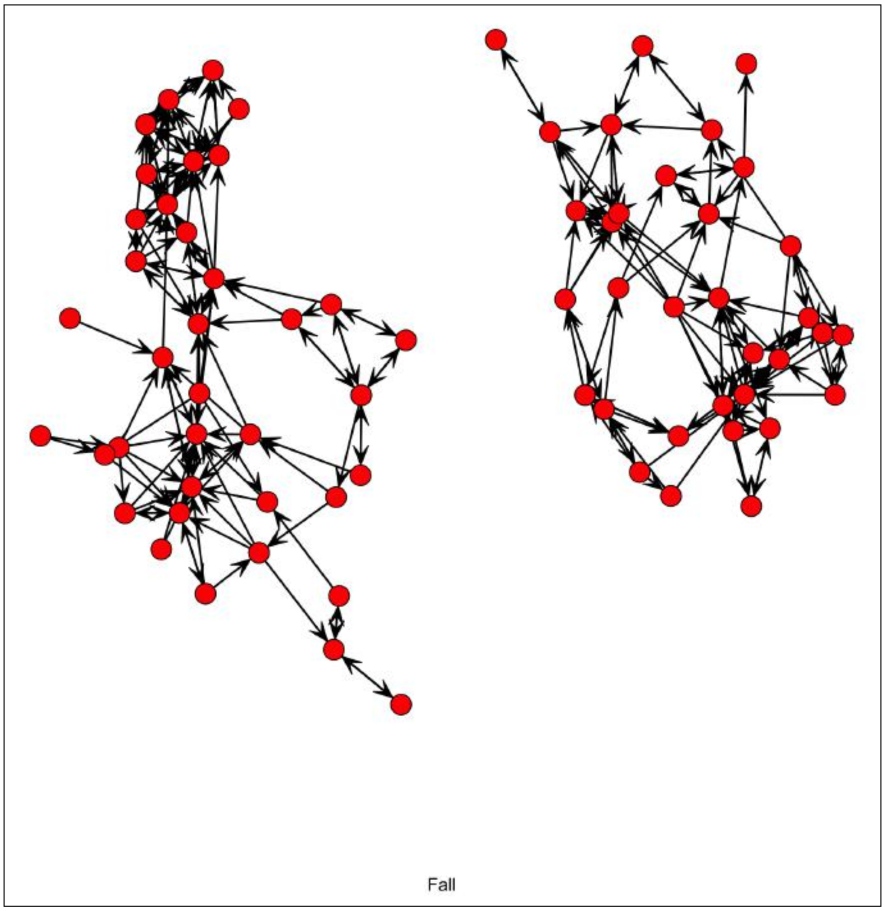
When it comes to the discussion of social networks, you will think of Facebook, Google+, LinkedIn, and so on. The essential characteristics of a social network are as follows:

- There is a collection of entities that participate in the network. Typically, these entities are people, but they could be something else entirely.
- There is at least one relationship between the entities of the network. On Facebook, this relationship is called friends. Sometimes, the relationship is all-or-nothing; two people are either friends or they are not. However, in other examples of social networks, the relationship has a degree. This degree could be discrete, for example, friends, family, acquaintances, or none as in Google+. It could be a real number; an example would be the fraction of the average day that two people spend talking to each other.
- There is an assumption of nonrandomness or locality. This condition is the hardest to formalize, but the intuition is that relationships tend to cluster. That is, if entity A is related to both B and C, then there is a higher probability than average that B and C are related.

Here are some varieties of social networks:

- **Telephone networks**: The nodes in this network are phone numbers and represent individuals
- **E-mail networks**: The nodes represent e-mail addresses, which represent individuals
- **Collaboration networks**: The nodes here represent individuals who published research papers; the edge connecting two nodes represent two individuals who published one or more papers jointly

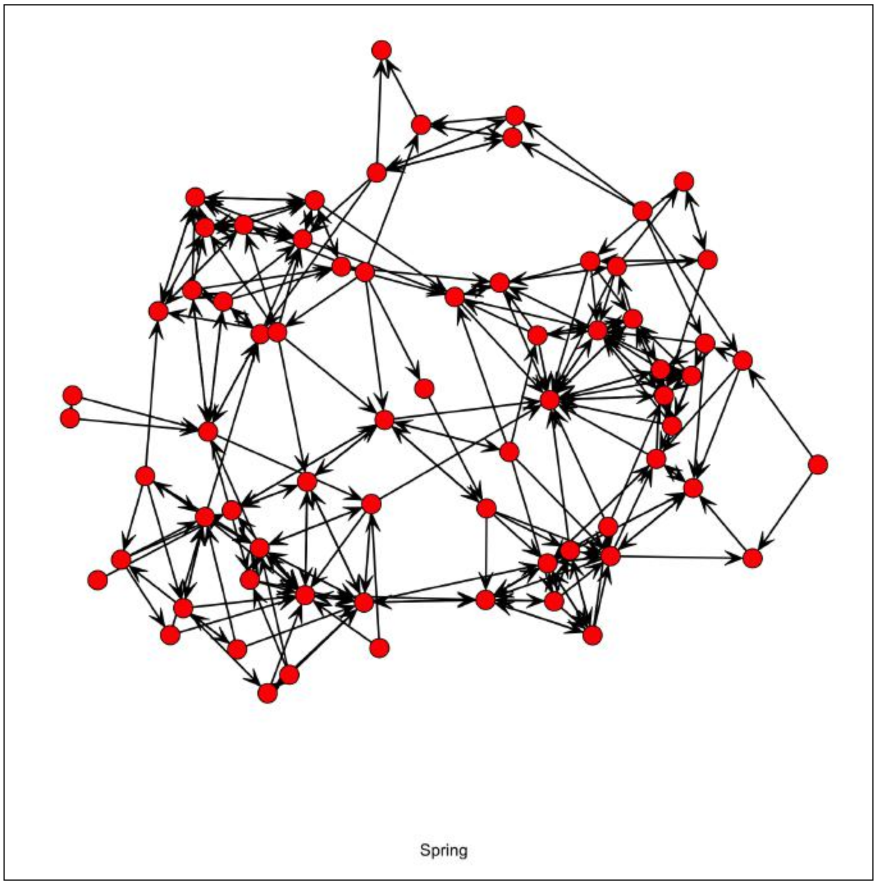
Social networks are modeled as undirected graphs. The entities are the nodes, and an edge connects two nodes if the nodes are related by the relationship that characterizes the network. If there is a degree associated with the relationship, this degree is represented by labeling the edges.



Downloading the example code

You can download the example code files from your account at `http://www.packtpub.com` for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Here is an example in which Coleman's High School Friendship Data from the sna R package is used for analysis. The data is from a research on friendship ties between 73 boys in a high school in one chosen academic year; reported ties for all informants are provided for two time points (fall and spring). The dataset's name is `coleman`, which is an array type in R language. The node denotes a specific student and the line represents the tie between two students.



Text mining

Text mining is based on the data of text, concerned with exacting relevant information from large natural language text, and searching for interesting relationships, syntactical correlation, or semantic association between the extracted entities or terms. It is also defined as automatic or semiautomatic processing of text. The related algorithms include text clustering, text classification, natural language processing, and web mining.

One of the characteristics of text mining is text mixed with numbers, or in other point of view, the hybrid data type contained in the source dataset. The text is usually a collection of unstructured documents, which will be preprocessed and transformed into a numerical and structured representation. After the transformation, most of the data mining algorithms can be applied with good effects.

The process of text mining is described as follows:

- Text mining starts from preparing the text corpus, which are reports, letters and so forth
- The second step is to build a semistructured text database that is based on the text corpus
- The third step is to build a term-document matrix in which the term frequency is included
- The final result is further analysis, such as text analysis, semantic analysis, information retrieval, and information summarization

Information retrieval and text mining

Information retrieval is to help users find information, most commonly associated with online documents. It focuses on the acquisition, organization, storage, retrieval, and distribution for information. The task of **Information Retrieval** (**IR**) is to retrieve relevant documents in response to a query. The fundamental technique of IR is measuring similarity. Key steps in IR are as follows:

- Specify a query. The following are some of the types of queries:
    - **Keyword query**: This is expressed by a list of keywords to find documents that contain at least one keyword
    - **Boolean query**: This is constructed with Boolean operators and keywords
    - **Phrase query**: This is a query that consists of a sequence of words that makes up a phrase

    - **Proximity query**: This is a downgrade version of the phrase queries and can be a combination of keywords and phrases
    - **Full document query**: This query is a full document to find other documents similar to the query document
    - **Natural language questions**: This query helps to express users' requirements as a natural language question

- Search the document collection.
- Return the subset of relevant documents.

Mining text for prediction

Prediction of results from text is just as ambitious as predicting numerical data mining and has similar problems associated with numerical classification. It is generally a classification issue.

Prediction from text needs prior experience, from the sample, to learn how to draw a prediction on new documents. Once text is transformed into numeric data, prediction methods can be applied.

Web data mining

Web mining aims to discover useful information or knowledge from the web hyperlink structure, page, and usage data. The Web is one of the biggest data sources to serve as the input for data mining applications.

Web data mining is based on IR, **machine learning** (**ML**), statistics, pattern recognition, and data mining. Web mining is not purely a data mining problem because of the heterogeneous and semistructured or unstructured web data, although many data mining approaches can be applied to it.

Web mining tasks can be defined into at least three types:

- **Web structure mining**: This helps to find useful information or valuable structural summary about sites and pages from hyperlinks
- **Web content mining**: This helps to mine useful information from web page contents
- **Web usage mining**: This helps to discover user access patterns from web logs to detect intrusion, fraud, and attempted break-in

The algorithms applied to web data mining are originated from classical data mining algorithms. They share many similarities, such as the mining process; however, differences exist too. The characteristics of web data mining makes it different from data mining for the following reasons:

- The data is unstructured
- The information of the Web keeps changing and the amount of data keeps growing
- Any data type is available on the Web, such as structured and unstructured data
- Heterogeneous information is on the web; redundant pages are present too
- Vast amounts of information on the web is linked
- The data is noisy

Web data mining differentiates from data mining by the huge dynamic volume of source dataset, a big variety of data format, and so on. The most popular data mining tasks related to the Web are as follows:

- **Information extraction** (**IE**): The task of IE consists of a couple of steps, tokenization, sentence segmentation, part-of-speech assignment, named entity identification, phrasal parsing, sentential parsing, semantic interpretation, discourse interpretation, template filling, and merging.
- **Natural language processing** (**NLP**): This researches the linguistic characteristics of human-human and human-machine interactive, models of linguistic competence and performance, frameworks to implement process with such models, processes'/models' iterative refinement, and evaluation techniques for the result systems. Classical NLP tasks related to web data mining are tagging, knowledge representation, ontologies, and so on.
- **Question answering**: The goal is to find the answer from a collection of text to questions in natural language format. It can be categorized into slot filling, limited domain, and open domain with bigger difficulties for the latter. One simple example is based on a predefined FAQ to answer queries from customers.
- **Resource discovery**: The popular applications are collecting important pages preferentially; similarity search using link topology, topical locality and focused crawling; and discovering communities.

Why R?

R is a high-quality, cross-platform, flexible, widely used open source, free language for statistics, graphics, mathematics, and data science—created by statisticians for statisticians.

R contains more than 5,000 algorithms and millions of users with domain knowledge worldwide, and it is supported by a vibrant and talented community of contributors. It allows access to both well-established and experimental statistical techniques.

R is a free, open source software environment maintained by R-projects for statistical computing and graphics, and the R source code is available under the terms of the Free Software Foundation's GNU General Public License. R compiles and runs on a wide variety for a variety of platforms, such as UNIX, LINUX, Windows, and Mac OS.

What are the disadvantages of R?

There are three shortages of R:

- One is that it is memory bound, so it requires the entire dataset store in memory (RAM) to achieve high performance, which is also called in-memory analytics.
- Similar to other open source systems, anyone can create and contribute package with strict or less testing. In other words, packages contributing to R communities are bug-prone and need more testing to ensure the quality of codes.
- R seems slow than some other commercial languages.

Fortunately, there are packages available to overcome these problems. There are some solutions that can be categorized as parallelism solutions; the essence here is to spread work across multiple CPUs that overcome the R shortages that were just listed. Good examples include, but are not limited to, RHadoop. You will read more on this topic soon in the following sections. You can download the SNOW add-on package and the Parallel add-on package from **Comprehensive R Archive Network** (**CRAN**).

Statistics

Statistics studies the collection, analysis, interpretation or explanation, and presentation of data. It serves as the foundation of data mining and the relations will be illustrated in the following sections.

Statistics and data mining

Statisticians were the first to use the term data mining. Originally, data mining was a derogatory term referring to attempts to extract information that was not supported by the data. To some extent, data mining constructs statistical models, which is an underlying distribution, used to visualize data.

Data mining has an inherent relationship with statistics; one of the mathematical foundations of data mining is statistics, and many statistics models are used in data mining.

Statistical methods can be used to summarize a collection of data and can also be used to verify data mining results.

Statistics and machine learning

Along with the development of statistics and machine learning, there is a continuum between these two subjects. Statistical tests are used to validate the machine learning models and to evaluate machine learning algorithms. Machine learning techniques are incorporated with standard statistical techniques.

Statistics and R

R is a statistical programming language. It provides a huge amount of statistical functions, which are based on the knowledge of statistics. Many R add-on package contributors come from the field of statistics and use R in their research.

The limitations of statistics on data mining

During the evolution of data mining technologies, due to statistical limits on data mining, one can make errors by trying to extract what really isn't in the data.

Bonferroni's Principle is a statistical theorem otherwise known as **Bonferroni correction**. You can assume that big portions of the items you find are bogus, that is, the items returned by the algorithms dramatically exceed what is assumed.

Machine learning

The data to which a ML algorithm is applied is called a training set, which consists of a set of pairs (x, y), called training examples. The pairs are explained as follows:

- `x`: This is a vector of values, often called the feature vector. Each value, or feature, can be categorical (values are taken from a set of discrete values, such as `{S, M, L}`) or numerical.
- `y`: This is the label, the classification or regression values for `x`.

The objective of the ML process is to discover a function $y = f(x)$ that best predicts the value of `y` associated with each value of `x`. The type of `y` is in principle arbitrary, but there are several common and important cases.

- `y`: This is a real number. The ML problem is called regression.
- `y`: This is a Boolean value true or false, more commonly written as +1 and -1, respectively. In this class, the problem is binary classification.
- `y`: Here this is a member of some finite set. The member of this set can be thought of as *classes*, and each member represents one class. The problem is multiclass classification.
- `y`: This is a member of some potentially infinite set, for example, a parse tree for `x`, which is interpreted as a sentence.

Until now, machine learning has not proved successful in situations where we can describe the goals of the mining more directly. Machine learning and data mining are two different topics, although some algorithms are shared between them—algorithms are shared especially when the goal is to extract information. There are situations where machine learning makes sense. The typical one is when we have idea of what we looking for in the dataset.

Approaches to machine learning

The major classes of algorithms are listed here. Each is distinguished by the function f.

- **Decision tree**: This form of f is a tree and each node of the tree has a function of `x` that determines which child or children the search must proceed for.
- **Perceptron**: These are threshold functions applied to the components of the vector $x = \{x_1, x_2, \ldots x_n\}$. A weight w_i is associated with the ith components, for each *i = 1, 2, … n*, and there is a threshold $\sum_{i=1}^{n} w_i x_i \geq \theta$. The output is +1 if and the output is -1 otherwise.

- **Neural nets**: These are acyclic networks of perceptions, with the outputs of some perceptions used as inputs to others.
- **Instance-based learning**: This uses the entire training set to represent the function f .
- **Support-vector machines**: The result of this class is a classifier that tends to be more accurate on unseen data. The target for class separation denotes as looking for the optimal hyper-plane separating two classes by maximizing the margin between the classes' closest points.

Machine learning architecture

The data aspects of machine learning here means the way data is handled and the way it is used to build the model.

- **Training and testing**: Assuming all the data is suitable for training, separate out a small fraction of the available data as the test set; use the remaining data to build a suitable model or classifier.
- **Batch versus online learning**: The entire training set is available at the beginning of the process for batch mode; the other one is online learning, where the training set arrives in a stream and cannot be revisited after it is processed.
- **Feature selection**: This helps to figure out what features to use as input to the learning algorithm.
- **Creating a training set**: This helps to create the label information that turns data into a training set by hand.

Data attributes and description

An **attribute** is a field representing a certain feature, characteristic, or dimensions of a data object.

In most situations, data can be modeled or represented with a matrix, columns for data attributes, and rows for certain data records in the dataset. For other cases, that data cannot be represented with matrices, such as text, time series, images, audio, video, and so forth. The data can be transformed into a matrix by appropriate methods, such as feature extraction.

The type of data attributes arises from its contexts or domains or semantics, and there are numerical, non-numerical, categorical data types or text data. Two views applied to data attributes and descriptions are widely used in data mining and R. They are as follows:

- **Data in algebraic or geometric view**: The entire dataset can be modeled into a matrix; linear algebraic and abstract algebra plays an important role here.
- **Data in probability view**: The observed data is treated as multidimensional random variables; each numeric attribute is a random variable. The dimension is the data dimension. Irrespective of whether the value is discrete or continuous, the probability theory can be applied here.

To help you learn R more naturally, we shall adopt a geometric, algebraic, and probabilistic view of the data.

Here is a matrix example. The number of columns is determined by *m*, which is the dimensionality of data. The number of rows is determined by *n*, which is the size of dataset.

$$A = \begin{pmatrix} x_{11} & \cdots & x_{1m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nm} \end{pmatrix} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \left(X_1 \cdots X_m \right)$$

Where x_i denotes the *i* row, which is an m-tuple as follows:

$$x_i = (x_{i1} = \left(x_{i1}, \cdots, x_{l} \right)$$

And x_j denotes the *j* column, which is an n-tuple as follows:

$$X_j = (X_{1j} = \left(X_{1j}, \cdots, X_{nj} \right)$$

Numeric attributes

Numerical data is convenient to deal with because it is quantitative and allows arbitrary calculations. The properties of numerical data are the same as integer or float data.

Numeric attributes taken from a finite or countable infinite set of values are called **discrete**, for example a human being's age, which is the integer value starting from 1,150. Other attributes taken from any real values are called **continuous**. There are two main kinds of numeric types:

- **Interval-scaled**: This is the quantitative value, measured on a scale of equal unit, such as the weight of some certain fish in the scale of international metric, such as gram or kilogram.
- **Ratio-scaled**: This value can be computed by ratios between values in addition to differences between values. It is a numeric attribute with an inherent zero-point; hence, we can say a value is a multiple of another value.

Categorical attributes

The values of categorical attributes come from a set-valued domain composed of a set of symbols, such as the size of human costumes that are categorized as `{S, M, L}`. The categorical attributes can be divided into two groups or types:

- **Nominal**: The values in this set are unordered and are not quantitative; only the equality operation makes sense here.
- **Ordinal**: In contrast to the nominal type, the data has an ordered meaning here. The inequality operation is available here in addition to the equality operation.

Data description

The basic description can be used to identify features of data, distinguish noise, or outliers. A couple of basic statistical descriptions are as follows:

- **Measures of central tendency**: This measures the location of middle or center of a data distribution: the mean, median, mode, midrange, and so on.
- **Measure of the dispersion of the data**: This is the range, quartiles, interquartile range, and so on.

Data measuring

Data measuring is used in clustering, outlier detection, and classification. It refers to measures of proximity, similarity, and dissimilarity. The similarity value, a real value, between two tuples or data records ranges from 0 to 1, the higher the value the greater the similarity between tuples. Dissimilarity works in the opposite way; the higher the dissimilarity value, the more dissimilar are the two tuples.

For a dataset, data matrix stores the *n* data tuples in *n* x *m* matrix (*n* tuples and *m* attributes):

$$\begin{pmatrix} x_{11} & \cdots & x_{1m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nm} \end{pmatrix}$$

The dissimilarity matrix stores a collection of proximities available for all *n* tuples in the dataset, often in a *n* x *n* matrix. In the following matrix, $d(i,j)$ means the dissimilarity between two tuples; value 0 for highly similar or near between each other, 1 for completely same, the higher the value, the more dissimilar it is.

$$\begin{pmatrix} 0 \\ d(2,1)0 \\ d(3,1)d(3,2)0 \\ \vdots\vdots\vdots \\ d(n,1)d(n,2)\ldots 0 \end{pmatrix}$$

Most of the time, the dissimilarity and similarity are related concepts. The similarity measure can often be defined using a function; the expression constructed with measures of dissimilarity, and vice versa.

Here is a table with a list of some of the most used measures for different attribute value types:

Attribute value types	**Dissimilarity**
Nominal attributes	The dissimilarity between two tuples can be computed by the following equation: *d (i, j) = (p-m)/p* Where, *p* is the dimension of data and *m* is the number of matches that is in same state.
Ordinal attributes	The treatment of ordinal attributes is similar to that of numeric attributes, but it needs a transformation first before applying the methods.
Interval-scaled	**Euclidean**, **Manhattan**, and **Minkowski** distances are used to calculate the dissimilarity of data tuples.

Data cleaning

Data cleaning is one part of data quality. The aim of **Data Quality** (**DQ**) is to have the following:

- Accuracy (data is recorded correctly)
- Completeness (all relevant data is recorded)
- Uniqueness (no duplicated data record)
- Timeliness (the data is not old)
- Consistency (the data is coherent)

Data cleaning attempts to fill in missing values, smooth out noise while identifying outliers, and correct inconsistencies in the data. Data cleaning is usually an iterative two-step process consisting of discrepancy detection and data transformation.

The process of data mining contains two steps in most situations. They are as follows:

- The first step is to perform audition on the source dataset to find the discrepancy.
- The second step is to choose the transformation to fix (based on the accuracy of the attribute to be modified and the closeness of the new value to the original value). This is followed by applying the transformation to correct the discrepancy.

Missing values

During the process to seize data from all sorts of data sources, there are many cases when some fields are left blank or contain a null value. Good data entry procedures should avoid or minimize the number of missing values or errors. The missing values and defaults are indistinguishable.

If some fields are missing a value, there are a couple of solutions—each with different considerations and shortages and each is applicable within a certain context.

- **Ignore the tuple**: By ignoring the tuple, you cannot make use of the remaining values except the missing one. This method is applicable when the tuple contains several attributes with missing values or the percentage of missing value per attribute doesn't vary considerably.
- **Filling the missing value manually**: This is not applicable for large datasets.
- **Use a global constant to fill the value**: Applying the value to fill the missing value will misguide the mining process, and is not foolproof.

- **Use a measure for a central tendency for the attribute to fill the missing value**: The measures of central tendency can be used for symmetric data distribution.
- **Use the attribute mean or median**: Use the attribute mean or median for all samples belonging to the same class as the given tuple.
- **Use the most probable value to fill the missing value**: The missing data can be filled with data determined with regression, inference-based tool, such as Bayesian formalism or decision tree induction.

The most popular method is the last one; it is based on the present values and values from other attributes.

Junk, noisy data, or outlier

As in a physics or statistics test, noise is a random error that occurs during the test process to seize the measured data. No matter what means you apply to the data gathering process, noise inevitably exists.

Approaches for data smoothing are listed here. Along with the progress of data mining study, new methods keep occurring. Let's have a look at them:

- **Binning**: This is a local scope smoothing method in which the neighborhood values are used to compute the final value for the certain bin. The sorted data is distributed into a number of bins and each value in that bin will be replaced by a value depending on some certain computation of the neighboring values. The computation can be bin median, bin boundary, which is the boundary data of that bin.
- **Regression**: The target of regression is to find the best curve or something similar to one in a multidimensional space; as a result, the other values will be used to predict the value of the target attribute or variable. In other aspects, it is a popular means for smoothing.
- **Classification or outlier**: The classifier is another inherent way to find the noise or outlier. During the process of classifying, most of the source data is grouped into couples of groups, except the outliers.

Data integration

Data integration combines data from multiple sources to form a coherent data store. The common issues here are as follows:

- **Heterogeneous data**: This has no common key
- **Different definition**: This is intrinsic, that is, same data with different definition, such as a different database schema
- **Time synchronization**: This checks if the data is gathered under same time periods
- **Legacy data**: This refers to data left from the old system
- **Sociological factors**: This is the limit of data gathering

There are several approaches that deal with the above issues:

- **Entity identification problem**: Schema integration and object matching are tricky. This referred to as the entity identification problem.
- **Redundancy and correlation analysis**: Some redundancies can be detected by correlation analysis. Given two attributes, such an analysis can measure how strongly one attribute implies the other, based on the available data.
- **Tuple Duplication**: Duplication should be detected at the tuple level to detect redundancies between attributes
- **Data value conflict detection and resolution**: Attributes may differ on the abstraction level, where an attribute in one system is recorded at a different abstraction level

Data dimension reduction

Reduction of dimensionality is often necessary in the analysis of complex multivariate datasets, which is always in high-dimensional format. So, for example, problems modeled by the number of variables present, the data mining tasks on the multidimensional analysis of qualitative data. There are also many methods for data dimension reduction for qualitative data.

The goal of dimensionality reduction is to replace large matrix by two or more other matrices whose sizes are much smaller than the original, but from which the original can be approximately reconstructed, usually by taking their product with loss of minor information.

Eigenvalues and Eigenvectors

An eigenvector for a matrix is defined as when the matrix (*A* in the following equation) is multiplied by the eigenvector (*v* in the following equation). The result is a constant multiple of the eigenvector. That constant is the eigenvalue associated with this eigenvector. A matrix may have several eigenvectors.

$$Av = \lambda v$$

An eigenpair is the eigenvector and its eigenvalue, that is, (v, λ) in the preceding equation.

Principal-Component Analysis

The **Principal-Component Analysis** (**PCA**) technique for dimensionality reduction views data that consists of a collection of points in a multidimensional space as a matrix, in which rows correspond to the points and columns to the dimensions.

The product of this matrix and its transpose has eigenpairs, and the principal eigenvector can be viewed as the direction in the space along which the points best line up. The second eigenvector represents the direction in which deviations from the principal eigenvector are the greatest.

Dimensionality reduction by PCA is to approximate the data by minimizing the root-mean-square error for the given number of columns in the representing matrix, by representing the matrix of points by a small number of its eigenvectors.

Singular-value decomposition

The **singular-value decomposition** (**SVD**) of a matrix consists of following three matrices:

- U
- ∑
- V

U and V are column-orthonormal; as vectors, the columns are orthogonal and their length is 1. ∑ is a diagonal matrix and the values along its diagonal are called singular values. The original matrix equals to the product of U, ∑, and the transpose of V.

SVD is useful when there are a small number of concepts that connect the rows and columns of the original matrix.

Dimensionality reduction by SVD for matrix U and V are typically as large as the original. To use fewer columns for U and V, delete the columns corresponding to the smallest singular values from U, V, and $\sum$. This minimizes the error in reconstruction of the original matrix from the modified U, $\sum$, and V.

CUR decomposition

The CUR decomposition seeks to decompose a sparse matrix into sparse, smaller matrices whose product approximates the original matrix.

The CUR chooses from a given sparse matrix a set of columns C and a set of rows R, which play the role of U and V^T in SVD. The choice of rows and columns is made randomly with a distribution that depends on the square root of the sum of the squares of the elements. Between C and R is a square matrix called U, which is constructed by a pseudo-inverse of the intersection of the chosen rows and columns.

> By CUR solution, the three component matrices C, U, and R will be retrieved. The product of those three will approximate the original matrix M. For R community, rCUR is an R package for the CUR matrix decomposition.

Data transformation and discretization

As we know from the previous section, there are always some data formats that are best suited for specific data mining algorithms. Data transformation is an approach to transform the original data to preferable data format for the input of certain data mining algorithms before the processing.

Data transformation

Data transformation routines convert the data into appropriate forms for mining. They're shown as follows:

- **Smoothing**: This uses binning, regression, and clustering to remove noise from the data
- **Attribute construction**: In this routine, new attributes are constructed and added from the given set of attributes
- **Aggregation**: In this summary or aggregation, operations are performed on the data

- **Normalization**: Here, the attribute data is scaled so as to fall within a smaller range
- **Discretization**: In this routine, the raw values of a numeric attribute are replaced by interval label or conceptual label
- **Concept hierarchy generation for nominal data**: Here, attributes can be generalized to higher level concepts

Normalization data transformation methods

To avoid dependency on the choice of measurement units on data attributes, the data should be normalized. This means transforming or mapping the data to a smaller or common range. All attributes gain an equal weight after this process. There are many normalization methods. Let's have a look at some of them:

- **Min-max normalization**: This preserves the relationships among the original data values and performs a linear transformation on the original data. The applicable ones of the actual maximum and minimum values of an attribute will be normalized.
- **z-score normalization**: Here the values for an attribute are normalized based on the mean and standard deviation of that attribute. It is useful when the actual minimum and maximum of an attribute to be normalized are unknown.
- **Normalization by decimal scaling**: This normalizes by moving the decimal point of values of attribute.

Data discretization

Data discretization transforms numeric data by mapping values to interval or concept labels. Discretization techniques include the following:

- **Data discretization by binning**: This is a top-down unsupervised splitting technique based on a specified number of bins.
- **Data discretization by histogram analysis**: In this technique, a histogram partitions the values of an attribute into disjoint ranges called buckets or bins. It is also an unsupervised method.
- **Data discretization by cluster analysis**: In this technique, a clustering algorithm can be applied to discretize a numerical attribute by partitioning the values of that attribute into clusters or groups.

- **Data discretization by decision tree analysis**: Here, a decision tree employs a top-down splitting approach; it is a supervised method. To discretize a numeric attribute, the method selects the value of the attribute that has minimum entropy as a split-point, and recursively partitions the resulting intervals to arrive at a hierarchical discretization.
- **Data discretization by correlation analysis**: This employs a bottom-up approach by finding the best neighboring intervals and then merging them to form larger intervals, recursively. It is supervised method.

Visualization of results

Visualization is the graphic presentation of data-portrayals meant to reveal complex information at a glance, referring to all types of structured representation of information. This includes graphs, charts, diagrams, maps, storyboards, and other structured illustrations.

Good visualization of results gives you the chance to look at data through the eyes of experts. It is beautiful not only for their aesthetic design, but also for the elegant layers of detail that efficiently generate insight and new understanding.

The result of every data mining algorithm can be visualized and clarified by the use of the algorithms. Visualization plays an important role in the data mining process.

There are four major features that create the best visualizations:

- **Novel**: It must not only merely being a conduit for information, but offer some novelty in the form of new style of information.
- **Informative**: The attention to these factors and the data itself will make a data visualization effective, successful, and beautiful.
- **Efficient**: A nice visualization has an explicit goal, a clearly defined message, or a special perspective on the information that it is made to convey. It must be as simple as possible and straightforward, but shouldn't lose out on necessary, relevant complexity. The irrelevant data serves as noises here. It should reflect the qualities of the data that they represent, reveal properties and relationships inherent and implicit in the data source to bring new knowledge, insight, and enjoyment to final user.
- **Aesthetic**: The graphic must serve the primary goal of presenting information, not only axes and layout, shapes, lines, and typography, but also the appropriate usage of these ingredients.

Visualization with R

R provides the production of publication-quality diagrams and plots. There are graphic facilities distributed with R, and also some facilities that are not part of the standard R installation. You can use R graphics from command line.

The most important feature of the R graphics setup is the existence of two distinct graphics systems within R:

- The traditional graphics system
- Grid graphics system

The most appropriate facilities will be evaluated and applied to the visualization of every result of all algorithms listed in the book.

Functions in the graphics systems and add-on packages can be divided into several types:

- High-level functions that produce complete plots
- Low-level functions to add further output to an existing plot
- The ones to work interactively with graphical output

> R graphics output can be produced in a wide range of graphical formats, such as PNG, JPEG, BMP, TIFF, SVG, PDF, and PS.

To enhance your knowledge about this chapter, here are some practice questions for you to have check about the concepts.

Time for action

Let's now test what we've learned so far:

- What is the difference between data mining and machine learning?
- What is data preprocessing and data quality?
- Download R and install R on your machine.
- Compare and contrast data mining and machine learning.

Summary

In this chapter, we looked at the following topics:

- An introduction to data mining and available data sources
- A quick overview of R and the necessity to use R
- A description of statistics and machine learning, and their relations to data mining
- The two standard industrial data mining process
- Data attributes types and the data measurement approaches
- The three important steps in data preprocessing
- An introduction to the scalability and efficiency of data mining algorithms, and data visualization methods and necessities
- A discussion on social network mining, text mining, and web data mining
- A short introduction about RHadoop and Map Reduce

In the following chapters, the reader will learn how to implement various data mining algorithms and manipulate data with R.

2
Mining Frequent Patterns, Associations, and Correlations

In this chapter, we will learn how to mine frequent patterns, association rules, and correlation rules when working with R programs. Then, we will evaluate all these methods with benchmark data to determine the interestingness of the frequent patterns and rules. We will cover the following topics in this chapter:

- Introduction to associations and patterns
- Market basket analysis
- Hybrid association rules mining
- Mining sequence datasets
- High-performance algorithms

The algorithms to find frequent items from various data types can be applied to numeric or categorical data. Most of these algorithms have one common basic algorithmic form, which is **A-Priori**, depending on certain circumstances. Another basic algorithm is **FP-Growth**, which is similar to A-Priori. Most pattern-related mining algorithms derive from these basic algorithms.

With frequent patterns found as one input, many algorithms are designed to find association and correlation rules. Each algorithm is only a variation from the basic algorithm.

Along with the growth, size, and types of datasets from various domains, new algorithms are designed, such as the multistage algorithm, the multihash algorithm, and the limited-pass algorithm.

An overview of associations and patterns

One popular task for data mining is to find relations among the source dataset; this is based on searching frequent patterns from various data sources, such as market baskets, graphs, and streams.

All the algorithms illustrated in this chapter are written from scratch in the R language for the purpose of explaining association analysis, and the code will be demonstrated using the standard R packages for the algorithms such as arules.

Patterns and pattern discovery

With many applications across a broad field, frequent pattern mining is often used in solving various problems, such as the market investigation for a shopping mall from the transaction data.

Frequent patterns are the ones that often occur in the source dataset. The dataset types for frequent pattern mining can be itemset, subsequence, or substructure. As a result, the frequent patterns found are known as:

- Frequent itemset
- Frequent subsequence
- Frequent substructures

These three frequent patterns will be discussed in detail in the upcoming sections.

These newly founded frequent patterns will serve as an important platform when searching for recurring interesting rules or relationships among the given dataset.

Various patterns are proposed to improve the efficiency of mining on a dataset. Some of them are as follows; they will be defined in detail later:

- Closed patterns
- Maximal patterns
- Approximate patterns
- Condensed patterns
- Discriminative frequent patterns

The frequent itemset

The **frequent itemset** originated from true market basket analysis. In a store such as Amazon, there are many orders or transactions; a certain customer performs a transaction where their Amazon shopping cart includes some items. The mass result of all customers' transactions can be used by the storeowner to find out what items are purchased together by customers. As a simple definition, itemset denotes a collection of zero or more items.

We call a transaction a **basket**, and a set of items can belong to any basket. We will set the variable `s` as the support threshold, which is compared with the count of a certain set of items that appear in all the baskets. If the count of a certain set of items that appear in all the baskets is not less than `s`, we would call the itemset a **frequent itemset**.

An itemset is called a k-itemset if it contains k pieces of items, where k is a non-zero integer. The support count of an itemset is $support\_count(X)$, the count of itemset contained X, given the dataset.

For a predefined minimum support threshold s, the itemset `X` is a frequent itemset if $support\_count(X) \geq s$. The minimum support threshold `s` is a customizable parameter, which can be adjusted by domain experts or experiences.

The frequent itemset is also used in many domains. Some of them are shown in the following table:

	Items	**Baskets**	**Comments**
Related concepts	Words	Documents	
Plagiarism	Documents	Sentences	
Biomarkers	Biomarkers and diseases	The set of data about a patient	

If an itemset is frequent, then any of its subset must be frequent. This is known as the A-Priori principle, the foundation of the A-Priori algorithm. The direct application of the A-Priori principle is to prune the huge number of frequent itemsets.

One important factor that affects the number of frequent itemsets is the minimum support count: the lower the minimum support count, the larger the number of frequent itemsets.

For the purpose of optimizing the frequent itemset-generation algorithm, some more concepts are proposed:

- An itemset X is closed in dataset S, if $\forall Y \in S, X \subset Y, then\, support\_count(X) \neq support\_count(Y)$; X is also called a closed itemset. In other words, if X is frequent, then X is a closed frequent itemset.
- An itemset X is a maximal frequent itemset if $\forall Y \in S, X \subset Y, then\, Y\, is\, not\, frequent$; in other words, `Y` does not have frequent supersets.
- An itemset X is considered a constrained frequent itemset once the frequent itemset satisfies the user-specified constraints.
- An itemset X is an approximate frequent itemset if X derives only approximate support counts for the mined frequent itemsets.
- An itemset X is a top-k frequent itemset in the dataset S if X is the k-most frequent itemset, given a user-defined value k.

The following example is of a transaction dataset. All itemsets only contain items from the set, $D = \{\mathrm{I}_k | \{I_k | k \in [1,7]\ \}$.Let's assume that the minimum support count is 3.

tid (transaction id)	**List of items in the itemset or transaction**
T001	I_1, I_2, I_4, I_7
T002	I_2, I_3, I_6
T003	I_1, I_4, I_6
T004	I_1, I_2, I_5
T005	I_2, I_3, I_4
T006	I_2, I_5, I_6
T007	I_2, I_4, I_7
T008	I_1, I_7
T009	I_1, I_2, I_3
T010	I_1, I_2, I_4

Then, we will get the frequent itemsets $L_1 = \{\mathrm{I}_k | k \in \{1, 2, 4, 6, 7\}\}$ and $L_2 = \{\{I_1, I_2\}, \{I_1, I_4\}, \{I_2, I_4\}\}$.

The frequent subsequence

The frequent sequence is an ordered list of elements where each element contains at least one event. An example of this is the page-visit sequence on a site by the specific web page the user is on more concretely speaking, the order in which a certain user visits web pages. Here are two examples of the frequent subsequence:

- **Customer**: Successive shopping records of certain customers in a shopping mart serves as the sequence, each item bought serves as the event item, and all the items bought by a customer in one shopping are treated as elements or transactions
- **Web usage data**: Users who visit the history of the WWW are treated as a sequence, each UI/page serves as the event or item, and the element or transaction can be defined as the pages visited by users with one click of the mouse

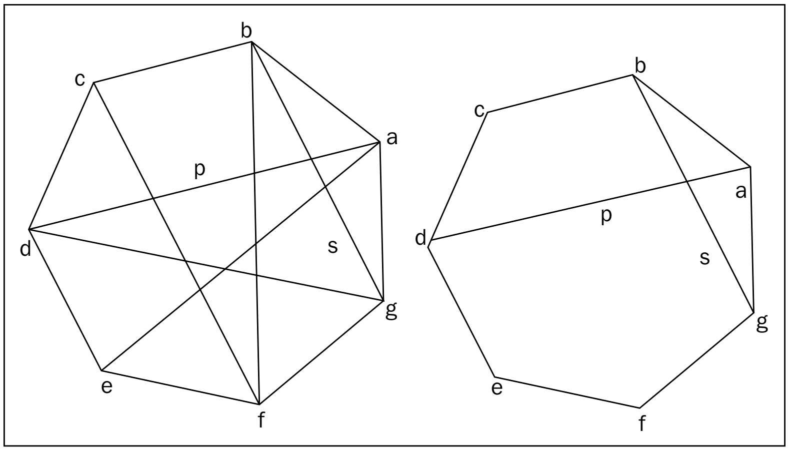
The length of a sequence is defined by the number of items contained in the sequence. A sequence of length k is called a k-sequence. The size of a sequence is defined by the number of itemsets in the sequence. We call a sequence $s_1 = < a_1 a_2 ... a_r >$ as a subsequence of the sequence $s_2 =< b_1 b ... b_r >$ or s_2 as the super sequence of s_1 when $\exists 1 \le j_1 \le j_2 \le ... \le j_{r-1} \le j_r \le v, and\ a_1 \sqsubseteq b_{j_1}, a_2 \sqsubseteq b_{j_2}, ..., a_r \sqsubseteq b_{j_r}$ is satisfied.

The frequent substructures

In some domains, the tasks under research can be modeled with a graph theory. As a result, there are requirements for mining common subgraphs (subtrees or sublattices); some examples are as follows:

- **Web mining**: Web pages are treated as the vertices of graph, links between pages serve as edges, and a user's page-visiting records construct the graph.
- **Network computing**: Any device with computation ability on the network serves as the vertex, and the interconnection between these devices serves as the edge. The whole network that is made up of these devices and interconnections is treated as a graph.
- **Semantic web**: XML elements serve as the vertices, and the parent/child relations between them are edges; all these XML files are treated as graphs.

A graph G is represented by `G = (V, E)`, where `V` represents a group of vertices, and `E` represents a group of edges. A graph $G' = (V', E')$ is called as subgraph of graph `G = (V, E)` once $V' \subseteq V$ and $V' \subseteq V$. Here is an example of a subgraph. There is the original graph with vertices and edges on the left-hand side of the following figure and the subgraph on the right-hand side with some edges omitted (or omission of vertices in other circumstances):



Relationship or rules discovery

Mining of association rules is based on the frequent patterns found. The different emphases on the interestingness of relations derives two types of relations for further research: association rules and correlation rules.

Association rules

In a later section, a method to show association analysis is illustrated; this is a useful method to discover interesting relationships within a huge dataset. The relations can be represented in the form of association rules or frequent itemsets.

Association rule mining is to find the result rule set on a given dataset (the transaction data set or other sequence-pattern-type dataset), a predefined minimum support count *s*, and a predefined confidence c, given any found rule $X \to Y$ $\text{support\_count}(\mathrm{X} \to \mathrm{Y}) \geq \mathrm{S}$, and $\mathit{confidence}(X \to Y) \geq c$.

$X \to Y$ is an association rule where $X \cap Y = \phi$; X and Y are disjoint. The interesting thing about this rule is that it is measured by its **support** and **confidence**. Support means the frequency in which this rule appears in the dataset, and confidence means the probability of the appearance of `Y` when `X` is present.

For association rules, the key measures of rule interestingness are rule support and confidence. Their relationship is given as follows:

$$confidence(X \rightarrow Y) = P(Y|X) = \frac{P(X \cup Y)}{P(X)} = \frac{support\_count(X \cup Y)}{support\_count(X)}$$

`support_count(X)` is the count of itemset in the dataset, contained `X`.

As a convention, in `support_count(X)`, in the confidence value and support count value are represented as a percentage between 0 and 100.

The association rule $X \rightarrow Y$ is strong once $confidence(X \rightarrow Y) \geq c$ and $support\_count(X \cup Y) > s$. The predefined minimum support threshold is *s*, and `c` is the predefined minimum confidence threshold.

The meaning of the found association rules should be explained with caution, especially when there is not enough to judge whether the rule implies causality. It only shows the co-occurrence of the prefix and postfix of the rule. The following are the different kinds of rules you can come across:

- A rule is a Boolean association rule if it contains association of the presence of the item
- A rule is a single-dimensional association if there is, at the most, only one dimension referred to in the rules
- A rule is a multidimensional association rule if there are at least two dimensions referred to in the rules
- A rule is a correlation-association rule if the relations or rules are measured by statistical correlation, which, once passed, leads to a correlation rule
- A rule is a quantitative-association rule if at least one item or attribute contained in it is quantitative

Correlation rules

In some situations, the support and confidence pairs are not sufficient to filter uninteresting association rules. In such a case, we will use support count, confidence, and correlations to filter association rules.

There are a lot of methods to calculate the correlation of an association rule, such as χ^2 analyses, all-confidence analysis, and cosine. For a k-itemset $X = \{i_1, i_2, \ldots, i_k\}$, define the all-confidence value of X as:

$$all\_confidence(X) = support\_count(X) / max\left\{support\_count(i_j) \vee \forall i_j \in X\right\}$$

$$lift(X \to Y) = \frac{confidence(X \to Y)}{P(Y)} = P(X \cup Y) / \left(P(X)P(Y)\right)$$

Market basket analysis

Market basket analysis is the methodology used to mine a shopping cart of items bought or just those kept in the cart by customers. The concept is applicable to a variety of applications, especially for store operations. The source dataset is a massive data record. The aim of market basket analysis is to find the association rules between the items within the source dataset.

The market basket model

The market basket model is a model that illustrates the relation between a basket and its associated items. Many tasks from different areas of research have this relation in common. To summarize them all, the market basket model is suggested as the most typical example to be researched.

The basket is also known as the transaction set; this contains the itemsets that are sets of items belonging to same itemset.

The A-Priori algorithm is a level wise, itemset mining algorithm. The Eclat algorithm is a tidset intersection itemset mining algorithm based on tidset intersection in contrast to A-Priori. FP-growth is a frequent pattern tree algorithm. The tidset denotes a collection of zeros or IDs of transaction records.

A-Priori algorithms

As a common strategy to design algorithms, the problem is divided into two subproblems:

- The frequent itemset generation
- Rule generation

The strategy dramatically decreases the search space for association mining algorithms.

Input data characteristics and data structure

As the input of the A-Priori algorithm, the original input itemset is binarized, that is, 1 represents the presence of a certain item in the itemset; otherwise, it is 0. As a default assumption, the average size of the itemset is small. The popular preprocessing method is to map each unique available item in the input dataset to a unique integer ID.

The itemsets are usually stored within databases or files and will go through several passes. To control the efficiency of the algorithm, we need to control the count of passes. During the process when itemsets pass through other itemsets, the representation format for each itemset you are interested in is required to count and store for further usage of the algorithm.

There is a monotonicity feature in the itemsets under research; this implies that every subset of a frequent itemset is frequent. This characteristic is used to prune the search space for the frequent itemset in the process of the A-Priori algorithm. It also helps compact the information related to the frequent itemset. This feature gives us an intrinsic view that focuses on smaller-sized frequent itemsets. For example, there are three frequent 2-itemsets contained by one certain frequent 3-itemset.

> When we talk about k-itemsets means an itemset containing k items.

The basket is in a format called the **horizontal format** and contains a basket or transaction ID and a number of items; it is used as the basic input format for the A-Priori algorithm. In contrast, there is another format known as the **vertical format**; this uses an item ID and a series of the transaction IDs. The algorithm that works on vertical data format is left as an exercise for you.

The A-Priori algorithm

Two actions are performed in the generation process of the A-Priori frequent itemset: one is **join**, and the other is **prune**.

> One important assumption is that the items within any itemset are in a lexicographic order.

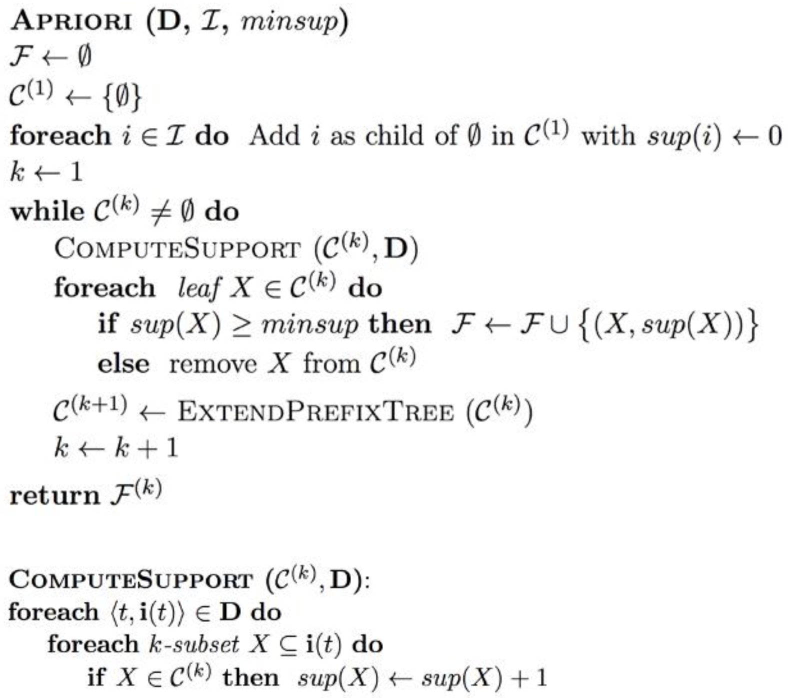
- **Join action**: Given that L_k is the set of frequent k-itemsets, a set of candidates to find L_k is generated. Let's call it C_k.

$$
\begin{aligned}
C_k = L_{k-1} \bowtie L_{k-1} = \{ l' \mid \exists\, l_1, l_2 \in L_{k-1}, \forall\ m \in [1, k-2], l_1[m] \\
= l_2[m] \text{and} l_1[k-1] \le l_2[k-1], \\
\mathit{then}\ l' < l_1[1], l_1[2], \ldots, l_1[k-2], l_1[k-1], l_2[k-1] > \}
\end{aligned}
$$

- **Prune action**: $L_k \subseteq C_k$, the size of C_k, the candidate itemset, is usually much bigger than L_k, to save computation cost; monotonicity characteristic of frequent itemset is used here to prune the size of C_k.

$$\forall c \in C_k, \exists c_{k-1} is\ a(k-1) - subset\ of\ c, and\ c_{k-1} \notin L_{k-1} \Rightarrow c_k \notin L_k$$

Here is the pseudocode to find all the frequent itemsets:



```
APRIORI (D, $\mathcal{I}$, $minsup$)
$\mathcal{F} \leftarrow \emptyset$
$\mathcal{C}^{(1)} \leftarrow \{\emptyset\}$
foreach $i \in \mathcal{I}$ do  Add $i$ as child of $\emptyset$ in $\mathcal{C}^{(1)}$ with $sup(i) \leftarrow 0$
$k \leftarrow 1$
while $\mathcal{C}^{(k)} \neq \emptyset$ do
    COMPUTESUPPORT ($\mathcal{C}^{(k)}$, D)
    foreach  $leaf\ X \in \mathcal{C}^{(k)}$ do
        if $sup(X) \geq minsup$ then  $\mathcal{F} \leftarrow \mathcal{F} \cup \{(X, sup(X))\}$
        else  remove $X$ from $\mathcal{C}^{(k)}$
    $\mathcal{C}^{(k+1)} \leftarrow$ EXTENDPREFIXTREE ($\mathcal{C}^{(k)}$)
    $k \leftarrow k + 1$
return $\mathcal{F}^{(k)}$


COMPUTESUPPORT ($\mathcal{C}^{(k)}$, D):
foreach $\langle t, \mathbf{i}(t) \rangle \in$ D do
    foreach $k$-$subset$ $X \subseteq \mathbf{i}(t)$ do
        if $X \in \mathcal{C}^{(k)}$ then  $sup(X) \leftarrow sup(X) + 1$
```

```
EXTENDPREFIXTREE ($\mathcal{C}^{(k)}$):
foreach leaf $X_a \in \mathcal{C}^{(k)}$ do
    foreach leaf $X_b \in \text{SIBLING}(X_a)$, such that $b > a$ do
        $X_{ab} \leftarrow X_a \cup X_b$

        if $X_j \in \mathcal{C}^{(k)}$, for all $X_j \subset X_{ab}$, such that $|X_j| = |X_{ab}| - 1$ then
            Add $X_{ab}$ as child of $X_a$ with $sup(X_{ab}) \leftarrow 0$

    if no extensions from $X_a$ then remove $X_a$ from $\mathcal{C}^{(k)}$
return $\mathcal{C}^{(k)}$
```

The R implementation

R code of the A-Priori frequent itemset generation algorithm goes here. D is a transaction dataset. Suppose `MIN_SUP` is the minimum support count threshold. The output of the algorithm is L, which is a frequent itemsets in D.

The output of the A-Priori function can be verified with the R add-on package, arules, which is a pattern-mining and association-rules-mining package that includes A-Priori and éclat algorithms. Here is the R code:

```
Apriori <- function (data, I, MIN_SUP, parameter = NULL){
  f <- CreateItemsets()
  c <- FindFrequentItemset(data,I,1, MIN_SUP)
  k <- 2
  len4data <- GetDatasetSize(data)
  while( !IsEmpty(c[[k-1]]) ){
        f[[k]] <- AprioriGen(c[k-1])
         for( idx in 1: len4data ){
             ft <- GetSubSet(f[[k]],data[[idx]])
             len4ft <- GetDatasetSize(ft)
             for( jdx in 1:len4ft ){
                IncreaseSupportCount(f[[k]],ft[jdx])
             }
         }
         c[[k]] <- FindFrequentItemset(f[[k]],I,k,MIN_SUP)
         k <- k+1
  }
  c
}
```

To verify the R code, the arules package is applied while verifying the output.

> Arules (*Hahsler et al., 2011*) provides the support to mine frequent itemsets, maximal frequent itemsets, closed frequent itemsets, and association rules too. A-Priori and Eclat algorithms are both available. Also cSPADE can be found in arulesSequence, the add-on for arules.

Given:

$$D = \left\{ tinnedfruit, tuna, milk, coke, water, biscuits, oil, soap \right\}$$

At first, we will sort D into an ordered list in a predefined order algorithm or simply the natural order of characters, which is used here. Then:

$$D = \left\{ \begin{matrix} I1 = biscuits, I2 = coke, I3 = milk, I4 = oil, \\ I5 = soap, I6 = tinnedfruit, I7 = tuna \end{matrix} \right\} = \{ I_k | k \in \left[1,7\right] \}$$

Let's assume that the minimum support count is 5; the following table is an input dataset:

tid (transaction id)	**List of items in the itemset or transaction**
T001	I_1, I_2, I_4, I_7
T002	I_2, I_3, I_6
T003	I_1, I_4, I_6
T004	I_1, I_2, I_5
T005	I_2, I_3, I_4
T006	I_2, I_5, I_6
T007	I_2, I_4, I_7
T008	I_1, I_7
T009	I_1, I_2, I_3
T010	I_1, I_2, I_4

In the first scan or pass of the dataset D, get the count of each candidate itemset C_1. The candidate itemset and its related count:

Itemset	Support count
$\{I_1\}$	6
$\{I_2\}$	8
$\{I_3\}$	2
$\{I_4\}$	5
$\{I_5\}$	2
$\{I_6\}$	3
$\{I_7\}$	3

We will get the L_1 after comparing the support count with minimum support count.

Itemset	Support count
$\{I_1\}$	6
$\{I_2\}$	8
$\{I_4\}$	5

We will generate C_2 by L_1, $C_2 = \{\{I_1, I_2\}, \{I_1, I_4\}, \{I_2, I_4\}\}$.

Itemset	Support count
$\{I_1, I_2\}$	4
$\{I_1, I_4\}$	3
$\{I_2, I_4\}$	4

After comparing the support count with the minimum support count, we will get $L_2 = \phi$. The algorithm then terminates.

A-Priori algorithm variants

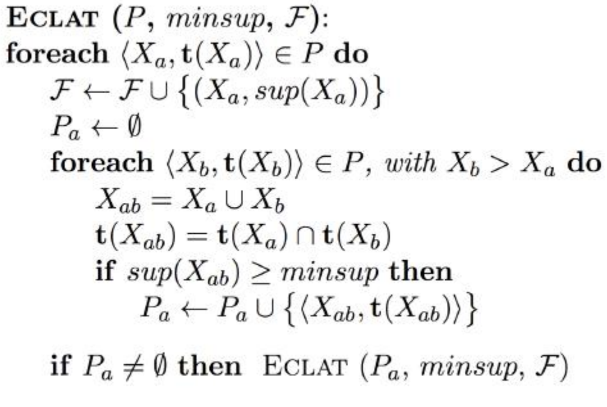
The various variants of A-Priori algorithms are designed mainly for the purpose of efficiency and scalability. Some of the improvements of the A-Priori algorithms are discussed in the upcoming sections.

The Eclat algorithm

The A-Priori algorithm loops as many times as the maximum length of the pattern somewhere. This is the motivation for the **Equivalence CLASS Transformation** (**Eclat**) algorithm. The Eclat algorithm explores the vertical data format, for example, using `<item id`, `tid set>` instead of `<tid`, `item id set>` that is, with the input data in the vertical format in the sample market basket file, or to discover frequent itemsets from a transaction dataset. The A-Priori property is also used in this algorithm to get frequent (*k+1*) itemsets from k itemsets.

The candidate itemset is generated by set intersection. The vertical format structure is called a tidset as defined earlier. If all the transaction IDs related to the item *I* are stored in a vertical format transaction itemset, then the itemset is the tidset of the specific item.

The support count is computed by the intersection between tidsets. Given two tidsets, `X` and `Y`, $support\_count(X \cap Y)$ is the cardinality of $X \cap Y$. The pseudocode is $\mathrm{F} \leftarrow \phi$, $\mathrm{P} \leftarrow \{<i, t(i)> | i \in I, |t(i)| \geq MIN\_SUP\}$.



```
$\textsc{Eclat}\ (P,\ minsup,\ \mathcal{F})$:
foreach $\langle X_a, \mathbf{t}(X_a)\rangle \in P$ do
    $\mathcal{F} \leftarrow \mathcal{F} \cup \{(X_a, sup(X_a))\}$
    $P_a \leftarrow \emptyset$
    foreach $\langle X_b, \mathbf{t}(X_b)\rangle \in P,\ with\ X_b > X_a$ do
        $X_{ab} = X_a \cup X_b$
        $\mathbf{t}(X_{ab}) = \mathbf{t}(X_a) \cap \mathbf{t}(X_b)$
        if $sup(X_{ab}) \geq minsup$ then
            $P_a \leftarrow P_a \cup \{\langle X_{ab}, \mathbf{t}(X_{ab})\rangle\}$

    if $P_a \neq \emptyset$ then $\textsc{Eclat}\ (P_a,\ minsup,\ \mathcal{F})$
```

The R implementation

Here is the R code for the Eclat algorithm to find the frequent patterns. Before calling the function, `f` is set to empty, and `p` is the set of frequent 1-itemsets:

```
Eclat  <- function (p,f,MIN_SUP){
  len4tidsets <- length(p)
  for(idx in 1:len4tidsets){
     AddFrequentItemset(f,p[[idx]],GetSupport(p[[idx]]))
     Pa <- GetFrequentTidSets(NULL,MIN_SUP)
       for(jdx in idx:len4tidsets){
         if(ItemCompare(p[[jdx]],p[[idx]]) > 0){
             xab <- MergeTidSets(p[[idx]],p[[jdx]])
             if(GetSupport(xab)>=MIN_SUP){
                AddFrequentItemset(pa,xab,
                GetSupport(xab))
               }
           }
     }
     if(!IsEmptyTidSets(pa)){
       Eclat(pa,f,MIN_SUP)
    }
  }
}
```

Here is the running result of one example, `I = {beer, chips, pizza, wine}`. The transaction dataset with horizontal and vertical formats, respectively, are shown in the following table:

tid	**X**
1	`{beer, chips, wine}`
2	`{beer, chips}`
3	`{pizza, wine}`
4	`{chips, pizza}`

x	**tidset**
`beer`	`{1,2}`
`chips`	`{1,2,4}`
`pizza`	`{3,4}`
`wine`	`{1,3}`

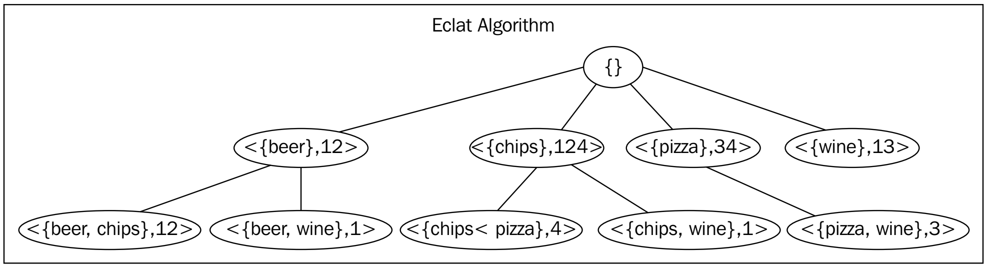
The binary format of this information is in the following table.

tid	beer	chips	pizza	wine
1	1	1	0	1
2	1	1	0	0
3	0	0	1	1
4	0	1	1	0

Before calling the Eclat algorithm, we will set *MIN_SUP=2*, $\mathrm{F} \leftarrow \{\}$,

$$\mathrm{P} \leftarrow \left\{ < beer, 12 >, < chips, 124 >, < pizza, 34 >, < wine, 13 > \right\}$$

The running process is illustrated in the following figure. After two iterations, we will get frequent tidsets, {`beer, 12 >, < chips, 124>, <pizza, 34>, <wine, 13>, <` {`beer, chips`}`, 12>`}:



The output of the Eclat function can be verified with the R add-on package, arules.

The FP-growth algorithm

The FP-growth algorithm is an efficient method targeted at mining frequent itemsets in a large dataset. The main difference between the FP-growth algorithm and the A-Priori algorithm is that the generation of a candidate itemset is not needed here. The pattern-growth strategy is used instead. The FP-tree is the data structure.

Input data characteristics and data structure

The data structure used is a hybrid of vertical and horizontal datasets; all the transaction itemsets are stored within a tree structure. The tree structure used in this algorithm is called a frequent pattern tree. Here is example of the generation of the structure, I = {A, B, C, D, E, F}; the transaction dataset D is in the following table, and the FP-tree building process is shown in the next upcoming image. Each node in the FP-tree represents an item and the path from the root to that item, that is, the node list represents an itemset. The support information of this itemset is included in the node as well as the item too.

tid	**X**
1	{A, B, C, D, E}
2	{A, B, C, E}
3	{A, D, E}
4	{B, E, D}
5	{B, E, C}
6	{E, C, D}
7	{E, D}

The sorted item order is listed in the following table:

item	E	D	C	B	A
support_count	7	5	4	4	3

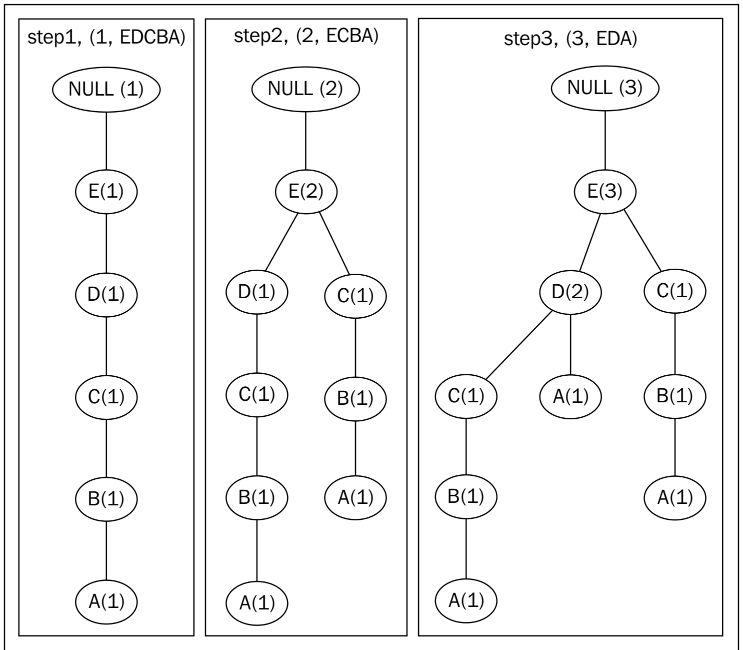
Reorder the transaction dataset with this new decreasing order; get the new sorted transaction dataset, as shown in this table:

tid	**X**
1	{E, D, C, B, A}
2	{E, C, B, A}
3	{E, D, A}
4	{E, D, B}
5	{E, C, B}
6	{E, D, C}
7	{E, D}

The FP-tree building process is illustrated in the following images, along with the addition of each itemset to the FP-tree. The support information is calculated at the same time, that is, the support counts of the items on the path to that specific node are incremented.

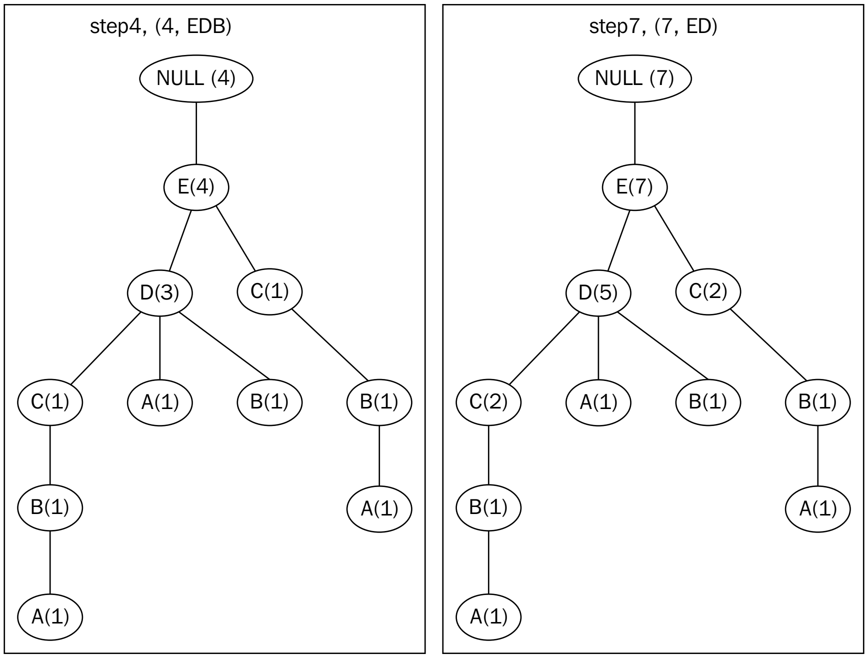
The most frequent items are put at the top of the tree; this keeps the tree as compact as possible. To start building the FP-tree, the items should be decreasingly ordered by the support count. Next, get the list of sorted items and remove the infrequent ones. Then, reorder each itemset in the original transaction dataset by this order.

Given *MIN_SUP=3*, the following itemsets can be processed according to this logic:



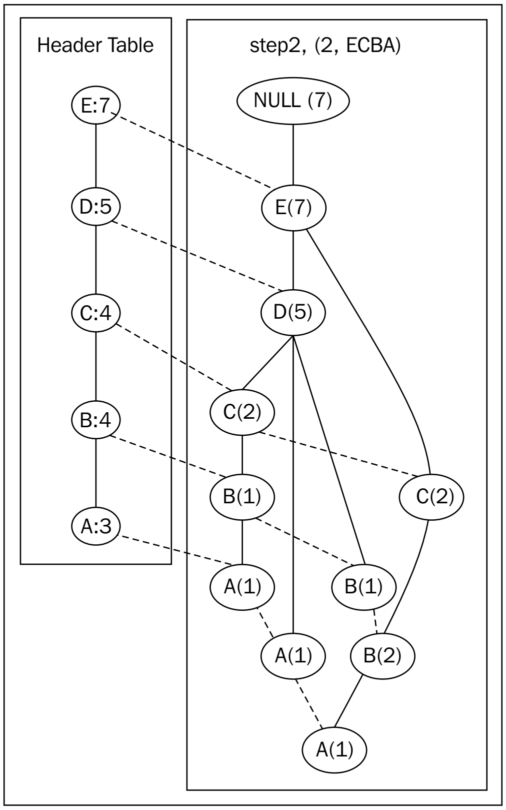
FP-tree building process, step1~step3

The result after performing steps 4 and 7 are listed here, and the process of the algorithm is very simple and straight forward:



FP-tree building process, step4~step7

A header table is usually bound together with the frequent pattern tree. A link to the specific node, or the item, is stored in each record of the header table.



The FP-tree serves as the input of the FP-growth algorithm and is used to find the frequent pattern or itemset. Here is an example of removing the items from the frequent pattern tree in a reverse order or from the leaf; therefore, the order is *A*, *B*, *C*, *D*, and *E*. Using this order, we will then build the projected FP-tree for each item.

The FP-growth algorithm

Here is the pseudocode with recursion definition; the input values are $R \leftarrow GenerateFPTree(D), P \leftarrow \phi, F \leftarrow \phi$

```
$\mathcal{F}[I] := \{\}$
for all $i \in \mathcal{I}$ occurring in $\mathcal{D}$ do
  $\mathcal{F}[I] := \mathcal{F}[I] \cup \{I \cup \{i\}\}$

  $\mathcal{D}^i := \{\}$
  $H := \{\}$
  for all $j \in \mathcal{I}$ occurring in $\mathcal{D}$ such that $j > i$ do
    if $support(I \cup \{i, j\}) \geq \sigma$ then
      $H := H \cup \{j\}$
    end if
  end for
  for all $(tid, X) \in \mathcal{D}$ with $i \in X$ do
    $\mathcal{D}^i := \mathcal{D}^i \cup \{(tid, X \cap H)\}$
  end for
  // Depth-first recursion
  Compute $\mathcal{F}[I \cup \{i\}](\mathcal{D}^i, \sigma)$
  $\mathcal{F}[I] := \mathcal{F}[I] \cup \mathcal{F}[I \cup \{i\}]$
end for
```

The R implementation

Here is the R source code of the main FP-growth algorithm:

```
FPGrowth  <- function (r,p,f,MIN_SUP){
    RemoveInfrequentItems(r)
    if(IsPath(r)){
       y <- GetSubset(r)
       len4y <- GetLength(y)
       for(idx in 1:len4y){
          x <- MergeSet(p,y[idx])
          SetSupportCount(x, GetMinCnt(x))
          Add2Set(f,x,support_count(x))
       }
  }else{
      len4r <- GetLength(r)
      for(idx in 1:len4r){
         x <- MergeSet(p,r[idx])
         SetSupportCount(x, GetSupportCount(r[idx]))
         rx <- CreateProjectedFPTree()
```

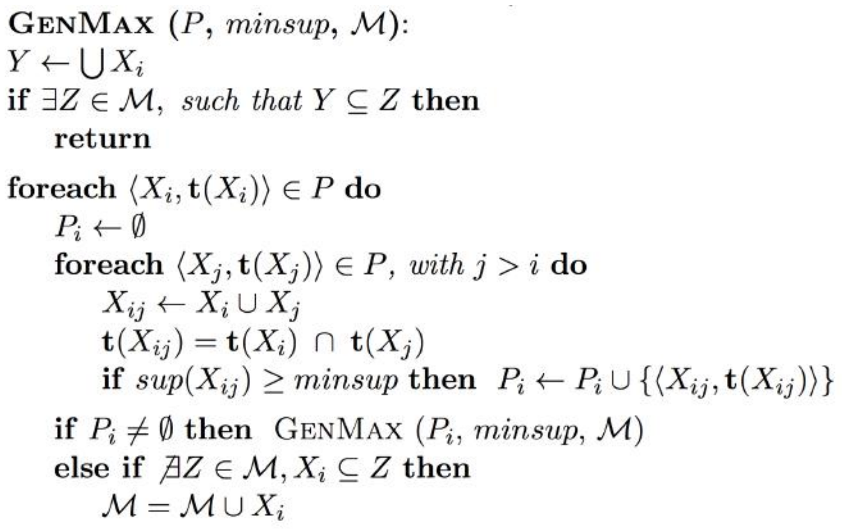
```
        path4idx <- GetAllSubPath(PathFromRoot(r,idx))
        len4path <- GetLength(path4idx)
        for( jdx in 1:len4path ){
          CountCntOnPath(r, idx, path4idx, jdx)
          InsertPath2ProjectedFPTree(rx, idx, path4idx, jdx,
            GetCnt(idx))
        }
        if( !IsEmpty(rx) ){
          FPGrowth(rx,x,f,MIN_SUP)
        }
      }
  }
}
```

The GenMax algorithm with maximal frequent itemsets

The GenMax algorithm is used to mine **maximal frequent itemset** (**MFI**) to which the maximality properties are applied, that is, more steps are added to check the maximal frequent itemsets instead of only frequent itemsets. This is based partially on the tidset intersection from the Eclat algorithm. The diffset, or the differential set as it is also known, is used for fast frequency testing. It is the difference between two tidsets of the corresponding items.

The candidate MFI is determined by its definition: assuming M as the set of MFI, if there is one X that belongs to M and it is the superset of the newly found frequent itemset Y, then Y is discarded; however, if X is the subset of Y, then X should be removed from M.

Here is the pseudocode before calling the GenMax algorithm, $M \leftarrow \phi, and\, P \leftarrow \{< \mathrm{X}_i,\ \mathrm{t}\left(\mathrm{X}_i\right) > | \, \mathrm{X}_i \in D, support\_count\left(X_i\right) \geq MIN\_SUP\}$, where D is the input transaction dataset.



```
GENMAX ($P$, $minsup$, $\mathcal{M}$):
$Y \leftarrow \bigcup X_i$
if $\exists Z \in \mathcal{M}$, such that $Y \subseteq Z$ then
    return
foreach $\langle X_i, \mathbf{t}(X_i) \rangle \in P$ do
    $P_i \leftarrow \emptyset$
    foreach $\langle X_j, \mathbf{t}(X_j) \rangle \in P$, with $j > i$ do
        $X_{ij} \leftarrow X_i \cup X_j$
        $\mathbf{t}(X_{ij}) = \mathbf{t}(X_i) \cap \mathbf{t}(X_j)$
        if $sup(X_{ij}) \geq minsup$ then $P_i \leftarrow P_i \cup \{\langle X_{ij}, \mathbf{t}(X_{ij}) \rangle\}$
    if $P_i \neq \emptyset$ then GENMAX ($P_i$, $minsup$, $\mathcal{M}$)
    else if $\nexists Z \in \mathcal{M}, X_i \subseteq Z$ then
        $\mathcal{M} = \mathcal{M} \cup X_i$
```

The R implementation

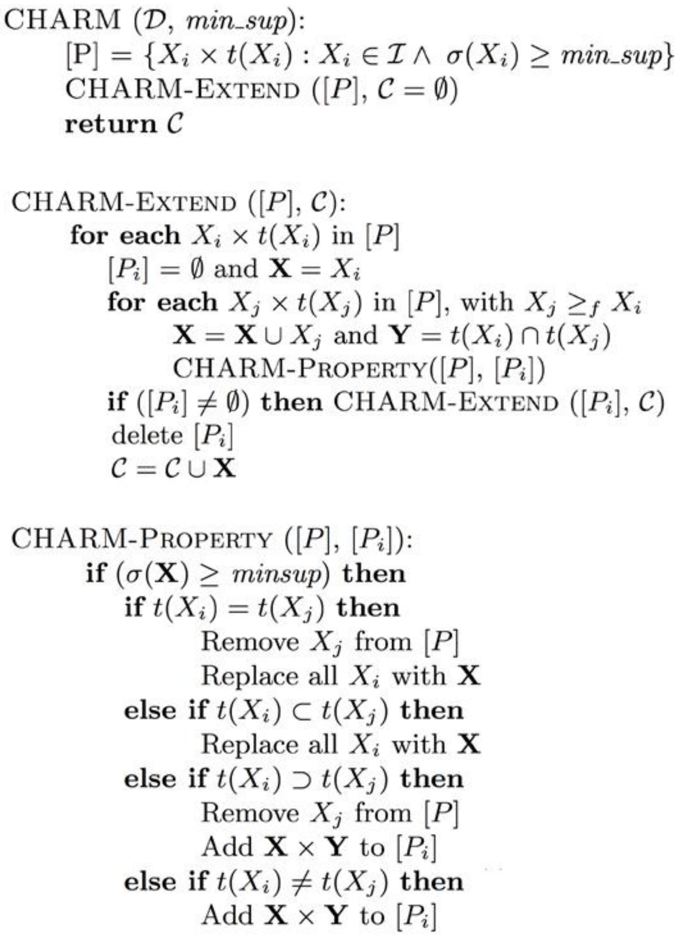
Here is the R source code of the main GenMax algorithm:

```
GenMax  <- function (p,m,MIN_SUP){
  y <- GetItemsetUnion(p)
  if( SuperSetExists(m,y) ){
    return
  }
  len4p <- GetLenght(p)
  for(idx in 1:len4p){
     q <- GenerateFrequentTidSet()
     for(jdx in (idx+1):len4p){
       xij <- MergeTidSets(p[[idx]],p[[jdx]])
         if(GetSupport(xij)>=MIN_SUP){
            AddFrequentItemset(q,xij,GetSupport(xij))
          }
     }
     if( !IsEmpty(q) ){
       GenMax(q,m,MIN_SUP)
     }else if( !SuperSetExists(m,p[[idx]]) ){
     Add2MFI(m,p[[idx]])
     }
   }
}
```

The Charm algorithm with closed frequent itemsets

Closure checks are performed during the mining of closed frequent itemsets. Closed frequent itemsets allow you to get the longest frequent patterns that have the same support. This allows us to prune frequent patterns that are redundant. The Charm algorithm also uses the vertical tidset intersection for efficient closure checks.

Here is the pseudocode before calling the Charm algorithm, $C \leftarrow \phi, and\ P \leftarrow \{< \mathrm{X}_i,\ \mathrm{t}(\mathrm{X}_i) >| \mathrm{X}_i \in D, support\_count(X_i) \geq MIN\_SUP\}$, where D is the input transaction dataset.



```
CHARM ($\mathcal{D}$, $min\_sup$):
    [P] = $\{X_i \times t(X_i) : X_i \in \mathcal{I} \wedge \sigma(X_i) \geq min\_sup\}$
    CHARM-EXTEND ([P], $\mathcal{C} = \emptyset$)
    return $\mathcal{C}$

CHARM-EXTEND ([P], $\mathcal{C}$):
    for each $X_i \times t(X_i)$ in [P]
        $[P_i] = \emptyset$ and $\mathbf{X} = X_i$
        for each $X_j \times t(X_j)$ in [P], with $X_j \geq_f X_i$
            $\mathbf{X} = \mathbf{X} \cup X_j$ and $\mathbf{Y} = t(X_i) \cap t(X_j)$
            CHARM-PROPERTY([P], $[P_i]$)
        if ($[P_i] \neq \emptyset$) then CHARM-EXTEND ($[P_i]$, $\mathcal{C}$)
        delete $[P_i]$
        $\mathcal{C} = \mathcal{C} \cup \mathbf{X}$

CHARM-PROPERTY ([P], $[P_i]$):
    if ($\sigma(\mathbf{X}) \geq minsup$) then
        if $t(X_i) = t(X_j)$ then
            Remove $X_j$ from [P]
            Replace all $X_i$ with $\mathbf{X}$
        else if $t(X_i) \subset t(X_j)$ then
            Replace all $X_i$ with $\mathbf{X}$
        else if $t(X_i) \supset t(X_j)$ then
            Remove $X_j$ from [P]
            Add $\mathbf{X} \times \mathbf{Y}$ to $[P_i]$
        else if $t(X_i) \neq t(X_j)$ then
            Add $\mathbf{X} \times \mathbf{Y}$ to $[P_i]$
```

The R implementation

Here is the R source code of the main algorithm:

```
Charm  <- function (p,c,MIN_SUP){
  SortBySupportCount(p)
  len4p <- GetLength(p)
  for(idx in 1:len4p){
     q <- GenerateFrequentTidSet()
     for(jdx in (idx+1):len4p){
        xij <- MergeTidSets(p[[idx]],p[[jdx]])
        if(GetSupport(xij)>=MIN_SUP){
          if( IsSameTidSets(p,idx,jdx) ){
            ReplaceTidSetBy(p,idx,xij)
            ReplaceTidSetBy(q,idx,xij)
            RemoveTidSet(p,jdx)
          }else{
             if( IsSuperSet(p[[idx]],p[[jdx]]) ){
               ReplaceTidSetBy(p,idx,xij)
               ReplaceTidSetBy(q,idx,xij)
             }else{
               Add2CFI(q,xij)
             }
          }
        }
     }
     if( !IsEmpty(q) ){
       Charm(q,c,MIN_SUP)
     }
     if( !IsSuperSetExists(c,p[[idx]]) ){
        Add2CFI(m,p[[idx]])
     }
  }
}
```

The algorithm to generate association rules

During the process of generating an algorithm for A-Priori frequent itemsets, the support count of each frequent itemset is calculated and recorded for further association rules mining processing, that is, for association rules mining.

To generate an association rule $X \rightarrow Y, l = X \cup Y$, l is a frequent itemset. Two steps are needed:

- First to get all the nonempty subsets of l.
- Then, for subset X of l, $Y = l - X$, the rule $X \rightarrow Y$ is a strong association rule only if $confidence(X \rightarrow Y) \geq minimum_{confidence}$ The support count of any rule of a frequent itemset is not less than the minimum support count.

Here is the pseudocode:

```
$AprioriGenerateRules(D, F, MIN_{SUP}, MIN\_CONF)$ {
  $R \leftarrow \phi;$
  $for(each\, I \in F)$ {
    $R \leftarrow R \cup I \Rightarrow \phi;$
    $C_1 \leftarrow \{\{1\} \mid i \in I\};$
    $k \leftarrow 1$
    $while(C_k \neq \phi)\{$
      $H_k \leftarrow \{X \in C_k | confidence(I - X \Rightarrow X) \geq MIN\_CONF\};$
      $for(any\, X, Y \in H_k,\ X[i] == Y[i]\, for\, 1 \leq i \leq k-1, and\, X[k] < Y[k]))\{$
        $I \leftarrow X \cup \{Y[k]\};$
        $if(\forall J \subseteq I, |J| == k, J \in H_k)\{$
          $C_{k+1} \leftarrow C_{k+1} \cup I$
        }
      }
      $k \leftarrow k + 1;$
    }
    $R \leftarrow R \cup \{I - X \Rightarrow X | X \in H_1 \cup \ldots \in H_k\};$
}
```

The R implementation

R code of the algorithm to generate A-Priori association is as follows:

```
Here is the R source code of the main
  algorithm:
AprioriGenerateRules  <- function
  (D,F,MIN_SUP,MIN_CONF){
```

```
  #create empty rule set
  r <- CreateRuleSets()
  len4f <- length(F)
  for(idx in 1:len4f){
     #add rule F[[idx]] => {}
     AddRule2RuleSets(r,F[[idx]],NULL)
     c <- list()
     c[[1]] <- CreateItemSets(F[[idx]])
     h <- list()
     k <-1
     while( !IsEmptyItemSets(c[[k]]) ){
       #get heads of confident association rule in c[[k]]
       h[[k]] <- getPrefixOfConfidentRules(c[[k]],
       F[[idx]],D,MIN_CONF)
       c[[k+1]] <- CreateItemSets()

       #get candidate heads
       len4hk <- length(h[[k]])
       for(jdx in 1:(len4hk-1)){
          if( Match4Itemsets(h[[k]][jdx],
             h[[k]][jdx+1]) ){
             tempItemset <- CreateItemset
             (h[[k]][jdx],h[[k]][jdx+1][k])
             if( IsSubset2Itemsets(h[[k]],
               tempItemset) ){
               Append2ItemSets(c[[k+1]],
               tempItemset)
         }
       }
     }
   }
   #append all new association rules to rule set
   AddRule2RuleSets(r,F[[idx]],h)
   }
  r
}
```

To verify the R code, Arules and Rattle packages are applied while verifying the output.

> Arules (*Hahsler et al., 2011*) and Rattle packages provide support for association rule analysis. AruleViz is used to visualize the output's association rules.

Hybrid association rules mining

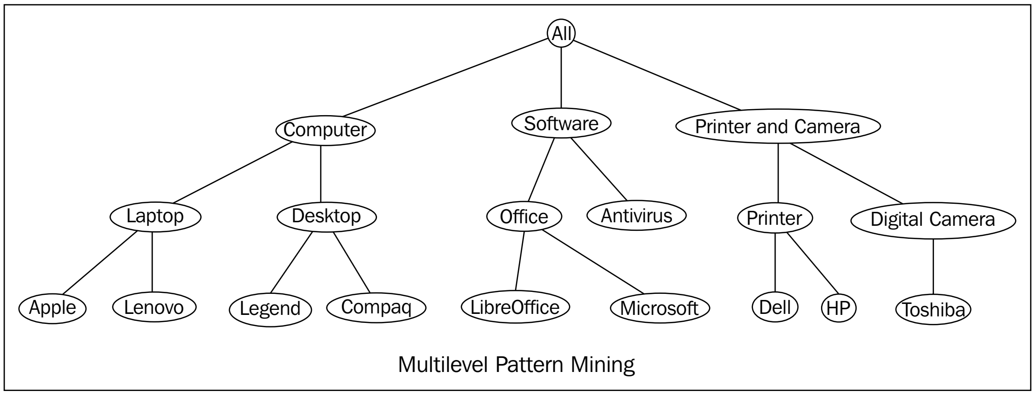
There are two interesting applications of association rules mining: one is **multilevel and multidimensional association rules mining**, while the other is **constraint-based mining**.

Mining multilevel and multidimensional association rules

For a given transactional dataset, if there is a conceptual hierarchy that exists from some dimensions of the dataset, then we can apply multilevel association rules mining to this dataset. Any association rules mining algorithm applicable to the transaction dataset can be used for this task. The following table shows an example from the Amazon store:

TID	Item purchased
1	Dell Venue 7 16 GB Tablet, HP Pavilion 17-e140us 17.3-Inch Laptop...
2	Samsung Galaxy Tab 3 Lite, Razer Edge Pro 256GB Tablet…
2	Acer C720P-2666 Chromebook, Logitech Wireless Combo MK270 with Keyboard and Mouse…
2	Toshiba CB35-A3120 13.3-Inch Chromebook, Samsung Galaxy Tab 3 (7-Inch, White)…

Have a look at the following flowchart that explains multilevel pattern mining:



Multilevel Pattern Mining

Based on the conceptual hierarchy, lower-level concepts can be projected to higher-level concepts, and the new dataset with higher-level concepts can replace the original lower-level concepts.

The support counts are calculated at each conceptual level. Many A-Priori-like algorithms are designed with slightly different treatment to support count; here is a possible list of treatments available:

- A uniform minimum support threshold is used across all the levels
- Reduced minimum support threshold is used for lower levels
- Group-based minimum support threshold

> Sometimes the A-Priori property is not always held here. There are some exceptions.
>
> Multilevel association rules are mined from multiple levels of the conceptual hierarchy.

Constraint-based frequent pattern mining

Constraint-based frequent pattern mining is a heuristic method with some user-specified constraints to prune the search space.

The ordinary constraints are, but not limited to, the following:

- Knowledge-type constraint (specifies what we are going to mine)
- Data constraint (limits to the original dataset)
- Dimension-level constraints
- Interestingness constraints
- Rule constraints

Mining sequence dataset

Sequential pattern mining is the major task for sequence dataset mining. The A-Priori-life algorithm is used to mine sequence patterns that use the A-Priori-life algorithm, which applies a breath-first strategy. However, for the pattern-growth method, a depth-first strategy is used instead. The algorithm sometimes integrates with constraints for various reasons.

The common purchase patterns of the customers of the store can be mined from sequential patterns. In other aspects, especially advertisement or market campaign, sequential patterns play an important role. The individual customer's behavior can be predicted from sequential patterns in the domain of web log mining, web page recommendation system, bioinformatics analysis, medical treatment sequence track and analysis, and disaster prevention and safety management.

The rules in this chapter, which are mined from sequence patterns, are of many types. Some of them are listed as follows:

- A **sequential rule** is $X \rightarrow Y$, where $X \subset Y$
- A **label sequential rule** (**LSR**) is of the form $X \rightarrow Y$, where *Y* is a sequence, and *X* a sequence generated from *Y* by replacing some of its items with wildcards
- A **class sequential rule** (**CSR**) is defined as *X* if:

$$X \rightarrow y, given\ S\ as\ a\ sequence\ dataset, I\ as\ the\ set\ of\ all\ items \in$$
$$S, Yas\ the\ set\ of\ all\ class\ lables, I \bigcap Y = \phi, X\ as\ a\ sequence, y \in Y,$$

Sequence dataset

A sequence dataset S is defined as a set of tuples, *(sid, s)*, in which sid is a sequence ID, and s is a sequence.

The support of a sequence X in a sequence dataset S is the number of tuples in S, which contains `X`: $support_S(X) = \{(sid,s) \vee (sid,s) \in S \leftarrow X \subseteq s\}$.

Here is a property intrinsic to sequential patterns, and it is applied to related algorithms such as the A-Priori property for the A-Priory algorithm. For a sequence `X` and its subsequence `Y`, $support(X) \leq support(Y)$.

The GSP algorithm

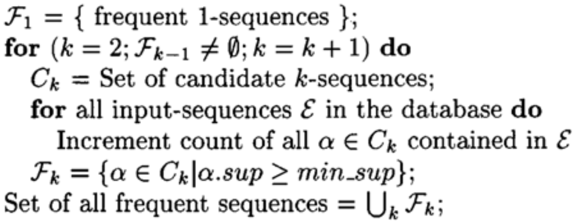
The **generalized sequential patterns** (**GSP**) algorithm is an A-Priori-like algorithm, but it is applied to sequence patterns. It is a level-wise algorithm and has a breadth-first approach. Here is the feature list:

- GSP is an extension of the A-Priori algorithm

  It uses the A-Priori property (downward-closed), that is, given the minimum support count, if a sequence is not accepted, all its super sequence will be discarded.

- The features require multiple passes of the initial transaction dataset
- It uses the horizontal data format
- In each pass, the candidate's set is generated by a self-join of the patterns found in the previous pass
- In the k-pass, a sequence pattern is accepted only if all its *(k-1)* subpatterns are accepted in the *(k-1)* pass

The overview of GSP algorithm goes here.



```
$\mathcal{F}_1$ = { frequent 1-sequences };
for ($k = 2; \mathcal{F}_{k-1} \neq \emptyset; k = k + 1$) do
   $C_k$ = Set of candidate $k$-sequences;
   for all input-sequences $\mathcal{E}$ in the database do
      Increment count of all $\alpha \in C_k$ contained in $\mathcal{E}$
   $\mathcal{F}_k = \{\alpha \in C_k | \alpha.sup \geq min\_sup\}$;
Set of all frequent sequences = $\bigcup_k \mathcal{F}_k$;
```

Here is the pseudocode:

```
$GSP(D, \Sigma, MIN\_SUP)$ {
  $F \leftarrow \phi$;
  $C^{(1)} \leftarrow \{\phi\}$;
  $for(each\, s \in \Sigma)$ {
    $in\, C^{(1)}, add\, s\, as\, child\, of\, \_\phi$
    $support\_count(s) \leftarrow 0$
  }
  $k \leftarrow 1$
$while(C^{(k)} = \phi)\{$
    $ComputSupport(C^{(k)}, D)$
    $for(each leaf\, r \in C^{(k)})\{$
      $if(support\_count(r) \geq MIN\_SUP)\{$
        $F \leftarrow F \cup \{(r, support\_count(r))\}$
      $\}else\{$
        $remove r\, from\, C^{(k)}$
      }
    }
    $C^{(k+1)} \leftarrow ExtendPrefixTree(C^{(k)})$;
    $k \leftarrow k + 1$;
  }
  $return\, F$;
}
$ComputeSupport(C^{(k)}, D)\{$
  $for(each\, s_i \in D)\{$
    $for(each\; r \in C^{(k)})\{$
      $if(r \subseteq s_i)\{$
        $support_{count(r)} \leftarrow support_{count(r)} + 1$
      }
```

```
$ComputeSupport\left(C^{(k)}, D\right)\{$
  $for\left(each\, s_i \in D\right)\{$
    $for\left(each\ r \in C^{(k)}\right)\{$
      $if\left(r \subseteq s_i\right)\{$
        $support_{count(r)} \leftarrow support_{count(r)} + 1$
      $\}$
    $\}$
  $\}$
$\}$
$ExtendPrefixTree\left(C^{(k)}\right)\{$
  $for\left(each\, leaf\ r_a \in C^{(k)}\right)\{$
    $for\left(each\, leaf\ r_b \in Children\left(Parent\left(r_a\right)\right)\right)\{$
      $r_{ab} \leftarrow r_a + r_b\left[k\right]$
      $if\left(r_c \in C^{(k)}, \forall r_c \subset r_{ab}, \left|r_c\right| = \left|r_{ab}\right| + 1\right)\{$
        $add\ r_{ab}\, as\, child\ of\ r_a\ with\, support\ \_\ count\left(r_{ab}\right) \leftarrow 0$
      $\}$
    $\}$
    $if\left(no\, extensions\ from\, r_a\right)\{$
      $remove\, r_a\ from\, C^{(k)}$
    $\}$
  $\}$
  $return\, C^{(k)}$
$\}$
```

The R implementation

Here is the R source code of the main algorithm:

```
GSP  <- function (d,I,MIN_SUP){
  f <- NULL
  c[[1]] <- CreateInitalPrefixTree(NULL)
  len4I <- GetLength(I)
  for(idx in 1:len4I){
    SetSupportCount(I[idx],0)
```

```
      AddChild2Node(c[[1]], I[idx],NULL)
    }
    k <- 1
    while( !IsEmpty(c[[k]]) ){
       ComputeSupportCount(c[[k]],d)
       while(TRUE){
         r <- GetLeaf(c[[k]])
         if( r==NULL ){
           break
         }
         if(GetSupport(r)>=MIN_SUP){
           AddFrequentItemset(f,r,GetSupport(r))
         }else{
           RemoveLeaf(c[[k]],s)
         }
      }
      c[[k+1]] <- ExtendPrefixTree(c[[k]])
      k <- K+1
    }
    f
  }
```

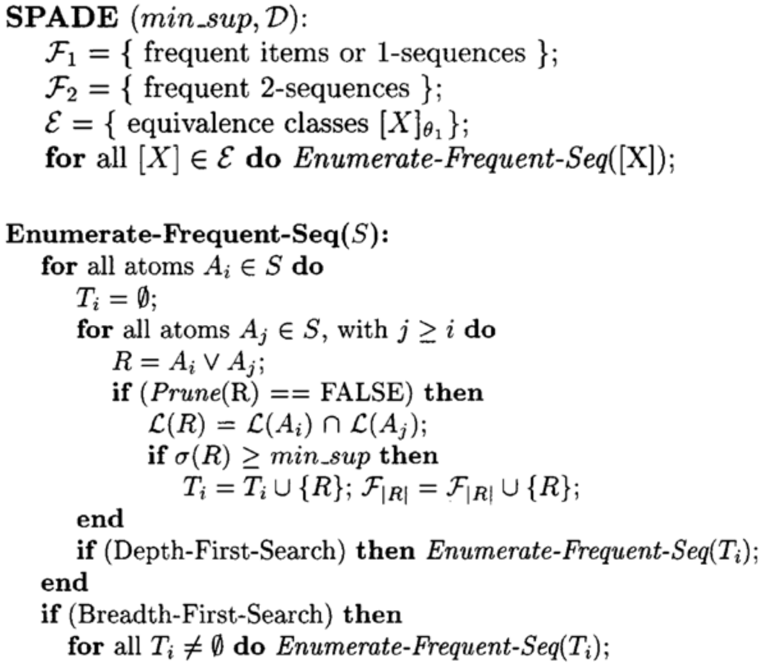
The SPADE algorithm

Sequential Pattern Discovery using Equivalent classes (**SPADE**) is a vertical sequence-mining algorithm applied to sequence patterns; it has a depth-first approach. Here are the features of the SPADE algorithm:

- SPADE is an extension of the A-Priori algorithm
- It uses the A-Priori property
- Multiple passes of the initial transaction data set are required
- The vertical data format is used
- It uses a simple join operation
- All sequences are found in three dataset passes

The short description of SPADE algorithm goes here.

Here is the pseudocode before calling the SPADE algorithm, $F \leftarrow \phi, \wedge k \leftarrow 0, P \leftarrow \left\{< s, L(s)\, s >\in \sum, \text{support\_count}(s) \geq \text{MIN\_SUP}\right\}$:



```
SPADE ($min\_sup, \mathcal{D}$):
  $\mathcal{F}_1$ = { frequent items or 1-sequences };
  $\mathcal{F}_2$ = { frequent 2-sequences };
  $\mathcal{E}$ = { equivalence classes $[X]_{\theta_1}$ };
  for all $[X] \in \mathcal{E}$ do Enumerate-Frequent-Seq([X]);

Enumerate-Frequent-Seq($S$):
  for all atoms $A_i \in S$ do
    $T_i = \emptyset$;
    for all atoms $A_j \in S$, with $j \geq i$ do
      $R = A_i \vee A_j$;
      if (Prune(R) == FALSE) then
        $\mathcal{L}(R) = \mathcal{L}(A_i) \cap \mathcal{L}(A_j)$;
        if $\sigma(R) \geq min\_sup$ then
          $T_i = T_i \cup \{R\}$; $\mathcal{F}_{|R|} = \mathcal{F}_{|R|} \cup \{R\}$;
    end
    if (Depth-First-Search) then Enumerate-Frequent-Seq($T_i$);
  end
  if (Breadth-First-Search) then
    for all $T_i \neq \emptyset$ do Enumerate-Frequent-Seq($T_i$);
```

The R implementation

Here is the R source code of the main algorithm:

```
SPADE  <- function (p,f,k,MIN_SUP){
  len4p <- GetLength(p)
  for(idx in 1:len4p){
     AddFrequentItemset(f,p[[idx]],GetSupport(p[[idx]]))
     Pa <- GetFrequentTidSets(NULL,MIN_SUP)
     for(jdx in 1:len4p){
       xab <- CreateTidSets(p[[idx]],p[[jdx]],k)
       if(GetSupport(xab)>=MIN_SUP){
         AddFrequentTidSets(pa,xab)
       }
     }
```

```
        if(!IsEmptyTidSets(pa)){
          SPADE(p,f,k+1,MIN_SUP)
        }
    }
  }
```

Rule generation from sequential patterns

Sequential rules, label sequential rules, and class sequential rules can be generated from sequential patterns, which you will get from the previous sequential patterns discovery algorithms.

High-performance algorithms

Along with the growth of the dataset size, there is a steady requirement for high-performance associations/patterns mining algorithms.

With the introduction of Hadoop and other MapReduce-like platforms to the world, there is a chance to meet these requirements. I will discuss this further in the upcoming chapters. Depending on the size of the dataset, some algorithms should be revised and adjusted, such as the recursive algorithm that will eventually run out of space on the call stack and might present a challenge when converting to MapReduce.

Time for action

To enhance your knowledge about this chapter, here are some practice questions that'll let you understand the concepts better:

- Write an R program to find how many unique items' names are contained in the given sample market basket transaction file. Map each item's name to a unique integer ID. Find out all the closed frequent itemsets. Find out all the maximal frequent itemsets and their support count. Set a support count threshold to various values yourself.
- Write an R program to implement the A-PrioriTid algorithm.

Summary

In this chapter, we looked at the following topics:

- Market basket analysis
- As the first step of association rule mining, the frequent itemset is the key factor. Along with the algorithm design, closed itemsets and maximum frequent itemsets are defined too.
- As the target of association rule mining, association rules are mined with the measure of support count and confidence. Correlation rules mining are mined with the correlation formulae, in addition to the support count.
- Monotonicity of frequent itemset; if an itemset is frequent, then all its subsets are frequent.
- The A-Priori algorithm, which is the first efficient mining algorithm to mine frequent patterns; many variants originated from it.
- Frequent patterns in sequence.

The next chapter will cover the basic classification algorithms, which is a major application of data mining, including ID3, C4.5, and CART.

3
Classification

In this chapter, you will learn the popular classification algorithms written in the R language. Empirical classifier performance and accuracy benchmarks are also included. Along with the introduction of various classification algorithms, b will also learn various ways to improve the classifier and so on.

Classification has massive applications in modern life. With the exponential growth of the information dataset, there is a need for high performance classification algorithms to judge an event/object belonging to a predefined categories set. Such algorithms have unlimited opportunity for implementation in a wide variety of industries such as bioinformatics, cybercrime, and banking. Successful classification algorithms use predefined categories from training information datasets to predict the unknown category for a single event given a common set of features.

Along with the continual growth of computer science, the classification algorithms need to be implemented on many diverse platforms including distributed infrastructure, cloud environment, real-time devices, and parallel computing systems.

In this chapter, we will cover the following topics:

- Classification
- Generic decision tree introduction
- High-value credit card customers classification using ID3
- Web spam detection using C4.5
- Web key resource page judgment using CART
- Trojan traffic identification method and Bayes classification
- Spam e-mail identification and Naïve Bayes classification
- Rule-based classification and the player types in computer games

Classification

Given a set of predefined class labels, the task of classification is to assign each data object of the input dataset with a label using the classifier's training model. Typically, the input can be a discrete or continuous value, but the output is discrete binary or nominal value and so forth. Classification algorithms are often described as learning models or functions, in which `x` is a tuple of attribute set with discrete or continuous value, and y is an attribute with discrete value such as categorical labels.

$$f(x,y)=0, x=(x_1,x_2,\dots,x_n)$$

This function can also be treated as a classification model. It can be used to distinguish objects belonging to different classes or to predict the class of a new tuple or `y` in the above `(x, y)`. In another point of view, classification algorithms are targeted to find a model from the input data, and apply this model to future classification usage predictions when given a common set of attributes.

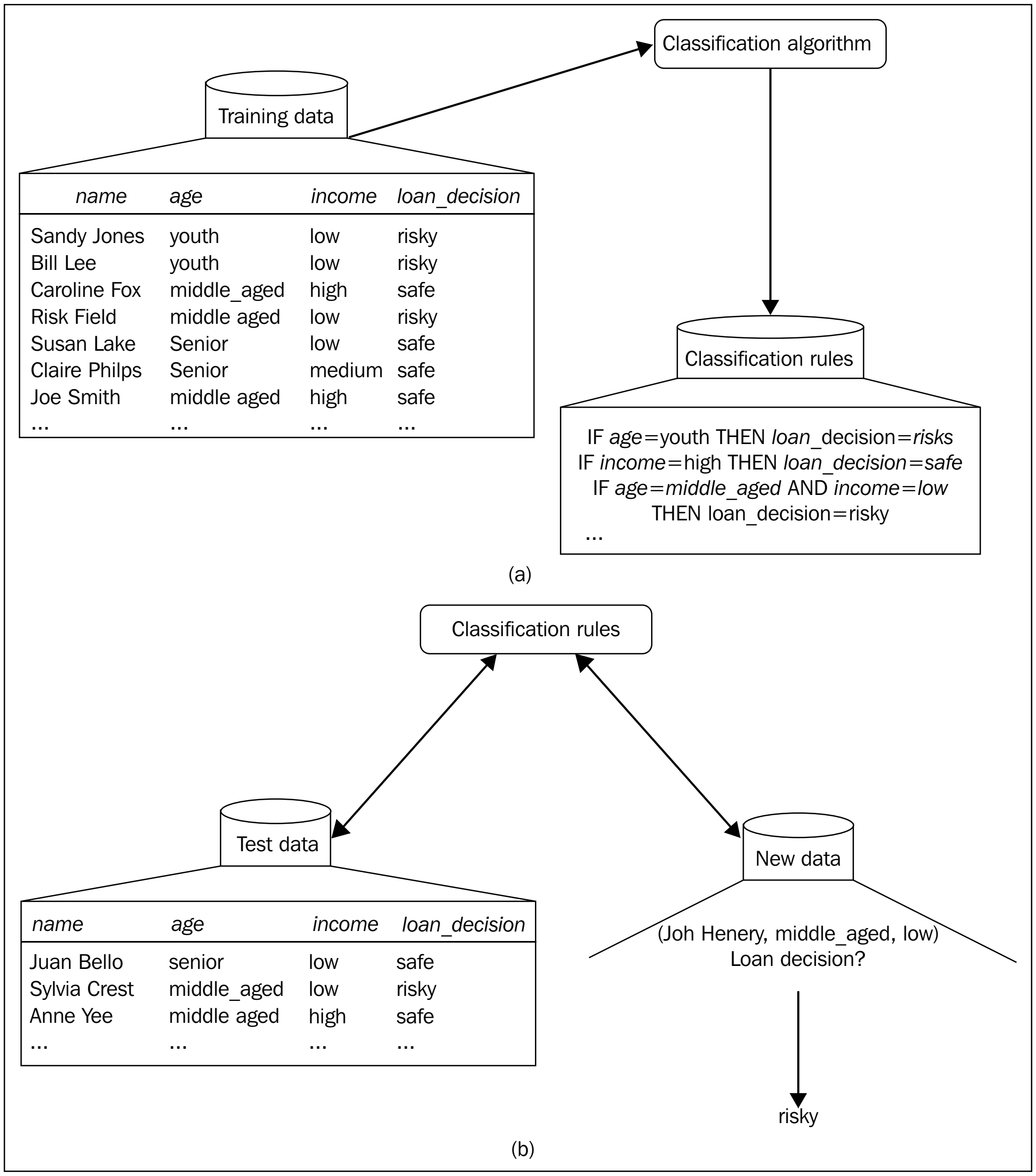
Generally speaking, $x=(x_1,x_2,\dots,x_n)$ is a set of attributes selected as the input for the classification system. There are special algorithms used to select only the useful attributes from this set to ensure the efficiency of the classification system.

Almost any classification tasks need this preprocessing procedure, but the exact means vary from case to case. Here are three mainstream methods applied:

- Data cleaning
- Relevance analysis
- Data transformation and reduction

A standard classification process often includes two steps. The classification model with the higher accepted accuracy is accepted as classifier to classify a dataset in production. The following two steps are illustrated with an example in the diagram:

- **Training** (**supervised learning**): The classification model is built upon the training dataset, that is, the (instance, class label) pairs
- **Classification validation**: The accuracy of the model is checked with the test dataset to decide whether to accept the model



name	age	income	loan_decision
Sandy Jones	youth	low	risky
Bill Lee	youth	low	risky
Caroline Fox	middle_aged	high	safe
Risk Field	middle aged	low	risky
Susan Lake	Senior	low	safe
Claire Philps	Senior	medium	safe
Joe Smith	middle aged	high	safe
...	...	...	...

IF *age*=youth THEN *loan*_decision=*risks*
IF *income*=high THEN *loan_decision*=*safe*
IF *age*=*middle_aged* AND *income*=*low*
THEN loan_decision=risky
...

(a)

name	age	income	loan_decision
Juan Bello	senior	low	safe
Sylvia Crest	middle_aged	low	risky
Anne Yee	middle aged	high	safe
...	...	...	...

(Joh Henery, middle_aged, low)
Loan decision?

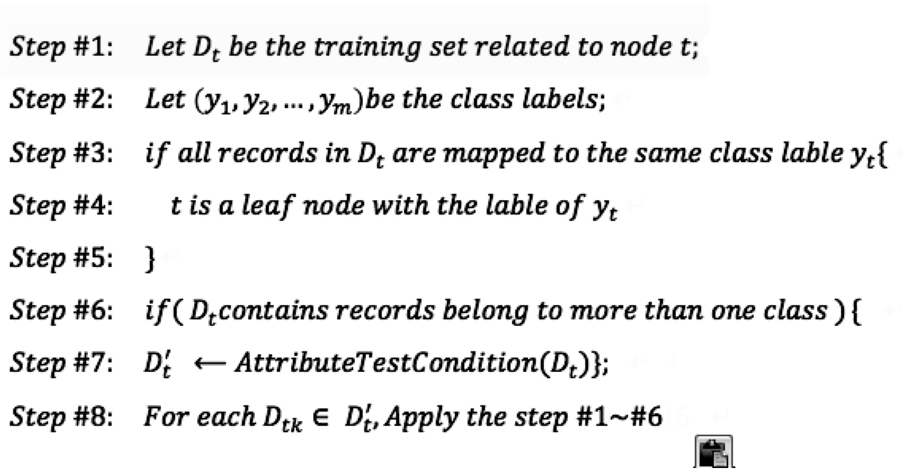
risky

(b)

In the following sections, we will introduce some classification algorithms with different designs.

Generic decision tree induction

There are various definitions of the term **decision tree**. Most commonly, a decision tree provides a representation of the process of judging the class of a given data instance or record from the root node down to some leaf node. As a major classification model, the decision tree induction builds a decision tree as a classification model using the input dataset and class label pairs. A decision tree can be applied to various combinations of the following attribute data types, but is not limited to, including nominal valued, categorical, numeric and symbolic data, and their mixture. The following list is an illustration of Hunt's decision tree definition. The Step #7 applies a selected attribute test condition to partition the records to smaller datasets.



```
Step #1:  Let $D_t$ be the training set related to node $t$;
Step #2:  Let $(y_1, y_2, \dots, y_m)$ be the class labels;
Step #3:  if all records in $D_t$ are mapped to the same class lable $y_t$ {
Step #4:      t is a leaf node with the lable of $y_t$
Step #5:  }
Step #6:  if ( $D_t$ contains records belong to more than one class ) {
Step #7:  $D'_t \leftarrow AttributeTestCondition(D_t)$};
Step #8:  For each $D_{tk} \in D'_t$, Apply the step #1~#6
```

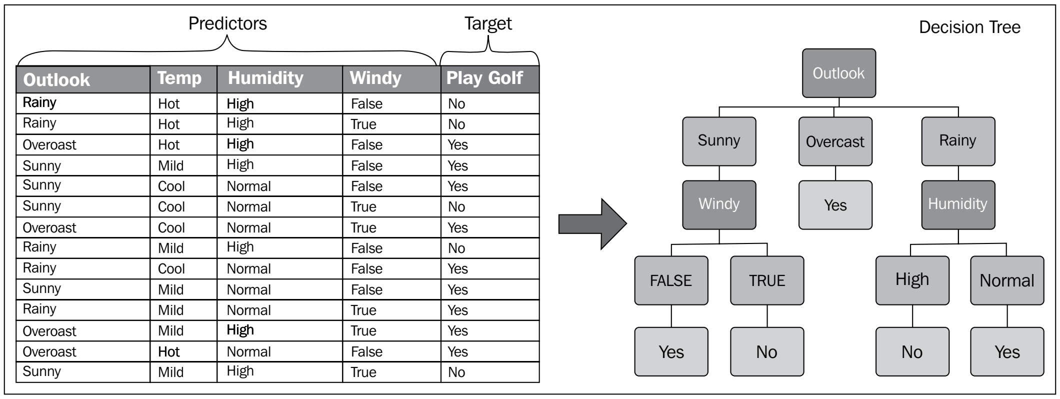
The decision tree is popular for its simplicity and low computational effort compared to other algorithms. Here are some characteristics of the decision tree induction:

- The greedy strategy is usually applied to the decision tree.
- It infers a decision tree once upon the entire training dataset.
- The algorithm requires no parameters to obtain the classification model from the input dataset.
- Like many other tasks, finding an optimal decision tree is an NP-complete problem.
- The algorithm to build the decision tree enables construction of the decision tree quickly. Tree construction is efficient, even upon large datasets.
- It provides an expressive way for discrete-valued functions.
- It is robust while opposed to noise.

- Using a top-down, recursive partition, the divide-and-conquer strategy is applied to most of the decision tree algorithms.
- The size of the sample dataset usually decreases dramatically when traversed down the tree.
- A subtree can be replicated many times in the decision tree.
- The test condition usually contains only one attribute.
- The performance of decision tree algorithms is affected by the impurity measure.

It is time to consider the decision tree when the instances in the source dataset are describable by the attribute-value pair, and the target function has discrete values, while the training dataset possibly has some noise.

An example of a decision tree built with the input dataset in a table (classical play golf dataset) is shown in the following diagram. The decision tree is composed of three entities or concepts: the root node, the internal node, and the leaf node. The leaf is given a class label. Nodes other than the leaf conduct tests on the attribute set to determine which input data belongs to which branch (child) of the node.



Predictors				Target
Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overoast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overoast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overoast	Mild	High	True	Yes
Overoast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

Given a built decision tree, a test record can be classified easily. From the root node, apply the test condition of that node to the test record and go to the next node with the corresponding test result until a leaf, by which we can decide which class the test record belongs to, is reached.

Now there will be two issues. One is how to divide the training set as per a certain node while the decision induction tree grows according to a chosen test condition upon various attribute sets. This will be a question related to attribute selection measure, which are illustrated in the following section. The second but important issue is related to model overfitting.

There are two strategies for the termination of the growth of the limiting decision induction tree node. Using the naïve strategy, for a certain node, when all the data objects within a node are assigned to it belong to the same class or all records with the same attribute values; as a result, the node related will be assigned with the class label as the majority of training records within that node. The second strategy terminates the algorithm earlier, which is meant to avoid model overfitting and will be introduced in the tree pruning section.

Attribute selection measures

The node can have more than two children or branches depending on the attribute test condition and the selected attribute. To split the node, attribute selection measures with various implementations are applied. Attribute selection measures within the same node may also vary for binary branches or multiway branches. Some common attribute selection measures are the following:

- **Entropy**: This concept is used in information theory to describe the impurity of an arbitrary collection of data. Given the target attribute class set with size of *c*, and p_i as the proportion/probability of *S* belonging to class *i*, the definition is here, and the definition *Gain* is shown in the next point. Entropy always means how disordered a dataset is. The higher the value of entropy, the more the uncertainty shown by the source dataset.

$$Entropy\left(S\right)=\sum_{i=1}^{c}-p_{i}log_{2}p_{i}$$

  The size and coverage of the training set assigned to a certain node affect the correctness of the following equations. The gain is better for those situations.

- **Gain**:

$$Gain\left(S,A\right)=Entropy\left(S\right)-\sum_{\upsilon\in Values\left(A\right)}\frac{|S_{\upsilon}|}{|S|}Entropy\left(S_{\upsilon}\right)$$

- **Gain Ratio**: This is applied in the C4.5 classification algorithm using the following formula:

$$GainRatio\left(S,T\right)=\frac{Gain\left(S,T\right)}{SplitInfo\left(S,T\right)}$$

- **Information Gain**: The ID3 algorithm uses this statistical property to decide which attribute is selected to be tested at any node in the tree, and measures the association between inputs and outputs of the decision tree.

  With the concept of information gain, the definition of a decision tree can be thought of in this way:

  - A decision tree is a tree-structured plan that uses a set of attribute tests to predict output
  - To decide which attribute should be tested first, simply find the one with the highest information gain
  - It, then, recurs

- **Gini Index**: It is used in the CART classification algorithm. The Gini index for a specific split point is calculated using the following equation. It is used to gauge the purity of the split point.

$$G(D) = 1 - \sum_{i=1}^{k} P(c_i \mid D)^2$$

- **Split Info**:

$$SplitInfo(S,T) = -\sum_{\upsilon \in Values(T_S)} \frac{|T_{S,\upsilon}|}{|T_S|} * log \frac{|T_{S,\upsilon}|}{|T_S|}$$

Tree pruning

The initial decision tree is often built with many branches reflecting outliers or noise, which are also common causes of model overfitting. Usually, the direct consequent in tree pruning is needed for the after-dealt of decision tree aiming, which is required for classifying higher accuracy or lower error rates. The two types of pruning in production are as follows:

- **Post-pruning**: This approach is to perform tree pruning after the tree grows to the maximum form. The cost-complexity pruning algorithm used in CART and the pessimistic pruning in C4.5 are both examples of post-pruning.
- **Pre-pruning**: This is also known as the early stopping strategy, which avoids an over-matured tree generation, to stop the growth of a tree earlier using additional restrictions, such as a threshold.

Repetition and replication are the two major factors that make decision trees unreasonably large and inefficient.

General algorithm for the decision tree generation

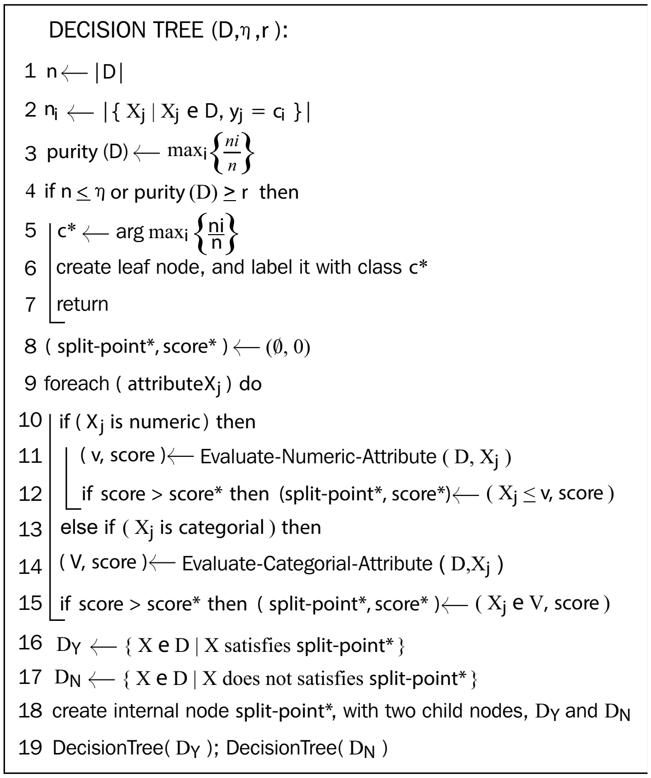
Here is the pseudocode of the general decision induction tree algorithm:

```
$TreeGrowth(E,F)$ {
  $if\left(StoppingCondition(E,F)\right)$ {
    $leaf \leftarrow CreateNode()$
    $leaf.label \leftarrow Classify(E)$
    $return\,leaf$
  }$else${
    $root \leftarrow CreateNode()$
    $root.test\_cond \leftarrow FindBestSplit(E,F)$
    $V \leftarrow \{v \mid v\,is\,a\,possible\,outcome\,of\,root.test\_cond\}$
    $for\left(each\,v \in V\right)$ {
      $E_v \leftarrow \{e|root.test\_cond(e) = c,\ and\ e \in E\}$
      $child \leftarrow TreeGrowth(E_v,F)$
      $AddDescendentAndLabelEdge(root,child,v)$
    }
  }
  $return\,root;$
}
```

Another variety of the algorithm is described here, with input parameters as follows:

- D denotes the training data
- The leaf size is defined by η
- The leaf purity threshold is defined by π

The output of the algorithm is a decision tree, as shown in the following screenshot:



```
    DECISION TREE (D,$\eta$,r ):
1  n $\longleftarrow$ |D|
2  $n_i \longleftarrow |\{ X_j \mid X_j$ e D, $y_j = c_i \}|$
3  purity (D) $\longleftarrow \max_i \left\{\frac{ni}{n}\right\}$
4  if n $\leq \eta$ or purity (D) $\geq$ r  then
5      c* $\longleftarrow$ arg $\max_i \left\{\frac{ni}{n}\right\}$
6      create leaf node, and label it with class c*
7      return
8  ( split-point*, score* ) $\longleftarrow (\emptyset, 0)$
9  foreach ( attribute$X_j$ ) do
10     if ( $X_j$ is numeric) then
11         ( v, score ) $\longleftarrow$ Evaluate-Numeric-Attribute ( D, $X_j$ )
12         if score > score* then (split-point*, score*) $\longleftarrow$ ( $X_j \leq$ v, score )
13     else if ( $X_j$ is categorial ) then
14     ( V, score ) $\longleftarrow$ Evaluate-Categorial-Attribute ( D,$X_j$ )
15     if score > score* then ( split-point*, score* ) $\longleftarrow$ ( $X_j$ e V, score )
16 $D_Y \longleftarrow$ { X e D | X satisfies split-point* }
17 $D_N \longleftarrow$ { X e D | X does not satisfies split-point* }
18 create internal node split-point*, with two child nodes, $D_Y$ and $D_N$
19 DecisionTree( $D_Y$ ); DecisionTree( $D_N$ )
```

Line 1 denotes the partition size. Line 4 denotes the stopping condition. Line 9 through line 17 try to get the two branches with the new split. Finally, line 19 applies the algorithm recursively on the two new subbranches to build the subtree. This algorithm is implemented with R in the following section.

The R implementation

The main function of the R code for the generic decision tree induction is listed as follows. Here `data` is the input dataset, `c` is the set of class labels, `x` is the set of attributes, and `yita` and `pi` have the same definitions as in the previous pseudocodes:

```
DecisionTree <- function(data,c,x,yita,pi){
        result.tree <- NULL
        if( StoppingCondition(data,c,yita,pi) ){
                result.tree <- CreateLeafNode(data,c,yita,pi)
                return(result.tree)
        }

        best.split <- GetBestSplit(data,c,x)
        newdata <- SplitData(data,best.split)

        tree.yes <- DecisionTree(newdata$yes,c,x,yita,pi)
        tree.no <- DecisionTree(newdata$no,c,x,yita,pi)
        result.tree <- CreateInternalNode(data,
                   best.split,tree.yes,tree.no)

        result.tree
}
```

One sample dataset is chosen to verify the generic decision tree induction algorithm, the weather dataset. It is from the R package Rattle, which contains 366 examples of 23 attributes, and one target or the class label. In the R language, weather is a data frame, which contains 366 observations of 24 variables. The details for the dataset can be retrieved with the following R code:

```
> Library(rattle)
> str(weather)
```

High-value credit card customers classification using ID3

The **Iterative Dichotomiser 3** (**ID3**) algorithm is one of the most popular designs of the decision induction tree. It is not tolerant of missing values or noisy, and the value of attributes must come from an infinite fixed set.

ID3 uses entropy to calculate the homogeneity of a sample and also for the split. The information gain `G` for each attribute `A` is computed using the following equation. The root of the final tree is assigned with an attribute with the highest information gain. Then the new subtree is built recursively upon each value of the attribute bound to the root.

$$Gain\left(S,A\right)=Entropy\left(S\right)-\sum_{\upsilon\in Values\left(A\right)}\frac{|S_\upsilon|}{|S|}Entropy\left(S_\upsilon\right)$$

$$Entropy(S) = \sum_{i=1}^{c} - p_i log_2 p_i$$

With the play golf dataset as the input dataset, you can calculate the information gain and list using the following formulas:

- *Entropy (root) = 0.940*
- *Gain () = 0.048, Gain (S, Humidity) = 0.151*
- *Gain (S, Temperature) = 0.029, Gain (S, Outlook) = 0.246*

ID3 (C4.5 and CART) builds the decision induction tree recursively in a top-down divide-and-conquer manner through the space of possible decision trees with a greedy strategy. Using the greedy search strategy, at each step, a decision that greatly improves the optimizing target is made. For each node, find the test condition best segment the training data assigned to it.

The characteristics of the decision induction tree in the case of ID3 include the following:

- Each node excluding the leaf of the tree corresponds to an input attribute, each arc to a possible value of that attribute
- Entropy is used to determine how informative a particular input attribute is about the output class on a given dataset
- The recursive algorithm

A quick description about the recursive algorithm can be defined as follows:

- Break the original problem into two or more smaller-sized problems with the same type
- Call the recursive algorithm on each smaller type of problem
- Group together the results of step 2 to solve the original problem

The ID3 algorithm

The input parameters for ID3 algorithm are as follows:

- *I*, denotes the set of input attributes, which may be tested by the result decision tree
- *T*, the set of training data objects, or training examples

The output parameter of the algorithm is as follows:

- O, denotes the set of output attribute, that is, the value of those attributes will be predicted by the tree

Here is the pseudocode of the general algorithm:

```
ID3(I,O,T) {
  if (T is emputy) {
    node ← CreateNode()
    node.label ← "Failure"
    return node
  }
  if (all instance in T share the save value "sameValue" for O) {
    node ← CreateNode()
    node.label ← "sameValue"
    return node
  }
  if (I is emputy) {
    node ← CreateNode()
    node.label ← the value of the most common value of O in T
    return node
  }
Compute the information gain for each attribute in I relative to T
let x be the attribute with lestG$(x,T)$ of the attribute in I
let $\{x_j | j \in [1,m]\}$ be the value set of x
let $\{T_j | j \in [1,m]\}$ be the set of subset of T after it is partitioned with value of x
root $\leftarrow$ CreateNode()
root.label $\leftarrow$ x
foreach( j in $[1,m]$ ) {
    create a new branch for root
    $childNode_j \leftarrow ID3(I - \{x\}, O, T_j)$
    if ( $T_j$ is empty ){
        leaf.$\leftarrow$CreateNode()
        node.label $\leftarrow$the value of the most common value of O in T
        add leaf as a leaf and the j – th child of root, under the new branch
    }else{
        add $childNode_j$ as the j – th child of root, under the new branch
    }
}
return root;
}
```

The R implementation

The main function of the R code for the ID3 algorithm is listed as follows. Here `data` is the input training dataset, `ix` is the set of input attributes, and `ox` is the output attribute:

```
ID3 <- function(data,ix,ox){
   result.tree <- NULL

   if( IsEmpty(data) ){
       node.value <- "Failure"
       result.tree <- CreateNode(node.value)
       return(result.tree)
   }
   if( IsEqualAttributeValue(data,ox) ){
       node.value <- GetMajorityAttributeValue(data,ox)
       result.tree <- CreateNode(node.value)
```

```
        return(result.tree)
    }
    if( IsEmpty(ix) ){
        node.value <- GetMajorityAttributeValue(data,ox)
        result.tree <- CreateNode(node.value)
        return(result.tree)
    }
    gain <- GetInformationGain(data,ix)
    best.split <- GetBestSplit(data,gain,ix)

    values <- GetAttributeValues(best.split)
    values.count <- GetAttributeValuesCount(best.split)
    data.subsets <- SplitData(data,best.split)

    node.value <- best.split
    result.tree <- CreateNode(node.value)
    idx <- 0
    while( idx<=values.count ){
        idx <- idx+1
        newdata <- GetAt(data.subsets,idx)
        value <- GetAt(values,idx)
        new.ix <- RemoveAttribute(ix,best.split)
        new.child <- ID3(newdata,new.ix,ox)
        AddChildNode(result.tree,new.child,value)
    }

    result.tree
}
```

Web attack detection

Along with the development of information technology, there have emerged many systems that identify malicious usage of the built software system, web system, and so on. One of them is the **Intrusion Detection System** (**IDS**), to detect the malicious behavior, conduct content inspection without the firewall. Also includes include signature detection, anomaly detection, and so on.

Classifier-like decision tree technologies, such as ID3, C4.5, and CART, play an important role as analyzers in addition to other important components of IDS, such as sensor, manager, operator, and administrator. The classifications needed here are activity monitor, file integrity checker, host firewall, log parser, and packet pattern matching.

Many issues occur for IDS. One of them is the new variety of a known attack pattern, often with low detection rate by the existing IDS. This drives the design of new types of IDS systems integrated with artificial intelligence, especially decision tree technologies.

Among real world examples, except the ones IDS has already built, there are also competitions for applying data mining techniques to web attack detection. One of them is KDD-Cup. The topic for KDD-Cup 1999 was Computer network intrusion detection, to build a classifier to predict the unauthorized behavior.

The dataset for it came from the DARPA Intrusion Detection Evaluation Program. More than five million data instances are contained in the training dataset and more than two million for test dataset. There are about 24 attack types in the training dataset, and 14 in the test dataset. Each data instance in the dataset contains 41 attributes, 9 for TCP connection, 13 for content features contained in the TCP connection, 9 for traffic features that use a two-second time window and the left for host-related traffic features. All the attacks can be categorized into the following four groups:

- **DOS**: This refers to denial of service
- **R2L**: This refers to unauthorized access to the local machine from the remote machine
- **U2R**: This refers to unauthorized access to local super-user privileges by a local unprivileged user
- **Probing**: This refers to surveillance and probing

By specific transformation, the ID3 algorithm can be applied to various web attack detection datasets with various sizes. When the size of the dataset increases, the performance of ID3 will be kept efficient by parallelization.

For simplicity, one example only takes the following four types of attacks to label a dataset for simple IDS:

- SQL Injection
- Cross Site Scripting
- Code Injection
- Directory Traversal

All the four types of attacks behave with a common pattern, the web queries with malicious pattern. Normalizing the web queries, the URL and collection of reserved tags, label-specific patterns with the appropriate label in the four types of attacks. After training ID3 on the dataset and applying it to the existing IDS, a better detection rate can be achieved.

High-value credit card customers classification

Following the growth of credit card usage, there has been a requirement in banking industry—finding high-value credit card customers from all customers to create a more customer-oriented strategy to increase profit. There are similar requirements such as finding interesting rules from the dataset.

To achieve this target, we need to enroll more correct customer attributes (no matter what type they are) to the training data object. The possible choices are transaction records, usage behaviors, customer age, annual income, education background, financial assets, and so on.

There is no need to include all customer-related attributes; the most key attributes on this target should be adopted. The domain experts might be helpful on this.

With the appropriate attributes selected, the ID3 algorithm can be applied here to finally extract sensitive features or representatives to help to judge which customer is more likely to be profitable.

Web spam detection using C4.5

C4.5 is an extension of ID3. The major extensions include handling data with missing attribute values, and handling attributes that belong to an infinite continuous range.

It is one of the decision tree algorithms, and is also a supervised learning classification algorithm. A model is learned and the input attribute values are mapped to the mutually exclusive class labels. Moreover, the learned model will be used to further classify new unseen instances or attribute values. The attribute select measure adopted in C4.5 is the gain ratio, which avoids the possible bias:

$$GainRatio\left(S,T\right)=\frac{Gain\left(S,T\right)}{SplitInfo\left(S,T\right)}$$

$$Gain(S, A) = Entropy(S) - \sum_{\upsilon \in Values(A)} \frac{|S_\upsilon|}{|S|} Entropy(S_\upsilon)$$

$$SplitInfo(S,T) = -\sum_{\upsilon \in Values\,(T_S)} \frac{|T_{S,\upsilon}|}{|T_S|} * log \frac{|T_{S,\upsilon}|}{|T_S|}$$

Based on the generic C4.5 algorithm, a suite for varieties derived, C4.5, C4.5-no-pruning, C4.5-rules, and so forth; all of them are called C4.5 algorithms, which means C4.5 is a suite of algorithms.

Compared to other algorithms, there are many important characteristics of C4.5:

- Tree pruning, a post-pruning strategy, is followed to remove some of the tree structure using selected accuracy criteria
- An improved use of continuous attributes
- Missing values handling
- Inducing rule sets
- It contains a multiway test that depends on the attribute value and is not just limited to the binary test
- Information theoretic tests are applied via the gain and gain ratio
- The greedy learning algorithm, that is, along with the tree growing, test best criteria test result is chosen
- The data fits in main memory (there are many extended algorithms that can use the secondary storages, such as BOAT, Rainforest, SLIQ, SPRINT, and so forth.)

The C4.5 algorithm

Here is the pseudocode for the basic C4.5 algorithm:

```
C4.5(T)
Input: training data set T; attributes S.
Output: decision tree Tree.
if T is NULL then
    return failure
end if
if S is NULL then
    return Tree as a single node with most frequent class label in T
end if
if |S| = 1 then
    return Tree as a single node S
end if
set Tree = {}
for a ∈ S do
    set $Info(a,T)$=0, and $SplitInfo(a,T)$= 0
    compute $Entropy$(a)
    for $v$ ∈ $values(a,T)$ do
        set $T_{a,v}$ as the subset of $T$ with attribute $a = v$
    $Info(a,T) += \frac{|Ta,v|}{|Ta|} Entroph(a_v)$
        $SplitInfo(a,T) += \frac{|Ta,v|}{|Ta|} \, log \, \frac{|Ta,v|}{|Ta|}$
end for
    $Gain(a,T) = Entropy(a) - Info(a,T)$
    $GainRatio(a,T) = \frac{Gain(a,T)}{SplitInfo(a,T)}$
end for
set $a_{best} = \underset{a}{argmax} \, \{GainRatio(a,T)\}$
attach $a_{best}$ into $Tree$
for $v$ ∈ $values(a_{best}, T)$ do
    call $C4.5(Ta,v)$
end for
return $Tree$
```

The R implementation

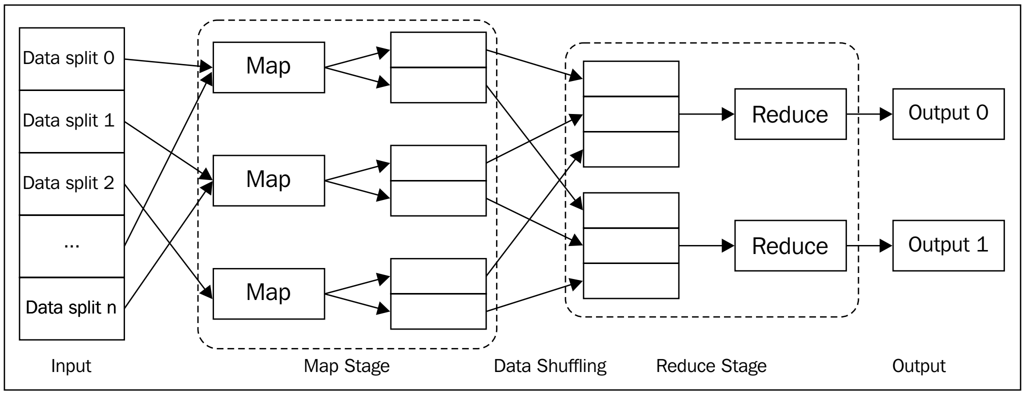
The R code for the ID3 is listed as follows:

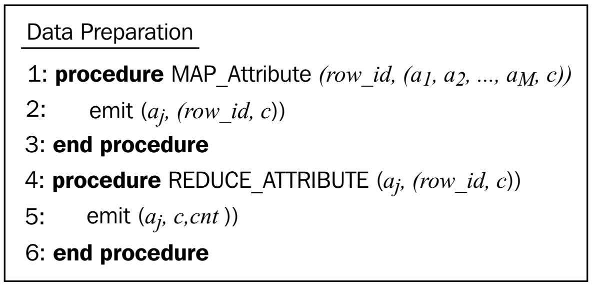
```
C45 <- function(data,x){
    result.tree <- NULL

    if( IsEmpty(data) ){
        node.value <- "Failure"
        result.tree <- CreateNode(node.value)
        return(result.tree)
    }
    if( IsEmpty(x) ){
        node.value <- GetMajorityClassValue(data,x)
        result.tree <- CreateNode(node.value)
        return(result.tree)
    }
    if( 1 == GetCount(x) ){
        node.value <- GetClassValue(x)
        result.tree <- CreateNode(node.value)
        return(result.tree)
    }

    gain.ratio <- GetGainRatio(data,x)
    best.split <- GetBestSplit(data,x,gain.ratio)

    data.subsets <- SplitData(data,best.split)
    values <- GetAttributeValues(data.subsets,best.split)
    values.count <- GetCount(values)

    node.value <- best.split
    result.tree <- CreateNode(node.value)
    idx <- 0
    while( idx<=values.count ){
        idx <- idx+1
        newdata <- GetAt(data.subsets,idx)
        value <- GetAt(values,idx)
        new.x <- RemoveAttribute(x,best.split)
        new.child <- C45(newdata,new.x)
        AddChildNode(result.tree,new.child,value)
    }

    result.tree
}
```

A parallel version with MapReduce

With the increase in the dataset volume or size, the C4.5 algorithm can be parallelized according to the MapReduce algorithm, the Hadoop technologies suite, and especially via RHadoop for R codes.

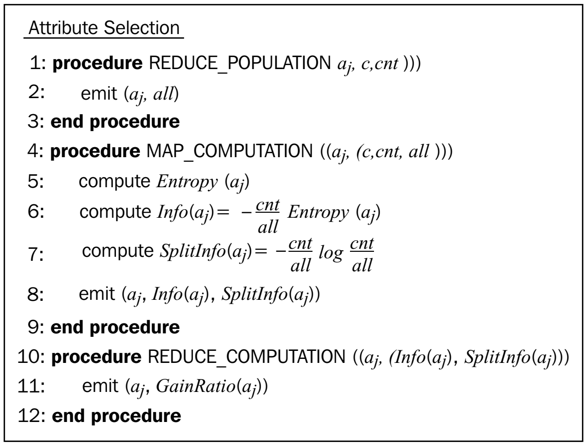
The MapReduce programming model is illustrated in the following diagram:





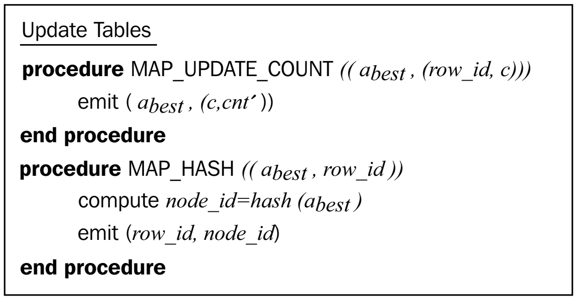
Data Preparation

```
procedure MAP_Attribute $(row\_id, (a_1, a_2, \ldots, a_M, c))$
    emit $(a_j, (row\_id, c))$
end procedure
procedure REDUCE_ATTRIBUTE $(a_j, (row\_id, c))$
    emit $(a_j, c, cnt))$
end procedure
```



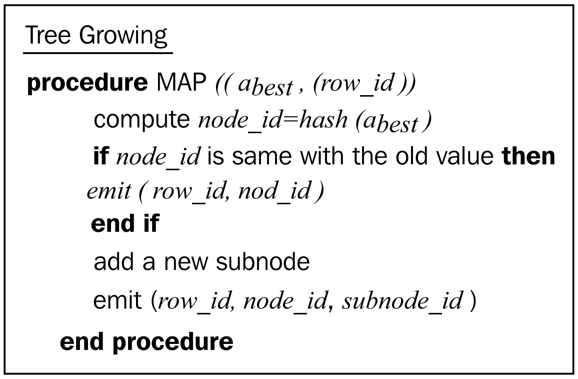
Attribute Selection

```
procedure REDUCE_POPULATION $a_j, c, cnt)))$
    emit $(a_j, all)$
end procedure
procedure MAP_COMPUTATION $((a_j, (c, cnt, all)))$
    compute $Entropy\ (a_j)$
    compute $Info(a_j) = -\frac{cnt}{all}\ Entropy\ (a_j)$
    compute $SplitInfo(a_j) = -\frac{cnt}{all}\ log\ \frac{cnt}{all}$
    emit $(a_j, Info(a_j), SplitInfo(a_j))$
end procedure
procedure REDUCE_COMPUTATION $((a_j, (Info(a_j), SplitInfo(a_j)))$
    emit $(a_j, GainRatio(a_j))$
end procedure
```



Update Tables

```
procedure MAP_UPDATE_COUNT (( $a_{best}$ , (row_id, c)))
        emit ( $a_{best}$ , (c,cnt′ ))
end procedure
procedure MAP_HASH (( $a_{best}$ , row_id ))
        compute node_id=hash ($a_{best}$ )
        emit (row_id, node_id)
end procedure
```



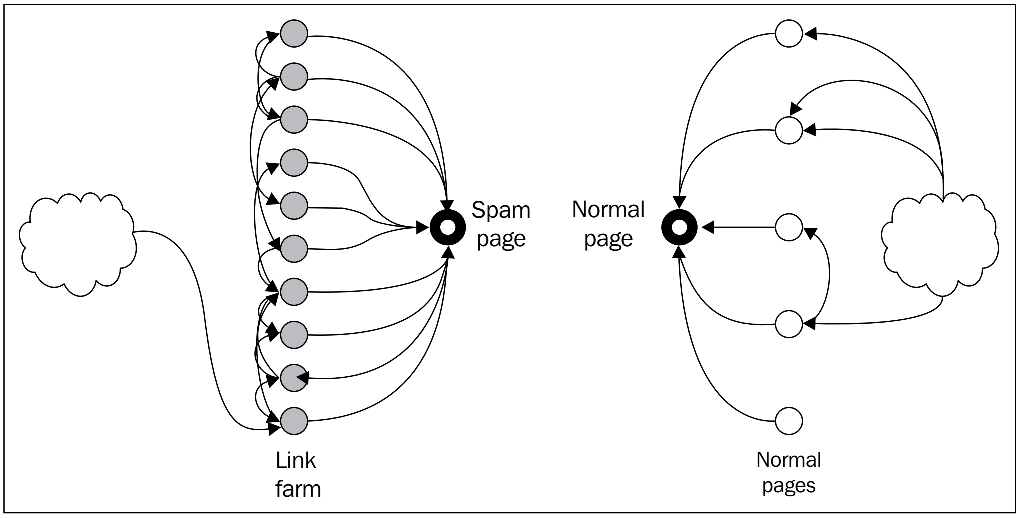
Tree Growing

```
procedure MAP (( $a_{best}$ , (row_id ))
         compute node_id=hash ($a_{best}$ )
         if node_id is same with the old value then
        emit ( row_id, nod_id )
         end if
         add a new subnode
         emit (row_id, node_id, subnode_id )
    end procedure
```

Web spam detection

Spamming occurs along with the emergence of search engine technologies to pursue higher rank with the deceiving search engines relevancy algorithm, but not improve their own website technologies. It performs deliberate actions to trigger unjustifiable favorable relevance or importance for a specific web page, in contrast to the page's true value or merit. The spam page finally receives a substantial amount of score from other spam pages to boost its rank in search engine results by deliberately manipulating the search engine indexes. Finally, traffic is driven to the spammed pages. As a direct result of the Web spam, the information quality of the Web world is degraded, user experience is manipulated, and the security risk for use increases due to exploitation of user privacy.

One classic example, denoted as link-farm, is illustrated in the following diagram, in which a densely connected set of pages is created with the target of cheating a link-based ranking algorithm, which is also called **collusion**:



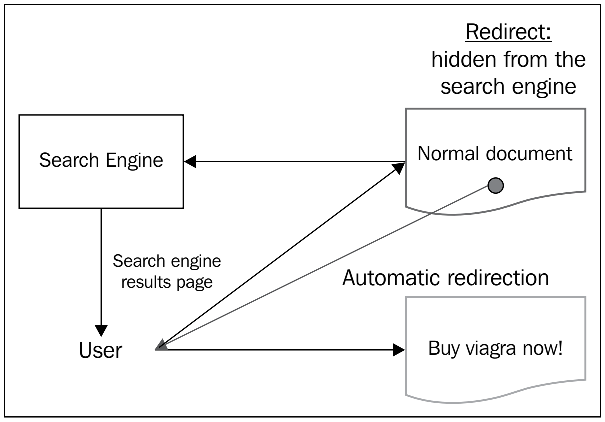
There are three major categories of spam from a business point of view:

- Page Spoofing
- Browser-Based Attacks
- Search Engine Manipulations

There are three major categories of spam from a technology point of view:

- **Link spam**: This consists of the creation of the link structure, often a tight-knit community of links, targeted at affecting the outcome of a link-based ranking algorithm. Possible technologies include honey pot, anchor-text spam, blog/wiki spam, link exchange, link farm, and expired domains.
- **Content spam**: This crafts the contents of web page pages. One example is inserting unrelated keywords to the web page content for higher ranking on search engines. Possible technologies include hidden text (size and color), repetition, keyword stuffing/dilution, and language-model-based technologies (phrase stealing and dumping).

- **Cloaking**: This sends content to search engines which looks different from the version viewed by human visitors.



The link-based spam detections usually rely on automatic classifiers, detecting the anomalous behavior of a link-based ranking algorithm, and so on. Classifier or language model disagreement can be adopted to detect content spam. While the cloaking detection solution is inherent, one of them is comparing the indexed page with the pages visitors saw.

How to apply a decision tree for web spam detection? The links and contents of the web spam, after statistical analysis, are unique compared to other normal pages. Some properties are valuable for detecting spam, trustworthiness of the page, neutrality (facts), and bias. Web spam detection can be a good example for illustrating the C4.5 algorithm.

Some domain-related knowledge can be applied to the classification solution. One observed phenomenon is that bad links always have links between them. The links between web pages and websites are often not randomly set but with certain rules and they can be detected by classifiers.

About the attributes for a certain data point, the dataset can be divided into two groups, link-based features and content-based features:

- **Link-based features**: These include degree-based measures such as in-degree and out-degree of the web hosts. The second is PageRank-based features, which is an algorithm to compute a score for every page. The third is TrustRank-based features, which measure the trustworthiness of certain web pages to some benchmarked, trustworthy web pages.

- **Content-based features**: These include the number of words in the page, number of words in the title, average word length, amount of anchor text, fraction of anchor text, fraction of visible text, fraction of page drawn from globally popular words, fraction of globally popular words, compressibility, corpus precision and corpus recall, query precision and query recall, independent trigram or n-gram likelihood, and entropy of trigrams or n-gram.

All these features are included in one data point to set up a preprocessed dataset, which in turn can be used by classification algorithms, especially decision tree algorithms such as C4.5 to work as web spam classifiers to distinguish spam from normal pages. Among all the classification solutions, C4.5 achieves the best performance.

Web key resource page judgment using CART

Classification and Regression Trees (**CART**) is one of the most popular decision tree algorithms. It is a binary recursive partitioning algorithm that can be used to process continuous and nominal attributes.

There are three main steps in the CART algorithm. The first is to construct the maximum tree (binary tree). The second step is to choose the right size of the tree. The last step is to classify new data using the result tree.

Compared to other algorithms, there are many important characteristics of CART:

- Binary decision tree (a binary recursive partitioning process)
- The source dataset can have continuous or nominal attributes
- No stopping rule (unless no possible splits are available)
- Tree pruning with cost-complexity pruning
- Nonparametric
- No variables to be selected in advance
- The missing value is dealt with an adaptive and better strategy
- The outlier can be easily handled
- No assumptions
- Computationally fast
- At each split point, only one variable is used

- Only one optimal tree is taken as the result tree, which is formed from a sequence of nested pruned candidate trees generated by CART
- Automatically handling the missing value in the source dataset

The weighted Gini index equation is defined as follows:

$$G\left(D_Y, D_N\right) = \frac{n_Y}{n} G\left(D_Y\right) + \frac{n_N}{n} G\left(D_N\right)$$

The CART measure is different; the goodness of the split point is proportional to the value of measure. The higher the better.

$$CART\left(D_Y, D_N\right) = 2 \frac{n_Y}{n} \frac{n_N}{n} \sum_{i=1}^{k} | P\left(c_i | D_Y\right) - P(c_i \mid D_N) |$$

The CART algorithm

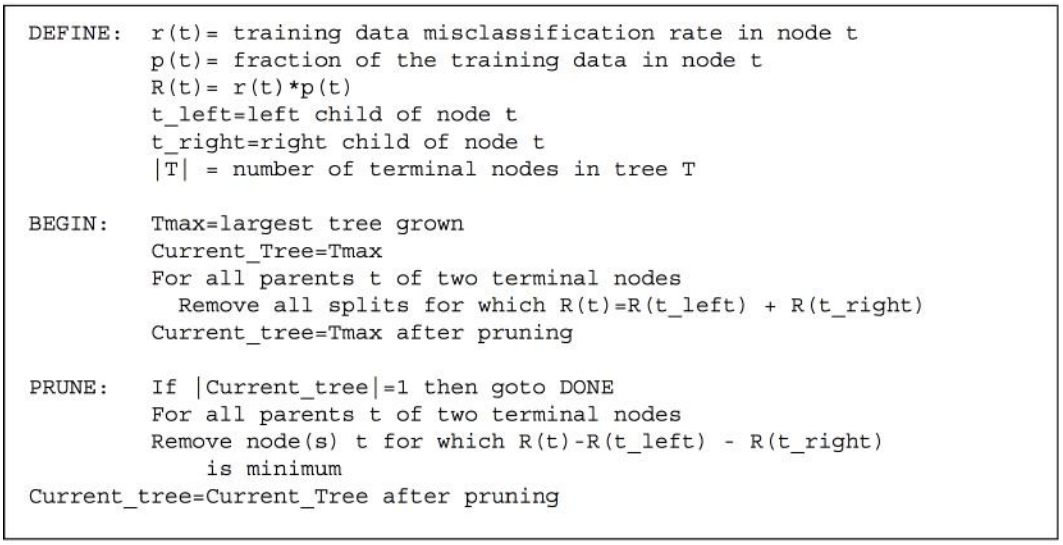
Splitting rules, with the omission of the following parts, is too lengthy to include in this section:

- Separate handling of continuous and categorical splitters
- Special handling for categorical splitters with multiple levels
- Missing value handling
- Tree pruning
- Tree selection

Here is the pseudocode for the CART algorithm, the simplified tree-growing algorithm:

```
BEGIN:  Assign all training data to the root node
        Define the root node as a terminal node

SPLIT:
New_splits=0
FOR every terminal node in the tree:
   If the terminal node sample size is too small or all instances in the
   node belong to the same target class goto GETNEXT
   Find the attribute that best separates the node into two child nodes
   using an allowable splitting rule
   New_splits+1
GETNEXT:
NEXT
```

The simplified pruning algorithm is as follows:



```
DEFINE:  r(t)= training data misclassification rate in node t
         p(t)= fraction of the training data in node t
         R(t)= r(t)*p(t)
         t_left=left child of node t
         t_right=right child of node t
         |T| = number of terminal nodes in tree T

BEGIN:   Tmax=largest tree grown
         Current_Tree=Tmax
         For all parents t of two terminal nodes
           Remove all splits for which R(t)=R(t_left) + R(t_right)
         Current_tree=Tmax after pruning

PRUNE:   If |Current_tree|=1 then goto DONE
         For all parents t of two terminal nodes
         Remove node(s) t for which R(t)-R(t_left) - R(t_right)
             is minimum
Current_tree=Current_Tree after pruning
```

The R implementation

Please look up the R codes file `ch_02_cart.R` from the bundle of R codes for the previously mentioned algorithms. One example is chosen to apply the CART algorithm to, in the following section.

Web key resource page judgment

Web key resource judgment arises from the domain of web information retrieval and web search engines. The original concept is from the authority value, the hub value, and the HITS algorithm. During queries of information from IR systems or search engines, finding important and related information from an overwhelmingly increasing volume of information is a challenging task. A better judgment leads to less indexing storage and a more informative querying result.

A key resource page is a high quality web page with much more information per selected topic compared to an ordinary web page on the same topic. In order to measure the importance of a certain web page, feature selection is the first thing required in the design.

The link-based features used in current search technologies can't resolve such issues at an acceptable accuracy rate. To improve the accuracy rate, more global information across many data instances can be adopted in addition to the single-data-instance-related attributes or features, which means local attributes.

Experimental results show that the key web page should contain in-site out-links with anchor text to other pages. Non-content attributes, such as web page links related attributes and content structures of pages, can be applied to judge key resource pages. The possible attributes are listed as follows:

- **In-degree or in-links**: This denotes the number of links pointing to the page. Observation shows that the higher the number of in-links related to the key page, the more the links from other sites to that page, which means more recommendations to a certain extent.
- **URL length or the depth of a page's URL**: There are four types of URLs defined in the following box: root, subroot, path, and filename. The four types of URLs map to four levels of length, that is, 1 to 4 respectively. A lower prior probability with a lower level and a higher prior probability mean a bigger possibility to be a key resource page.
- **In-site out-link anchor text rate**: This refers to the rate of the length of the anchor text to the document or page content length.
- **In-site out-link number**: This refers to the number of links embedded in a page.
- **Document length (in words)**: This filters out specified predefined non-usable characters from the document. This attribute can predict the relevance of a page because of the non-uniform distribution.

With the attributes just mentioned, the uniform sampling problem can be bypassed to a certain extent. The dataset can be easily built and used by decision tree induction algorithms such as CART.

Trojan traffic identification method and Bayes classification

Among probabilistic classification algorithms is the Bayes classification, which is based on Bayes' theorem. It predicts the instance or the class as the one that makes the posterior probability maximal. The risk for Bayes classification is that it needs enough data to estimate the joint probability density more reliably.

Given a dataset D with a size `n`, each instance or point `x` belonging to `D` with a dimension of *m*, for each $x_i = \left(x_{i1}, x_{i2}, \ldots, x_{im},\right) \in D, y_i \; is\, the\, class\; for\, x_i, y_i \in \left\{c_1, c_2, \ldots, c_k\right\}$. To predict the class y' of any `x`, we use the following formula:

$$\hat{y} = arg\, max\left\{P\left(c_i | \, x\right) | \forall i\right\}$$

Basing on Bayes' theorem, $P\ (x \,|\, c_i)$ is the likelihood:

$$P\left(c_i | x\right) = \frac{P(x \,|\, c_i)P\left(c_i\right)}{P\left(x\right)} \quad , P\left(x\right) = \sum_{j=1}^{k} P\left(x | c_j\right) P\left(c_j\right)$$

Then we get the following new equations for predicting y' for `x`:

$$\hat{y} = arg\, max\left\{P\left(c_i | x\right) | \forall i\right\} = arg\, max\left\{\frac{P\left(x | c_i\right) P\left(c_i\right)}{P\left(x\right)}\right\} = arg\, \max\{P(x \,|\, c_i) P\left(c_i\right)\}$$

Estimating

With new definitions to predict a class, the prior probability and its likelihood needs to be estimated.

Prior probability estimation

Given the dataset D, if the number of instances in D labeled with class c_i is n_i and the size of `D` is `n`, we get the estimation for the prior probability of the class c_i as follows:

$$\hat{P}\left(c_i\right) = \frac{n_i}{n}$$

Likelihood estimation

For numeric attributes, assuming all attributes are numeric, here is the estimation equation. One presumption is declared: each class c_i is normally distributed around some mean μ_i with the corresponding covariance matrix $\sum_i$. $\hat{\mu}_i$ is used to estimate μ_i, $\hat{\sum}_i$ for $\sum_i$:

$$\hat{y} = arg\max\{f_i(x)P(c_i) \mid \forall i\}$$

$$f_i(x) = f(x|\mu_i, \Sigma_i) = \frac{1}{\left(\sqrt{2\pi}\right)^d \sqrt{|\Sigma_i|}} \exp\left\{-\frac{(x-\mu_i)^T \sum_i^{-1} (x-\mu_i)}{2}\right\}$$

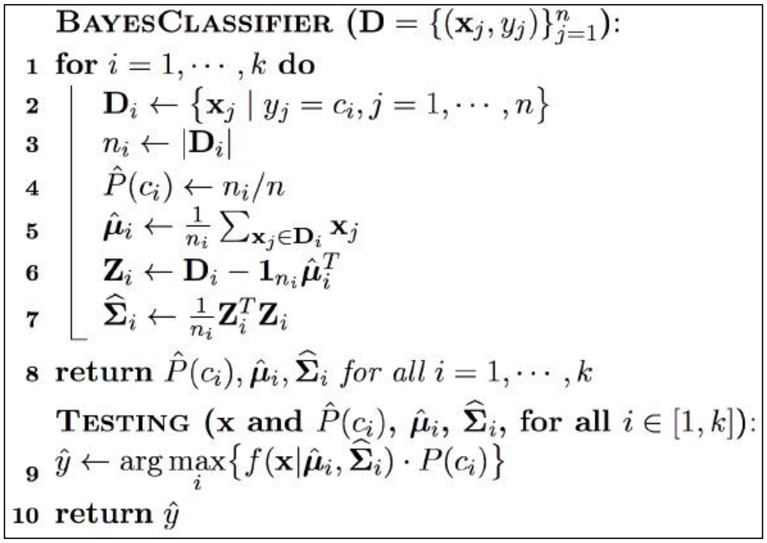
$$\hat{\mu}_i = \frac{1}{n_i} \sum_{x_j \in D_i} x_j$$

$$\hat{\Sigma}_i = \frac{1}{n_i} Z_i^T Z_i$$

For categorical attributes, it can also be dealt with similarly but with minor difference.

The Bayes classification

The pseudocode of the Bayes classification algorithm is as follows:



```
BayesClassifier ($\mathbf{D} = \{(\mathbf{x}_j, y_j)\}_{j=1}^n$):
for $i = 1, \cdots, k$ do
    $\mathbf{D}_i \leftarrow \{\mathbf{x}_j \mid y_j = c_i, j = 1, \cdots, n\}$
    $n_i \leftarrow |\mathbf{D}_i|$
    $\hat{P}(c_i) \leftarrow n_i/n$
    $\hat{\boldsymbol{\mu}}_i \leftarrow \frac{1}{n_i} \sum_{\mathbf{x}_j \in \mathbf{D}_i} \mathbf{x}_j$
    $\mathbf{Z}_i \leftarrow \mathbf{D}_i - \mathbf{1}_{n_i} \hat{\boldsymbol{\mu}}_i^T$
    $\widehat{\boldsymbol{\Sigma}}_i \leftarrow \frac{1}{n_i} \mathbf{Z}_i^T \mathbf{Z}_i$
return $\hat{P}(c_i), \hat{\boldsymbol{\mu}}_i, \widehat{\boldsymbol{\Sigma}}_i$ for all $i = 1, \cdots, k$
Testing ($\mathbf{x}$ and $\hat{P}(c_i)$, $\hat{\boldsymbol{\mu}}_i$, $\widehat{\boldsymbol{\Sigma}}_i$, for all $i \in [1, k]$):
$\hat{y} \leftarrow \arg\max_i \{f(\mathbf{x}|\hat{\boldsymbol{\mu}}_i, \widehat{\boldsymbol{\Sigma}}_i) \cdot P(c_i)\}$
return $\hat{y}$
```

The R implementation

The R code for the Bayes classification is listed as follows:

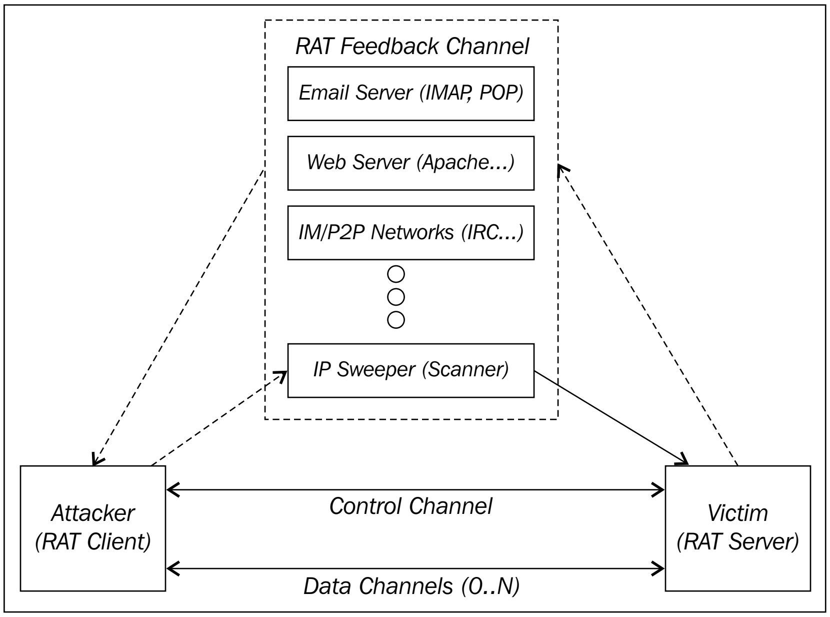
```
BayesClassifier <- function(data,classes){
    bayes.model <- NULL

    data.subsets <- SplitData(data,classes)
    cards <- GetCardinality(data.subsets)
    prior.p <- GetPriorProbability(cards)
    means <- GetMeans(data.subsets,cards)
   cov.m <-GetCovarianceMatrix(data.subsets,cards,means)

    AddCardinality(bayes.model,cards)
    AddPriorProbability(bayes.model,prior.p)
    AddMeans(bayes.model,means)
    AddCovarianceMatrix(bayes.model,cov.m)

    return(bayes.model)
}

TestClassifier <- function(x){
    data <- GetTrainingData()
    classes <- GetClasses()
    bayes.model <- BayesClassifier(data,classes)

    y <- GetLabelForMaxPostProbability(bayes.model,x)

    return(y)
}
```

One example is chosen to apply the Bayes classification algorithm, in the following section.

Trojan traffic identification method

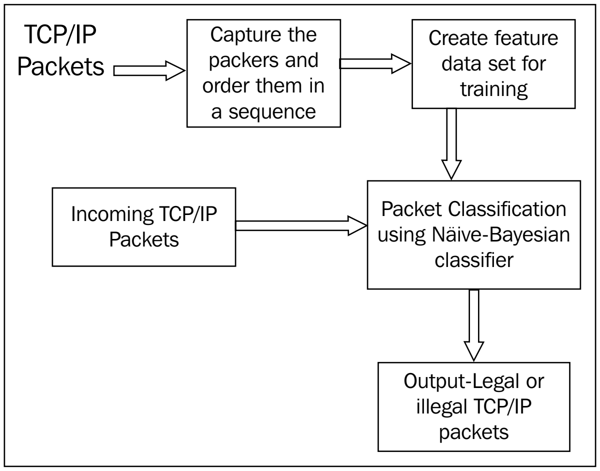
A Trojan horse, which is a malicious program, surreptitiously performs its operation under the guise of a legitimate program. It has a specific pattern and unique malicious behavior (such as traffic and other operations). For example, it may obtain account information and sensitive system information for further attacks. It can also fork processes for dynamic ports, impersonate software and redirect traffic of affected services to other systems, make them available to attackers to hijack connections, intercept valuable data, and inject fake information or phishing.

Depending on the purpose of Trojans, there are many versatile types of designs for Trojans, each with a certain traffic behavior. With the ability to identify the Trojan traffic, further processing can be performed to protect information. As a result, detecting the traffic of Trojans is one of the main tasks to detect Trojans on system. The behavior of Trojans is an outlier compared to the normal software. So the classification algorithms such as the Bayesian classification algorithm can be applied to detect the outliers. Here is a diagram showing the Trojan traffic behavior:



The malicious traffic behaviors include but are not limited to spoofing the source IP addresses and (short and long) term scanning the flow of the address/port that serves as the survey for successive attacks. Known Trojan traffic behaviors are used as the positive training data instances. The normal traffic behaviors are used as the negative data instances in the training dataset. These kinds of datasets are continuously collected by NGOs.

The attributes used for a dataset may include the latest DNS request, the NetBIOS name table on the host machine, ARP cache, intranet router table, socket connections, process image, system ports behavior, opened files updates, remote files updates, shell history, packet TCP/IP headers information, identification fields (IPID) of the IP header, **Time To Live** (**TTL**), and so forth. One possible attribute set for a dataset is source IP, port, target IP, target port, number of flows, number of packets, number of bytes, timestamp at certain checkpoint, and the class label for the type of detection. The DNS traffic plays an important role in the Trojans' detection too; the traffics of Trojans has certain a relation with DNS traffic.


TCP/IP Packets
Capture the packers and order them in a sequence
Create feature data set for training
Incoming TCP/IP Packets
Packet Classification using Näive-Bayesian classifier
Output-Legal or illegal TCP/IP packets

The traditional technologies for detecting a Trojan often rely on the Trojan's signature and can be deceived by dynamic ports, encrypted messages, and so on. This led to the introduction of mining technologies for the classification of Trojan traffic. The Bayesian classifier is one of the better solutions among others. The preceding diagram is one such possible structure.

Identify spam e-mail and Naïve Bayes classification

The Naïve Bayes classification presumes that all attributes are independent; it simplifies the Bayes classification and doesn't need the related probability computation. The likelihood can be defined with the following equation:

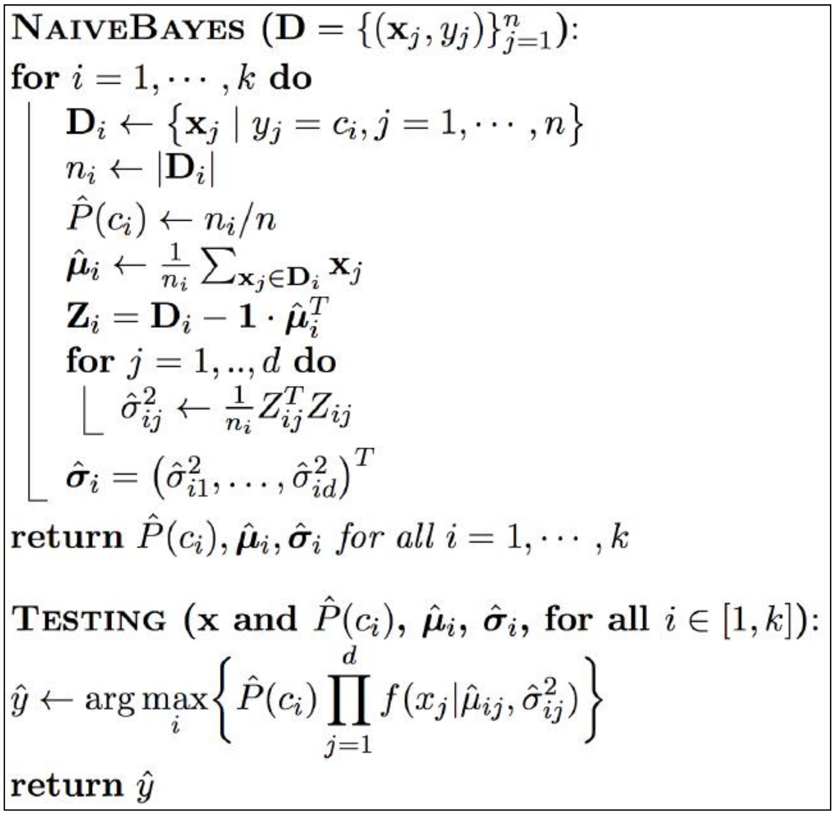
$$P\left(x|c_i\right) = P\left(x_1, x_2, .., x_d|c_i\right) = \prod_{j=1}^{d} P(x_j \mid c_i)$$

Some of the characteristics of the Naïve Bayes classification are as follows:

- Robust to isolated noise
- Robust to irrelevant attributes
- Its performance might degrade due to correlated attributes in the input dataset

The Naïve Bayes classification

The pseudocode of the Naïve Bayes classification algorithm, with minor differences from the Bayes classification algorithm, is as follows:



```
NAIVEBAYES ($\mathbf{D} = \{(\mathbf{x}_j, y_j)\}_{j=1}^{n}$):
for $i = 1, \cdots, k$ do
    $\mathbf{D}_i \leftarrow \{\mathbf{x}_j \mid y_j = c_i, j = 1, \cdots, n\}$
    $n_i \leftarrow |\mathbf{D}_i|$
    $\hat{P}(c_i) \leftarrow n_i / n$
    $\hat{\boldsymbol{\mu}}_i \leftarrow \frac{1}{n_i} \sum_{\mathbf{x}_j \in \mathbf{D}_i} \mathbf{x}_j$
    $\mathbf{Z}_i = \mathbf{D}_i - \mathbf{1} \cdot \hat{\boldsymbol{\mu}}_i^T$
    for $j = 1, .., d$ do
        $\hat{\sigma}_{ij}^2 \leftarrow \frac{1}{n_i} Z_{ij}^T Z_{ij}$
    $\hat{\boldsymbol{\sigma}}_i = \left(\hat{\sigma}_{i1}^2, \ldots, \hat{\sigma}_{id}^2\right)^T$
return $\hat{P}(c_i), \hat{\boldsymbol{\mu}}_i, \hat{\boldsymbol{\sigma}}_i$ for all $i = 1, \cdots, k$

TESTING ($\mathbf{x}$ and $\hat{P}(c_i)$, $\hat{\boldsymbol{\mu}}_i$, $\hat{\boldsymbol{\sigma}}_i$, for all $i \in [1, k]$):
$\hat{y} \leftarrow \arg\max_i \left\{ \hat{P}(c_i) \prod_{j=1}^{d} f(x_j | \hat{\mu}_{ij}, \hat{\sigma}_{ij}^2) \right\}$
return $\hat{y}$
```

The R implementation

The R code for the Naïve Bayes classification is listed as follows:

```
NaiveBayesClassifier <- function(data,classes){
    naive.bayes.model <- NULL

    data.subsets <- SplitData(data,classes)
    cards <- GetCardinality(data.subsets)
    prior.p <- GetPriorProbability(cards)
    means <- GetMeans(data.subsets,cards)
    variances.m <- GetVariancesMatrix
      (data.subsets,cards,means)

    AddCardinality(naive.bayes.model,cards)
    AddPriorProbability(naive.bayes.model,prior.p)
    AddMeans(naive.bayes.model,means)
    AddVariancesMatrix(naive.bayes.model,variances.m)

    return(naive.bayes.model)
}

TestClassifier <- function(x){
    data <- GetTrainingData()
    classes <- GetClasses()
    naive.bayes.model <- NaiveBayesClassifier(data,classes)

    y <- GetLabelForMaxPostProbability(bayes.model,x)

    return(y)
}
```

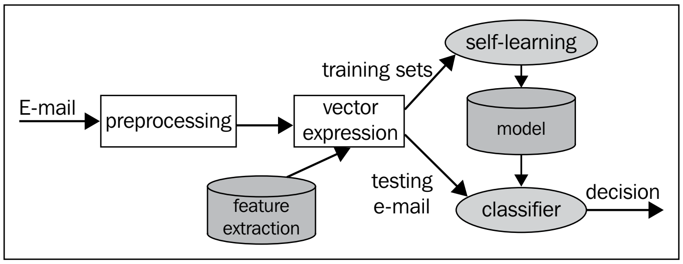
One example is chosen to apply the Naïve Bayes classification algorithm, in the following section.

Identify spam e-mail

E-mail spam is one of the major issues on the Internet. It refers to irrelevant, inappropriate, and unsolicited emails to irrelevant receivers, pursuing advertisement and promotion, spreading malware, and so on.

> *Unsolicited, unwanted e-mail that was sent indiscriminately, directly or indirectly, by a sender having no current relationship with the recipient* is the formal definition of e-mail spam from spam track at the **Text Retrieval Conference** (**TREC**).

The increase in e-mail users, business e-mail campaigns, and suspicious usage of e-mail have resulted in a massive dataset of spam e-mails, which in turn necessitate high-efficiency solutions to detect e-mail spam:



E-mail spam filters are automated tools that recognize spam and prevent further delivery. The classifier serves as a spam detector here. One solution is to combine inputs of a couple of e-mail spam classifiers to present improved classification effectiveness and robustness.

Spam e-mail can be judged from its content, title, and so on. As a result, the attributes of the e-mails, such as subject, content, sender address, IP address, time-related attributes, in-count /out-count, and communication interaction average, can be selected into the attributes set of the data instance in the dataset. Example attributes include the occurrence of HTML form tags, IP-based URLs, age of link-to domains, nonmatching URLs, HTML e-mail, number of links in the e-mail body, and so on. The candidate attributes include discrete and continuous types.

The training dataset for the Naïve Bayes classifier will be composed of the labeled spam e-mails and legitimate e-mails.

Rule-based classification of player types in computer games and rule-based classification

Compared to other classification algorithms, the learned model for a rule-based classification is set up by an IF-THEN rules set. The rules set can be transformed from the decision tree or by the following algorithm. An IF-THEN rule has the following format:

```
IF condition_holds_true THEN make_a_conclusion
```

An alternative format is as follows:

$$RULE\ Antecedent \leftarrow RULE\ Consequent$$

For a given instance or record in the source dataset, if the `RULE` antecedent holds true, the rule is defined to cover the instance, and it is satisfied.

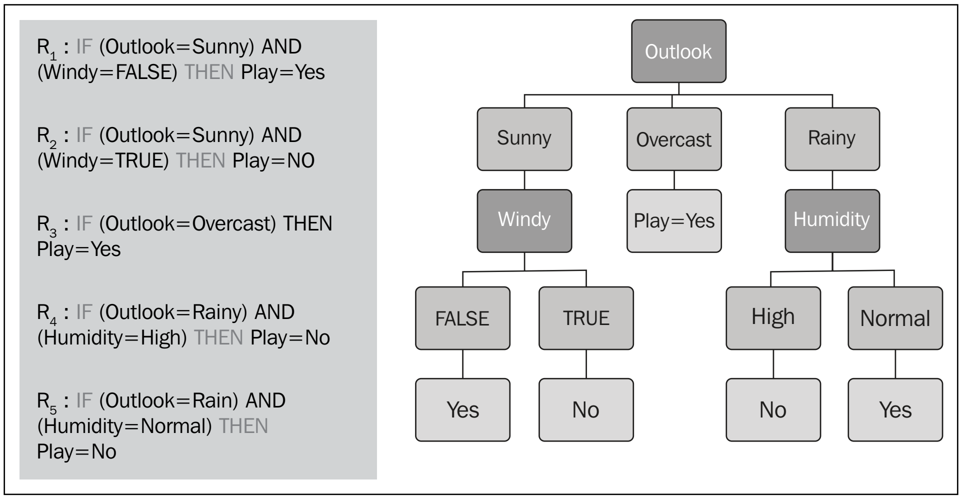
Given a rule `R`, the coverage and accuracy are defined as follows:

$$Coverage(R) = \frac{|D_{cover}|}{|D|}$$

$$Accuracy(R) = \frac{|D_{correct}|}{|D_{cover}|}$$

Transformation from decision tree to decision rules

It is very convenient to transform the decision tree into a decision rules set for further processing. Along with every path from the root to a leaf in the decision tree, a rule can be written. The left-hand side, the rule antecedent, of any rule is constructed by the combination of the label of the nodes and the labels of the arcs, then the rule consequent by the leaf node. One example of extracting classification rules from the decision tree is illustrated in the following diagram:



R_1 : IF (Outlook=Sunny) AND (Windy=FALSE) THEN Play=Yes

R_2 : IF (Outlook=Sunny) AND (Windy=TRUE) THEN Play=NO

R_3 : IF (Outlook=Overcast) THEN Play=Yes

R_4 : IF (Outlook=Rainy) AND (Humidity=High) THEN Play=No

R_5 : IF (Outlook=Rain) AND (Humidity=Normal) THEN Play=No

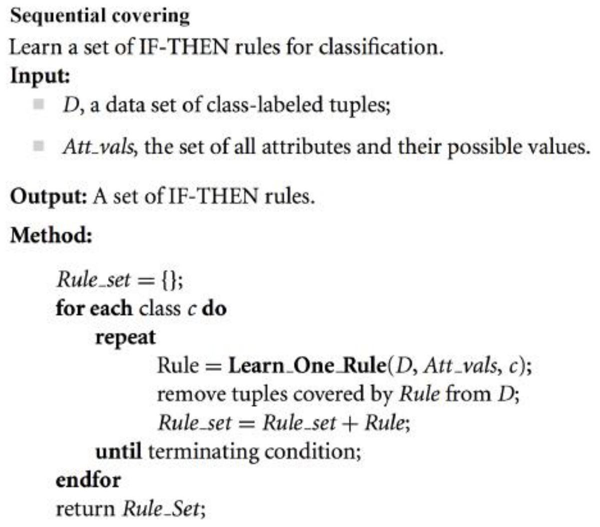
One important question is the pruning of the resulting rules set.

Rule-based classification

Rules are learned sequentially, one at a time. Here is the pseudocode of the algorithm to build a rule-based classifier. The `LearnOneRule` function is designed with the greedy strategy. Its target is to cover the positive instance in the source dataset as much as possible, and none or as few as possible of the negative instance at the same time. All the instances in the source dataset with a specific class are defined as positive, and those that belong to other classes are considered to be negative. An initial rule `r` is generated, which keeps refining until the stop condition is met.

Sequential covering algorithm

The pseudocode of the generic sequential covering algorithm is as follows. The input parameters include the dataset with class-labeled tuples and the attributes set with all of their possible values. The output is a set of IF-THEN rules as follows:



Sequential covering

Learn a set of IF-THEN rules for classification.

Input:

- *D*, a data set of class-labeled tuples;
- *Att_vals*, the set of all attributes and their possible values.

Output: A set of IF-THEN rules.

Method:

```
Rule_set = {};
for each class c do
    repeat
        Rule = Learn_One_Rule(D, Att_vals, c);
        remove tuples covered by Rule from D;
        Rule_set = Rule_set + Rule;
    until terminating condition;
endfor
return Rule_Set;
```

The RIPPER algorithm

Repeated Incremental Pruning to Produce Error Reduction (**RIPPER**) is a direct rule-based classifier, in which the rule set is relatively convenient to interpret and the most practical for imbalance problems.

As per the growth of a rule, the algorithm starts from an empty rule and adds conjuncts, which maximize or improve the information gain measure, that is, the FOIL. It stops at the situation so that the rule does not cover negative rules. The resulting rule is pruned immediately with incremental reduced error pruning. Any final sequence of conditions is removed once it maximizes the measure of pruning v, which is calculated as follows:

$$v = \frac{p-n}{p+n}$$

The sequential covering algorithm is used to build a rule set; the new **description length** (**DL**) is computed once a new rule is added to the rule set. The rule set is then optimized.

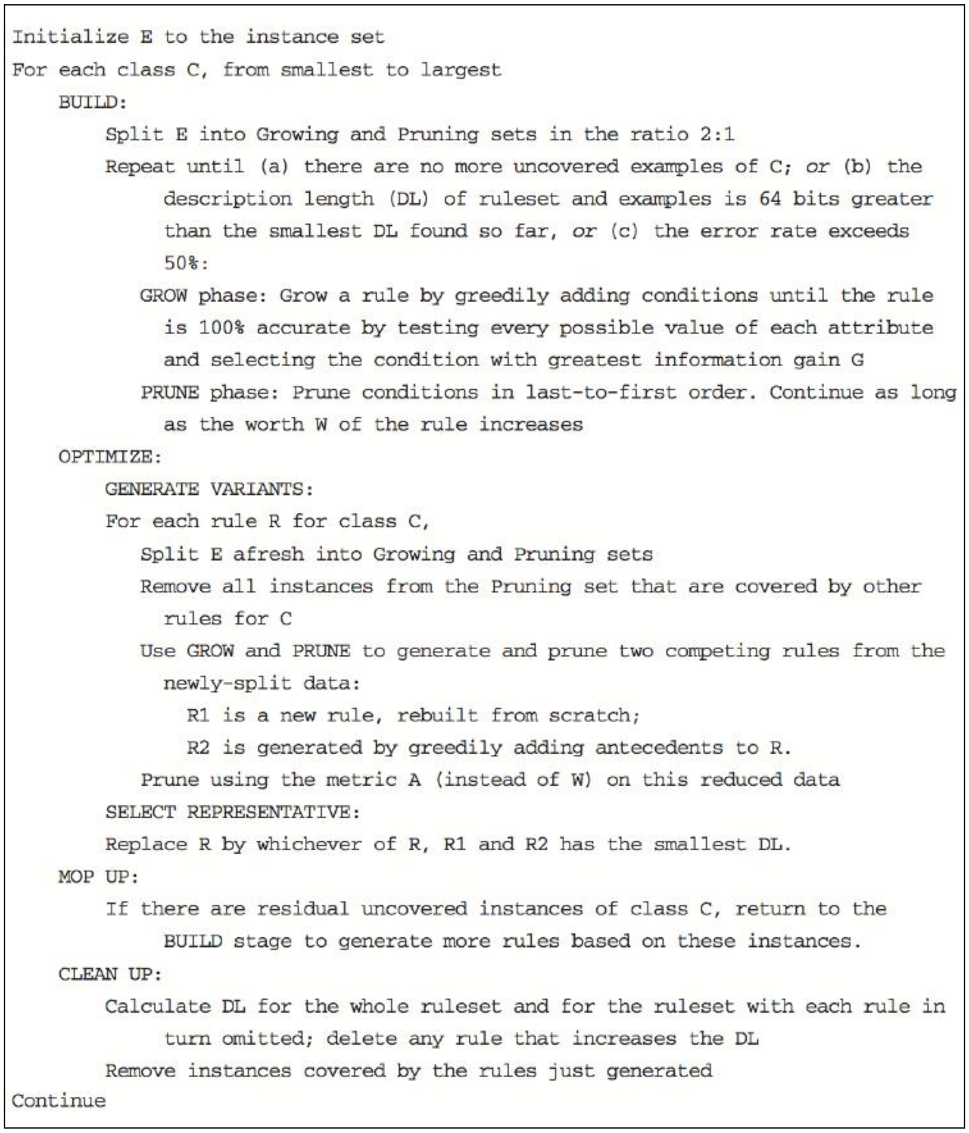
Given are *p* as the number of positive examples covered by this rule and *n* as the number of negative rules covered by this rule. *P* denotes the number of positive examples of this class, and *N* the number of the negative examples of this class.

$$G = p\left[\log\left(\frac{p}{t}\right) - \log\left(\frac{P}{T}\right)\right]$$

$$W = \frac{p+1}{t+2}$$

$$A = \frac{p+n'}{T}, current\, rule' s accuracy$$

The pseudocode of the RIPPER algorithm is as follows:



```
Initialize E to the instance set
For each class C, from smallest to largest
    BUILD:
        Split E into Growing and Pruning sets in the ratio 2:1
        Repeat until (a) there are no more uncovered examples of C; or (b) the
             description length (DL) of ruleset and examples is 64 bits greater
             than the smallest DL found so far, or (c) the error rate exceeds
             50%:
           GROW phase: Grow a rule by greedily adding conditions until the rule
             is 100% accurate by testing every possible value of each attribute
             and selecting the condition with greatest information gain G
           PRUNE phase: Prune conditions in last-to-first order. Continue as long
             as the worth W of the rule increases
    OPTIMIZE:
        GENERATE VARIANTS:
        For each rule R for class C,
           Split E afresh into Growing and Pruning sets
           Remove all instances from the Pruning set that are covered by other
             rules for C
           Use GROW and PRUNE to generate and prune two competing rules from the
             newly-split data:
               R1 is a new rule, rebuilt from scratch;
               R2 is generated by greedily adding antecedents to R.
           Prune using the metric A (instead of W) on this reduced data
        SELECT REPRESENTATIVE:
        Replace R by whichever of R, R1 and R2 has the smallest DL.
    MOP UP:
        If there are residual uncovered instances of class C, return to the
             BUILD stage to generate more rules based on these instances.
    CLEAN UP:
        Calculate DL for the whole ruleset and for the ruleset with each rule in
             turn omitted; delete any rule that increases the DL
        Remove instances covered by the rules just generated
Continue
```

The R implementation

The R code for the rule-based classification is listed as follows:

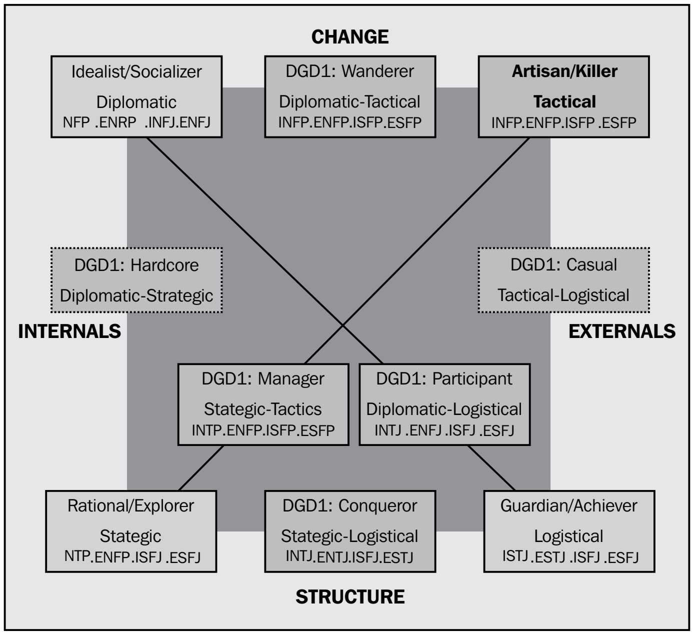
```
SequentialCovering <- function(data,x,classes){
    rule.set <- NULL

    classes.size <- GetCount(classes)
     idx <- 0
    while( idx <= classes.size ){
        idx <- idx+1
        one.class <- GetAt(classes,idx)
        repeat{
            one.rule <- LearnOneRule(newdata,x,one.class)
            data <- FilterData(data,one.rule)
            AddRule(rule.set,one.rule)
            if(CheckTermination(data,x,classes,rule.set)){
                break;
            }
        }
    }
    return(rule.set)
}
```

One example is chosen to apply the rule-based classification algorithm, in the following section.

Rule-based classification of player types in computer games

During computer game progressing and in the game context, improving the experience of a game is always a continual task. Classification of player types is one major task, which in turn brings more improvements including game design.

One of the popular player models of typological of temperature is the DGD player topology, which is illustrated in the following diagram. Given this model, the game players can be labeled with appropriate types, the game can be explained, it helps in designing new games, and so forth.



Based on the player behaviors or models, we can train the decision tree model with the dataset, and the rules set from the trained decision tree model. The dataset will come from the game log and some predefined domain knowledge.

Time for action

Here are some practices for you to check what you've learned so far:

- Running the R code of the ID3 algorithm step by step upon a minor dataset to trace the values of the important factors at each step
- Preparing the dataset related to web logs and creating an application that detects web attacks using ID3
- Implementing an R code to generate decision rules from a decision tree
- What is Gain Ratio?

Summary

In this chapter, we learned the following facts:

- Classification is a class of dispatch instances to one of predefined categories
- Decision tree induction is to learn the decision tree from the source dataset with the (instance and class-label) pairs under the supervised learning mode
- ID3 is a decision tree induction algorithm
- C4.5 is an extension of ID3
- CART is a decision tree induction
- Bayes classification is a statistical classification algorithm
- Naïve Bayes classification is a simplified version of Bayes classification in which there is a presumption of independence
- Rule-based classification is a classification model applying the rule set, which can be collections by direct algorithm, the sequential covering algorithm, and the indirect method by decision tree transforming

In the next chapter, you'll cover the more-advanced classification algorithms, including Bayesian Belief Network, SVM, k-Nearest Neighbors algorithm, and so on.

4
Advanced Classification

In this chapter, you will learn about the top classification algorithms written in the R language. You will also learn the ways to improve the classifier.

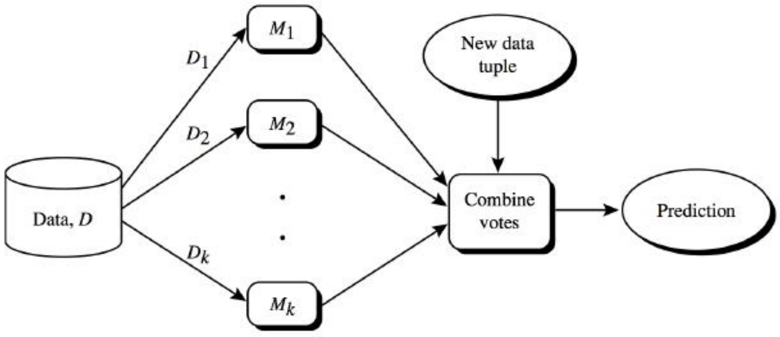
We will cover the following topics:

- Ensemble methods
- Biological traits and Bayesian belief network
- Protein classification and the k-Nearest Neighbors algorithm
- Document retrieval and Support Vector Machine
- Text classification using sentential frequent itemsets and classification using frequent patterns
- Classification using the backpropagation algorithm

Ensemble (EM) methods

To improve the accuracy of classification, EM methods are developed. The accuracy is dramatically improved by at least one grade compared to its base classifiers, because the EM methods make mistakes only when at least half of the result of the base classifiers are wrong.

The concept structure of EM methods is illustrated in the following diagram:



The label for the new data tuple is the result of the voting of a group of base classifiers. A combined classifier is created based on several base classifiers. Each classifier is trained with a different dataset or training set re-sampled with the replacement of the original training dataset.

Three popular EM methods are discussed in the successive sections:

- Bagging
- Boosting
- Random forests

The bagging algorithm

Here is a concise description of the bagging algorithm (noted as the bootstrap aggregation), followed by the summarized pseudocode. For iteration *i* ($i \in [1,k]$), a training set, D_i , of *d* tuples is sampled with replacement from the original set of tuples, *D*. Any training set is sampled by employing bootstrap sampling (with replacement) for it, which in turn is used to learn a classifier model, M_i . To classify an unknown or test tuple, *X*, each classifier, M_i , returns its class prediction, which counts as one vote. Assume the number of classifiers as follows, which predicts the same class, c_j , given the test tuple *X*:

$$v_j(X) = \left| \{ M_i(x) = \mathrm{c}_j \middle| i \in \left[1, K\right] \} \right|$$

The bagged classifier, M_*, counts the votes and assigns the class with the most votes to X. Each vote has the same weight in the equation;

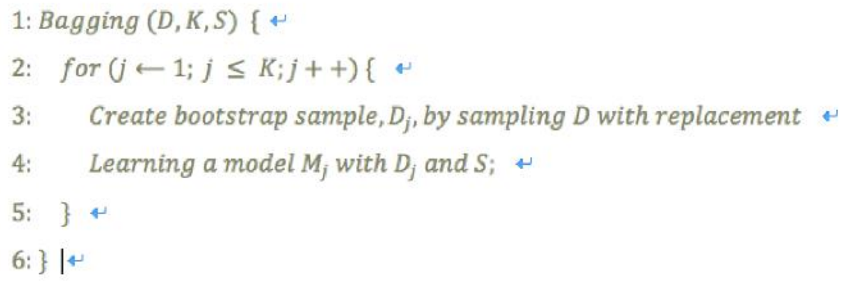
$$M_*\left(X\right) = \underbrace{argmax}_{c_j}\{v_j\left(X\right)|j \in \left[1, K\right]\}$$

About the prediction of continuous values, the average value of each prediction for a given test tuple is used as the result. The algorithm reduces the variance, given a more correct result than the base classifiers.

The input parameters for bagging algorithm are as follows:

- *D*: This is the training tuples dataset
- *K*: This is the number of classifiers combined
- *S*: This is a classification learning algorithm or scheme to learning base classifier
- M_*: This is the ensemble classifier, which is the output of the algorithm

The summarized pseudocode for the bagging algorithm is as follows:



```
Bagging $(D, K, S)$ {
  for $(j \leftarrow 1;\ j \leq K; j++)$ {
    Create bootstrap sample, $D_j$, by sampling $D$ with replacement
    Learning a model $M_j$ with $D_j$ and $S$;
  }
}
```

The boosting and AdaBoost algorithms

As opposed to an ensemble algorithm, the bagging algorithm is the weighted voting and weighted sample training tuple dataset for each base classifier. The base classifiers are learned iteratively. Once a classifier is learned, the relative weights are updated with a certain algorithm for the next learning of the base classifiers. The successive model learning will emphasize the tuples misclassified by the former classifier. As a direct result, the accuracy of a certain classifier will play an important role in the voting of the final label once an unknown test tuple is provided for the combined classifier.

Adaptive Boosting or AdaBoost is one of the boosting algorithms. If it contains *K* base classifiers, then the AdaBoost will be performed as *K* passes. Given the tuple dataset D_i and its corresponding classifier M_i, the error rate $error(M_i)$ of classifier M_i is defined as follows:

$$error(M_i) = \sum_{j=1}^{d} w_j * error\left(X_j\right)$$

The new classifier M_i will be discarded once the error (M_i) is bigger than 0.5. The training tuples set will be resampled for this classifier and perform the training of this classifier from scratch.

For tuple $\forall X_j \in D_i$, the error function is as follows:

$$error\left(X_j\right) = 1\, for\, mis - classified, 0\, for\, others$$

All the weights of the tuples in the training tuples dataset D_i are initialized with $\frac{1}{d}, d = |D_i|$. When the classifier M_i is learned from the training tuples set D_i, the weight for the tuple, which is correctly classified, is multiplied by $\frac{error\left(M_i\right)}{1-error\left(M_i\right)}$. After the updates, the weights for all tuples are normalized, which means the weight of the classified tuple increases and the weight of the others decreases.

The weight of the vote of classifier M_i is as follows:

$$\log \frac{1-error\left(M_i\right)}{error\left(M_i\right)}$$

We defined $v_j\left(x\right)$ as the weight of the vote for class c_j upon the K classifiers, α_i representing the weight of the *i*th classifier:

$$v_j\left(X\right) = \sum_{i=1}^{K} \alpha_i * I\left(M_t\left(x\right) = c_j\right), I\left(M_t\left(x\right) = c_j\right) = 1\, when\, M_t\left(x\right) = c_j, 0\, for\, others$$

The AdaBoost combined classifier, M_*, counts the votes with their respective weights multiplied and assigns the class with the most votes to X. Each vote has the same weight in the equation:

$$M_*\left(X\right) = \underbrace{argmax}_{c_j}\{v_j\left(X\right)|j \in \left[1, K\right]\}$$

The input parameters for the AdaBoost algorithm are as follows:

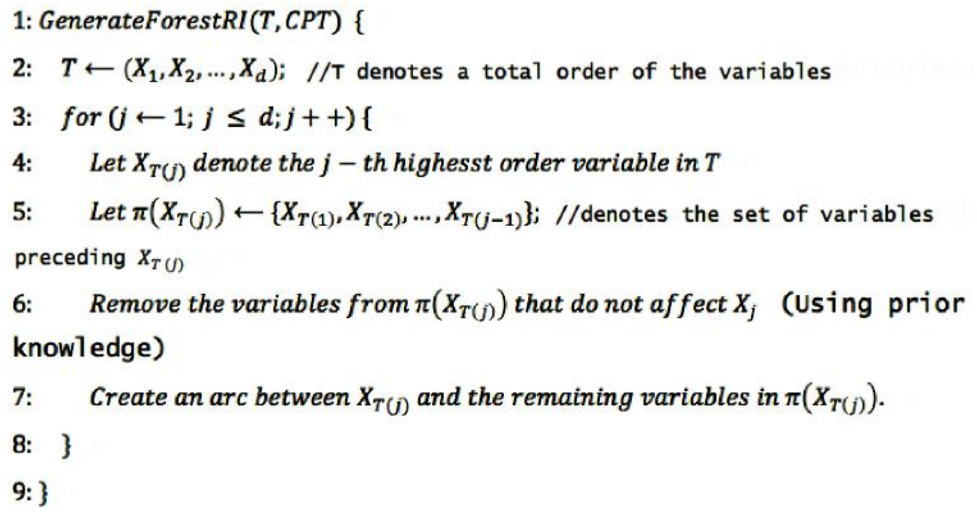
- *D*, which denotes a set of training tuples
- *k*, which is the number of rounds
- A classification learning algorithm

The output of the algorithm is a composite model. The pseudocode of AdaBoost is listed here:

```
1:  $Adaboost\left(D, K, S\right)$ {
2:    $set\ the\ initial\ value\ as\ \frac{1}{d}\ for\ the\ weight\ of\ each\ training\ tuple;$
3:    $for\left(j \leftarrow 1; j \le k; j++\right)$ {
4:        $Create\ bootstrap\ sample, D_j, by\ sampling\ D\ with\ replacement$
5:        $Sample\ D\ with\ replacement\ according\ to\ the\ tuple\ weights\ to\ get\ D_j;$
6:       $Learning\ a\ model\ M_j\ with\ D_j;$
7:       $compute\ the\ error\ rate\ of\ M_j, i.e., error\left(D_j\right);$
8:       $if\left(error\left(M_j\right) > 0.5\right)$ {
9:          $go\ back\ to\ step\ 3\ and\ try\ again;$
10:      }
11:      $for\left(each\ tuple\ in\ D_j\ that\ was\ corretly\ classified\right)$ {
12:         $updates\ its\ weight\ value\ by\ multiplying\ with\ \frac{error\left(M_j\right)}{1 - error\left(M_j\right)};$
13:      }
14:     $normalize\ the\ weight\ of\ each\ tuple;$
15:   }
16: }
```

The Random forests algorithm

Random forests algorithm is an ensemble method to combine a group of decision trees, which is generated by a strategy of applying a random selection of attributes at each node that will be split. Given an unknown tuple, each classifier votes, and the most popular one decides the final result. The pseudocode for the ForestRI algorithm to generate a forest is as follows:



```
1: $GenerateForestRI(T, CPT)$ {
2:   $T \leftarrow (X_1, X_2, \ldots, X_d)$; //T denotes a total order of the variables
3:   $for\ (j \leftarrow 1;\ j \le d; j++)$ {
4:     $Let\ X_{T(j)}\ denote\ the\ j - th\ highesst\ order\ variable\ in\ T$
5:     $Let\ \pi(X_{T(j)}) \leftarrow \{X_{T(1)}, X_{T(2)}, \ldots, X_{T(j-1)}\}$; //denotes the set of variables preceding $X_{T(j)}$
6:     $Remove\ the\ variables\ from\ \pi(X_{T(j)})\ that\ do\ not\ affect\ X_j$ (Using prior knowledge)
7:     $Create\ an\ arc\ between\ X_{T(j)}\ and\ the\ remaining\ variables\ in\ \pi(X_{T(j)})$.
8:   }
9: }
```

`T` denotes a total order of the variables in line 2. In line 5, $\pi\left(X_{T(j)}\right)$ denotes the set of variables preceding $X_{T(j)}$. Prior knowledge is required for line 6.

Instead of random selection of attributes on splitting the node, for another algorithm, ForestRC, the random linear combination strategy of the existing attributes is used to split the task. New attributes are built by the random linear combination of the original attributes set. With a couple of new attributes added, the best split is searched over the updated attributes set including the new and original attributes.

The R implementation

Here we provide three R implementations, bagging, AdaBoost, and Random forests. Please look up the R codes file `ch_04_bagging.R`, `ch_04_adaboost.R`, `ch_04_forestrc.R`, and `ch_04_forestri.R` from the bundle of R codes for the previously mentioned algorithms. The codes can be tested with the following commands:

```
> source("ch_04_bagging.R")
> source("ch_04_adaboost.R")
> source("ch_04_forestrc.R")
> source("ch_04_forestri.R")
```

Parallel version with MapReduce

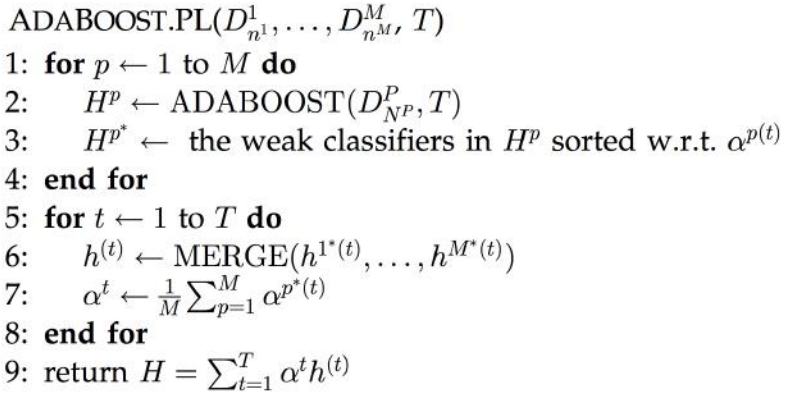
The following algorithm is the parallelized AdaBoost algorithm, which depends on a couple of workers to construct boosting classifiers. The dataset for the *p*th worker is defined using the following formula, where, n^p denoting its size is $p \in \{1,...,M\}$:

$$D^p_{n^p} = \{(\mathrm{x}^p_1, \mathrm{y}^p_1), (\mathrm{x}^p_2, \mathrm{y}^p_2), ..., (\mathrm{x}^p_{n^p}, \mathrm{y}^p_{n^p})\}$$

The classifier H^p is defined in the following format, with $\alpha^{p(\mathrm{t})}$ as the weight:

$$\{(h^{p(1)}, \alpha^{p(1)}), (h^{p(2)}, \alpha^{p(2)}), ..., (h^{p(T)}, \alpha^{p(T)})\}$$

The output is the final classifier. The input is the training dataset of M workers $(D_{n^1}{}^1, ..., D_{n^M}{}^M)$
.



```
ADABOOST.PL($D^1_{n^1}, \ldots, D^M_{n^M}$, $T$)
for $p \leftarrow 1$ to $M$ do
    $H^p \leftarrow$ ADABOOST($D^P_{N^P}, T$)
    $H^{p^*} \leftarrow$ the weak classifiers in $H^p$ sorted w.r.t. $\alpha^{p(t)}$
end for
for $t \leftarrow 1$ to $T$ do
    $h^{(t)} \leftarrow$ MERGE($h^{1^*(t)}, \ldots, h^{M^*(t)}$)
    $\alpha^t \leftarrow \frac{1}{M} \sum_{p=1}^{M} \alpha^{p^*(t)}$
end for
return $H = \sum_{t=1}^{T} \alpha^t h^{(t)}$
```

Biological traits and the Bayesian belief network

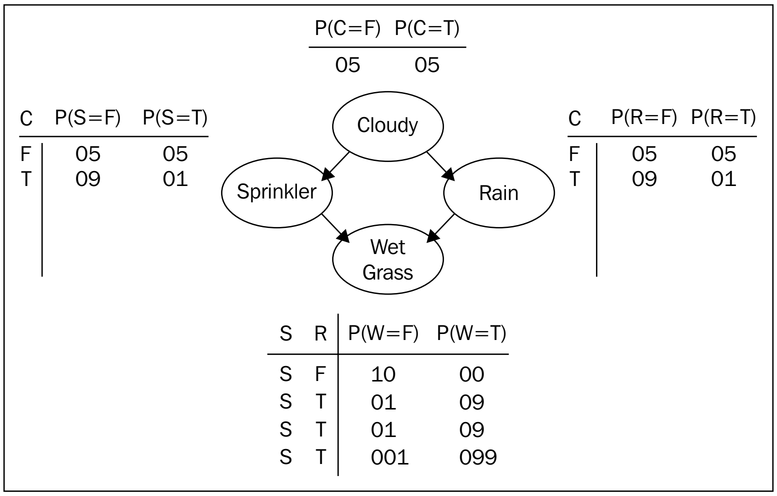
The Bayesian belief network, once trained, can be used for classification. Based on the Bayes' theorem, which is defined in the *The Bayes classification* section of *Chapter 3*, *Classification*, it is defined with two parts, one directed acyclic graph and **conditional probability tables** (**CPT**) for each variable; this is in turn represented by one node in the graph and models the uncertainty by graphically representing the conditional dependencies between distinct components. The arcs in the image give a representation of causal knowledge. The interaction among the diverse sources of uncertainty is also graphically illustrated.

The uncertainty comes from various sources:

- The way to associate the knowledge by the expert
- The domain intrinsic uncertainty
- The requirement of the knowledge to be translated
- The accuracy and availability of knowledge

Here is an example of the Bayesian belief network with four Boolean variables and the corresponding arcs. *Whether the grass is wet* is influenced by the work results of *sprinkler* and *whether it has just rained*, and so on. Each arc has a certain probability.

Let us have a look at the CPT representation of $P(WetGrass = T|Sprinkler = F, Rain = T) = 0.9$:



P(C=F)	P(C=T)
05	05

C	P(S=F)	P(S=T)
F	05	05
T	09	01

C	P(R=F)	P(R=T)
F	05	05
T	09	01

S	R	P(W=F)	P(W=T)
S	F	10	00
S	T	01	09
S	T	01	09
S	T	001	099

In the network, each variable is conditionally independent of its non-descendants. Here is the definition of the joint probability distribution:

$$P\left(x_1,x_2,\ldots,x_n\right)=\sum_{i=1}^{n}P(x_i \mid Parents\left(Y_i\right))$$

The Bayesian belief network (BBN) algorithm

Before the application of the BBN algorithm to classification, we need to train it first. In the process of training the network, the expert knowledge, that is, the prior knowledge, can be used in the training process to help the design of the network. For the variables that participated in direct dependency, experts must specify their conditional probability. There are many algorithms to learn the network from the training dataset; we will introduce an adaptive probabilistic networks algorithm.

The input parameters for the BBN algorithm are as follows:

- *T*, denotes a total order of the variables
- *CPT*

The output of the algorithm is the topology structure of BBN, which is as follows:

```
1: $GenerateBBN\left(T,CPT\right)\{$
2: $T \leftarrow \left(X_1,X_2,\ldots,X_d\right);$
3: $for\ (j \leftarrow 1; j \le d; j++)\{$
4:     $Let\ X_{T(j)}\ denote\ the\ j-th\ highesst\ order\ variable\ in\ T$
5:     $Let\ \pi\left(X_{T(j)}\right) \leftarrow \left\{X_{T(1)},X_{T(2)},\ldots,X_{T(j-1)}\right\};$
6:     $Remove\ the\ variables\ from\ \pi\left(X_{T(j)}\right) that\ do\ not\ affect\ X_j$
7:     $Create\ an\ arc\ between\ X_{T(j)}\ and\ the\ remaining\ variables\ in\ \pi\left(X_{T(j)}\right).$
8:   $\}$
9: $\}$
```

T denotes a total order of the variables in line 2. In line 5, $\pi\left(X_{T(j)}\right)$ denotes the set of variables preceding $X_{T(j)}$.

The R implementation

Please look up the R codes file `ch_04_bnn.R` from the bundle of R codes for the previously mentioned algorithms. The codes can be tested with the following command:

```
> source("ch_04_bnn.R")
```

Biological traits

A biological trait is one of the important applications of the BBN algorithm.

Protein classification and the k-Nearest Neighbors algorithm

The **k-Nearest Neighbors** (**kNN**) algorithm is one of the lazy learners that postpones the learning until the test tuple or test instance is provided.

A single training tuple is represented by a point in an *n*-dimensional space. In other words, *n* attributes' combinations are used to represent the specific training tuple. There is no specific training before the arrival of the test tuple that needs to be classified. Some preprocessing steps are needed, such as normalization for some attributes with large values compared to other attributes' values. Data normalization approaches in the data transformation can be applied here for preprocessing.

When a test tuple is given, the k-nearest training tuples are found from the training tuples space by a specific measure to calculate the distance between test tuple and the training tuple. The k-nearest training tuples are also known as the kNN. One popular solution is the Euclidean distance in real space, illustrated in the following equation. This method is only applicable to numeric attributes:

$$distance\left(X_p, X_q\right) = \sqrt{\sum_{i=1}^{n}\left(x_{qi} - x_{qi}\right)^2}, X_p, X_q \in R^n$$

For nominal attributes, one solution is that the difference between two attribute values is defined as *1*, or as *0*. We already know that many approaches deal with missing values in the attributes. With a predefined threshold, the value of *k* is selected with the number of tuples with the lowest error-rate among all the training tuples.

The class label of the test tuple is defined by the voting of the most common class in the kNN.

The kNN algorithm

The input parameters for kNN algorithm are as follows:

- *D*, the set of training objects
- *z*, the test object, which is a vector of attribute values
- *L*, the set of classes used to label the objects

The output of the algorithm is the class of z, represented as $c_z \in L$.

The pseudocode snippet for kNN is illustrated here:

```
1: $GenerateKNN(D, z, L)$ {
2:   $for\ (each\ object\ y \in D)$ {
3:     $Compute\ d(z, y), the\ distance\ between\ z\ and\ y;$
4:   }
5:   $Select\ N \subseteq D, the\ set\ (neighborhood)\ of\ k\ closest\ training\ object\ for\ z;$
6:   $\boldsymbol{c}_z = \text{argmax}_{v \in L} \sum_{y \in L} I\left(v = class\left(c_y\right)\right), I\left(v = class\left(c_y\right)\right),$   I() is an indicator function that
returns the value 1 if its argument is true and 1 otherwise.
7: }
```

The *I* function in line 6 denotes an indicator function that returns the value `1` if its argument is true and `0` otherwise.

The R implementation

Please look up the R codes file `ch_04_knn.R` from the bundle of R codes for the previously mentioned algorithm. The codes can be tested with the following command:

```
> source("ch_04_knn.R")
```

Document retrieval and Support Vector Machine

Support Vector Machine (**SVM**) is a classification algorithm applicable to both linear and nonlinear data classification. It is based on an assumption: if two classes of data cannot be divided by a hyper-plane, then after mapping the source dataset to sufficient higher dimension spaces, the optimal separating hyper-plane must exist.

Here are two concepts that need to be clearly defined:

- **Linearly separable**: This means that a dataset can be divided into the target classes with a linear equation with the input of a training tuple.
- **Nonlinearly separable**: This means that none of the linear equations exist in the space with the same dimension as that of the training tuple.

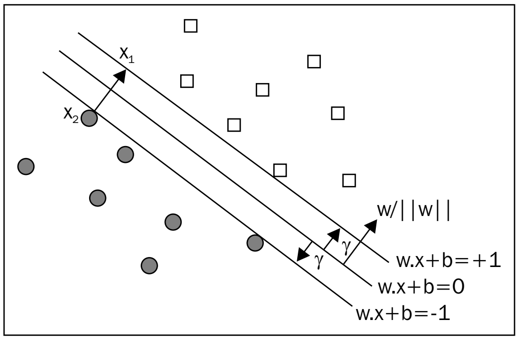
The linear hyper-plane can be represented as the linear discriminant equation, given the weight vector w and the training tuple x,

$$\boldsymbol{W} = \left(\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_{\boldsymbol{n}}\right)$$

$$\boldsymbol{X} = \left(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{\boldsymbol{n}}\right)$$

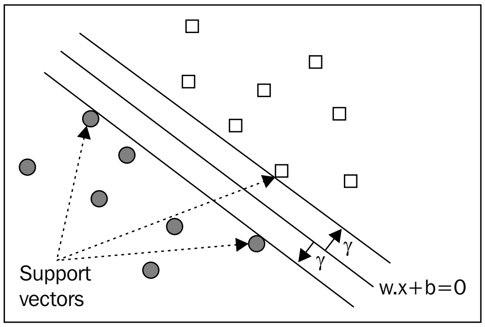
$$g\left(\boldsymbol{X}\right) = \boldsymbol{W} * \boldsymbol{X}^T + b = 0$$

With the preceding equation, we have the following image to illustrate a hyper-plane:

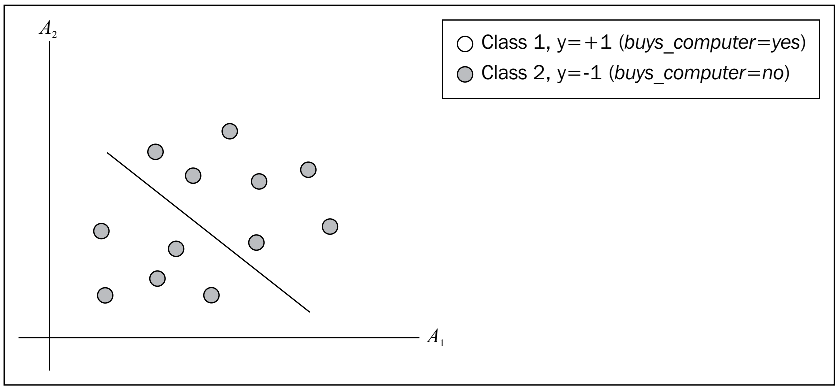


The target of SVM is to find the optimal hyper-plane, by which the margin between data points belonging to different classes are maximized.

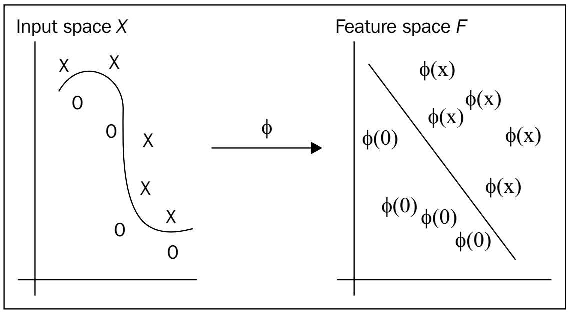
There are two hyper-planes with equal distance and that are parallel to the $g(\boldsymbol{X}) = \boldsymbol{W} * \boldsymbol{X}^T + b = 0$ hyper-plane. They are boundary hyper-planes and all support vectors are on them. This is illustrated in the following diagram:



In the following diagram, the case of a nonlinearly separable case is illustrated:


A_2
○ Class 1, y=+1 (*buys_computer=yes*)
● Class 2, y=-1 (*buys_computer=no*)
A_1

In the following diagram, after mapping the vector from a low-dimensional space to high-dimensional space, the nonlinearly separable case will be transformed into to a linearly separable case:



The SVM algorithm

The input parameters for a dual SVM algorithm are as follows:

- *D*, the set of training objects
- *K*
- *C*
- ε

The output of the algorithm is the SVM algorithm. The pseudocode snippet for this algorithm is illustrated here:

```
$DualSVM(D,K,C,\in)$ {
    $for(each\ x_j \in \mathrm{D})$ {
        $x_j \leftarrow \begin{pmatrix} x_j \\ 1 \end{pmatrix}$
    }
    $if(loss = quadratic)${
        $K \leftarrow \{K(x_i,x_j) + \frac{1}{2*C}\delta_{ij}\}, i, j = 1,...,n$
    }$else\ if(loss = longe)${
        $K \leftarrow \{K(x_i,x_j)\}, i, j = 1,..,n$
    }
    $for(each\ j \in [1,n])$ {
        $\eta_j \leftarrow \frac{1}{K\left(x_{j,}x_j\right)}$
    }
$t \leftarrow 0$
$\alpha_0 \leftarrow (0,...,0)^T$
$do\{$
    $\alpha \leftarrow \alpha_t$
    $for(k = 1; k \le 1; k++\}\{$
        $\alpha_k \leftarrow \alpha_k + \eta_k * \left(1 - y_k * \left(\sum_{i=1}^{n} \alpha_i * y_i * K(x_i, x_k)\right)\right)$
        $if(\alpha_k < 0)\alpha_k \leftarrow 0$
        $if(\alpha_k < C)\alpha_k \leftarrow C$
    $\}$
    $\alpha_{t+1} \leftarrow \alpha$
    $t \leftarrow t+1$
$\}until(\|\alpha_t - \alpha_{t-1}\| \le \varepsilon)$
$\}$
```

Here is the pseudocode for another version of the SVM algorithm, which is the primal kernel SVM algorithm. The input parameters for the primal kernel SVM algorithm are as follows:

- *D*, the set of training objects
- *K*
- *C*
- $\mathcal{E}$

The output of the algorithm is the SVM model. The pseudocode snippet for this algorithm is illustrated here:

```
$PrimalKernalSVM(D, K, C, \varepsilon)\{$
    $for(each\ x_j \in D)\{$
        $x_j \leftarrow \begin{pmatrix} x_j \\ \overline{1} \end{pmatrix}$
    $\}$
    $K \leftarrow \{K(x_i, x_i)\}, i, j = 1,.., n$
    $t \leftarrow 0$
    $\beta_0 \leftarrow (0,...,0)^T, \beta_0, \beta_t \in R^n$
    $do\{$
        $\text{v} \leftarrow \sum_{yi(K_i^T * \beta_t)<1} y_i * K_i$
        $s \leftarrow \sum_{yi(K_i^T * \beta_t)<1} y_i * K_i$
        $\nabla \leftarrow (K + 2CS)\beta_t - 2C * v$
        $H \leftarrow K + 2CS$
        $22 : \beta_{t+1} \leftarrow \beta_t - \eta_t * H^{-1} * \nabla$
        $t \leftarrow t + 1$
    $\}until(\|\beta_t - \beta_{t-1}\| \le \varepsilon)$
$\}$
```

The R implementation

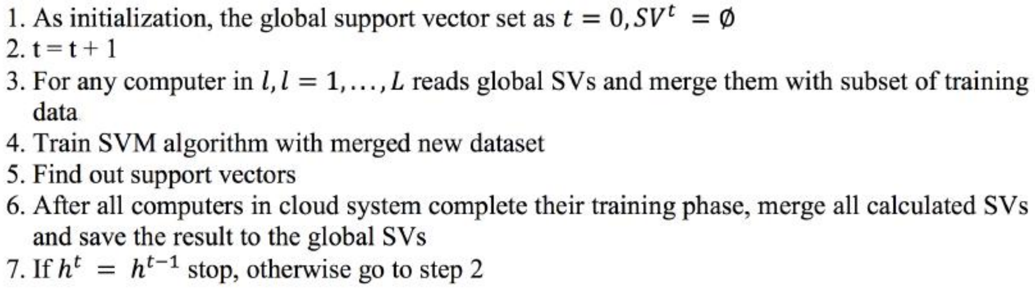
Please look up the R codes file `ch_04_svm.R` from the bundle of R codes for the previously mentioned algorithm. The codes can be tested with the following command:

```
> source("ch_04_svm.R")
```

Parallel version with MapReduce

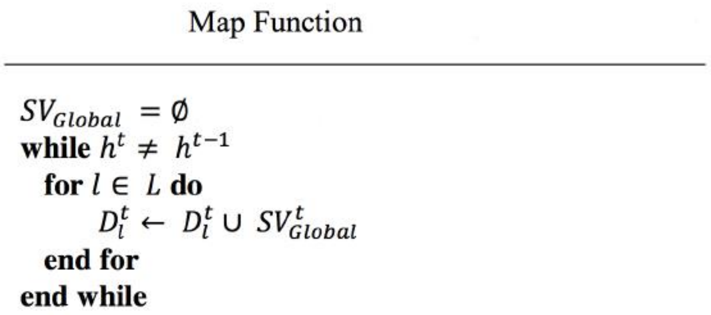
Along with the online computation requests, such as mobile cloud applications for classification, a high-performance, robust, and high-accuracy classification platform or algorithm is widely required. The parallelized SVM distributes the computing of optimization by the MapReduce technique to achieve working on a large scale of datasets and high performance at the same time. There are many implementations of various versions of parallelized SVM. One of them is shown here.

The `t` denotes for iteration number, `l` for MapReduce function size, h^t for the best hypothesis at iteration `t`, D_l for sub-dataset for node `l`, SV_l for support vectors at node `l` and SV_{Global} for global support vector:



1. As initialization, the global support vector set as $t = 0, SV^t = \emptyset$
2. t = t + 1
3. For any computer in $l, l = 1, \ldots, L$ reads global SVs and merge them with subset of training data
4. Train SVM algorithm with merged new dataset
5. Find out support vectors
6. After all computers in cloud system complete their training phase, merge all calculated SVs and save the result to the global SVs
7. If $h^t = h^{t-1}$ stop, otherwise go to step 2

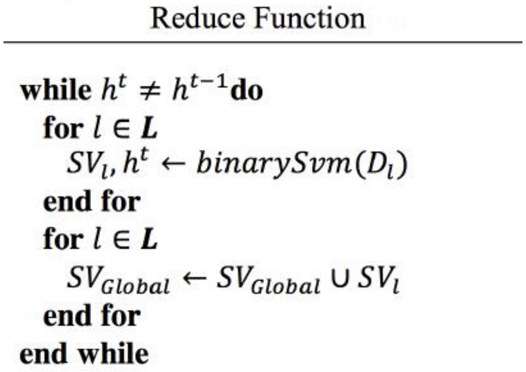
Next, the mapper algorithm is designed and the for `loop` is immediately behind the while `loop`, which is looped for each subset:



Map Function

```
$SV_{Global} = \emptyset$
while $h^t \neq h^{t-1}$
  for $l \in L$ do
        $D_l^t \leftarrow D_l^t \cup SV_{Global}^t$
  end for
end while
```

Finally, the reducer algorithm is designed. On line 3, the code inside the for `loop` immediately follows the while `loop`. This is for training by merging the datasets to obtain support vectors and binary-class hypothesis:



Reduce Function

```
while $h^t \neq h^{t-1}$ do
  for $l \in L$
    $SV_l, h^t \leftarrow binarySvm(D_l)$
  end for
  for $l \in L$
    $SV_{Global} \leftarrow SV_{Global} \cup SV_l$
  end for
end while
```

Document retrieval

One of the important applications of SVM is document retrieval, which has a static information source; the task is to fetch a ranking of documents as a response to a user request. The vector model is a widely implemented document-retrieval or information-retrieval model.

Classification using frequent patterns

There are two types of classification using frequent patterns:

- Associative classification model as well as association rules, which are generated from frequent patterns and used for classifications
- Discriminative frequent pattern-based classification

The associative classification

The generic association classification algorithm is defined here. The input parameters for the kNN algorithm are as follows:

- *D*, which is a set of training objects
- *F*, which is the itemset
- *MIN_SUP*, which is the minimal support
- *MIN_CONF*, which is the minimal confidence

The output of the algorithm is a rule-based classifier and is shown as follows:

```
GenerateGenericAC(D, F, MIN_SUP, MIN_CONF){
    Perform association mining, satisfying the minimal support and confidence too,
    Build a rule-based classifier based on the transformation of result rules
}
```

Two popular algorithms are illustrated in the successive sections, one is **Classification Based on Association** (**CBA**), and the other is **Classification Based on Multiple Association Rules** (**CMAR**).

CBA

Here is the pseudocode for CBA:

```
GenerateCBA(D, F, MIN_SUP, MIN_CONF){
    Perform association mining
    Let R is the resulting rules set
    Let C be a empty rule − based classifier
    for(each r in R){
        Intergrating r into the C with an improved rule-based classifier
    }
}
```

Discriminative frequent pattern-based classification

Here is the pseudocode for discriminative frequent pattern-based classification:

```
GenerateDFPC(D, F, MIN_SUP, MIN_CONF){
    Perform association mining, satisfying the minimal support and confidence
    Let R is the resulting rules set, R is ordered in decreasing precedence based,
on the MIN _SUP and MIN _CONF, for the rules with same antecedent,
the one with highest confidence is kept, others discarded
    Let C be a empty rule-based classifier
    for(each r in R){
        Intergrating r into the C with an improved rule-based classifier
    }
}
```

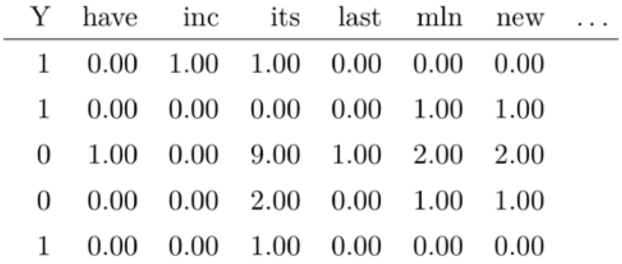
The R implementation

Please look up the R codes files `ch_04_associative_classification.R`, `ch_04_cba.R`, and `ch_04_frequent_pattern_based_classification.R` from the bundle of R codes for the previously mentioned algorithms. The codes can be tested with the following commands:

```
> source("ch_04_associative_classification.R")
> source("ch_04_cba.R")
> source("ch_04_frequent_pattern_based_classification.R")
```

Text classification using sentential frequent itemsets

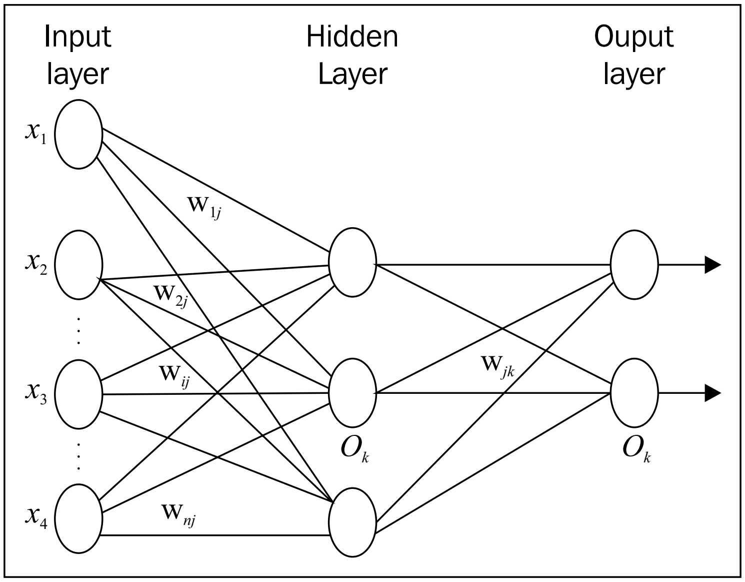
One application of CBA is text classification. The key here is to build a matrix with a document or text term and labels. With the matrix built, any classification algorithm can be applied to it. One example of document matrix is illustrated here. The term might include a character, word, phrase, and concept and so on.



Y	have	inc	its	last	mln	new	...
1	0.00	1.00	1.00	0.00	0.00	0.00	
1	0.00	0.00	0.00	0.00	1.00	1.00	
0	1.00	0.00	9.00	1.00	2.00	2.00	
0	0.00	0.00	2.00	0.00	1.00	1.00	
1	0.00	0.00	1.00	0.00	0.00	0.00	

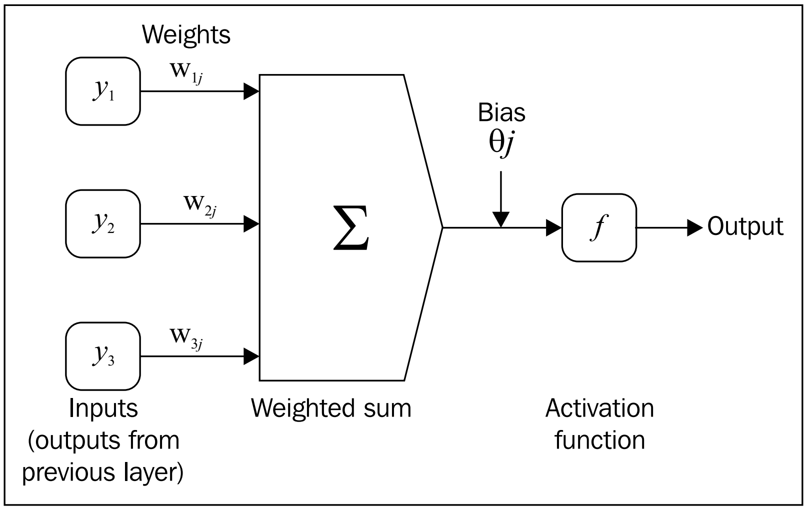
Classification using the backpropagation algorithm

The **backpropagation** (**BP**) algorithm learns the classification model by training a multilayer feed-forward neural network. The generic architecture of the neural network for BP is shown in the following diagrams, with one input layer, some hidden layers, and one output layer. Each layer contains some units or perceptron. Each unit might be linked to others by weighted connections. The values of the weights are initialized before the training. The number of units in each layer, number of hidden layers, and the connections will be empirically defined at the very start.



The training tuples are assigned to the input layer; each unit in the input layer calculates the result with certain functions and the input attributes from the training tuple, and then the output is served as the input parameter for the hidden layer; the calculation here happened layer by layer. As a consequence, the output of the network contains all the output of each unit in the output layer. The training is performed iteratively by updating the weights of the connections with the feedback of errors, that is, back propagation of the errors.

The prototype for the unit in the hidden/output layer is shown here, the one in the input layer with only one input and other minor differences compared to this. W_{1j} denotes the weight related to that link or connection. Each unit is bound with a bias, which is θ. The threshold or activation function bound with each unit is f.



For a certain unit or perceptron in the hidden or output layer, the net input is a combination of the linear combination of each of its input from the previous unit, that is, the output of it, which is O_p. Let *k* denote the number of input connections of the unit *q*:

$$I_q = \sum_{p=1}^{k} w_{pq} * O_p + \theta_q$$

The output of this unit *q* is O_q.

$$O_q = \frac{1}{1+e^{-I_q}}$$

If the unit *q* is in the output layer and T_q denotes the expected or known output value in the training tuple, then its error Err_q can be calculated as follows:

$$Err_q = O*(1-O_q)*(T_q - Q_q)$$

If the unit *q* is in the hidden layer, then let w_{qp} denote the weight of the connection from unit *q* to one unit with the error Err_p in the next layer or the known output value in the training tuple. We can then calculate its error as Err_q. Let *M* denote the number of output connections of the unit *q*:

$$Err_q = O * (1 - O_q) * \sum_{p=1}^{M} Err_p * w_{qp}$$

After this preprocessing or preparation, the weights and biases can be updated accordingly with the backpropagation strategy. η denotes the learning rate; it is an empirical parameter, which belongs to `(0,1)`:

$$\Delta w_{pq} = (\eta) * Err_q * O_p$$

$$w_{pq} = w_{pq} + \Delta w_{pq}$$

$$\Delta \theta_q = (\eta) + Err_q$$

$$\theta_q = \theta_q + \Delta \theta_q$$

The weights and biases are updated per tuple of the training tuple dataset.

The BP algorithm

The input parameters for the BP algorithm, the topology of the neural network, the number of hidden layers, and the connections, are defined before the start of the training:

- *D*, which is the training tuples set
- *W*, which is the initial values for all the weights.
- θ, which is the bias for each units
- *I*, which is the learning rate

The output of the algorithm is the topology structure of BP, which consists of:

- *BPNN*, which is the trained BP neural network
- *W*, which is a set of weights of the connections in the neural network

Here is the pseudocode for the training of the backpropagation network:

```
$GenerateBPNN(D,W,\theta,I)\{$
  $Intialized\ all\ the\ weights\ and\ biases\ in\ network;$
  $While(termination\ condition\ is\ not\ true)\{$
      $for(each\ tuple\ X\ in\ D)\{$
      $for(each\ input\ layer\ unit\ j)\{$
          $O_j \leftarrow I_j$
      $\}$
      $for(each\ children\ or\ output\ layer\ unit\ j)\{$
          $I_j \leftarrow \sum_i w_{ij} O_i + \theta_j$
          $O_j \leftarrow \frac{1}{1-e^{-I_j}}$
      $\}$
      $for(each\ output\ layer\ unit\ j)\{$
          $Err_j \leftarrow O_j * (1-O_j) * (T_{j\_}O_j)$
      $\}$
      $for(each\ output\ layer\ unit\ j)\{$
          $Err_j \leftarrow O_j * (1-O_j) * (T_{j\_}O_j)$
      $\}$
      $for(each\ unit\ in\ the\ hidden\ layers, from\ last\ to\ first\ hidden\ layer)\{$
          $Err_j \leftarrow O_j * (1-O_j) * \sum_k Err_k * w_{jk}$
      $\}$
      $for(each\ weight\ w_{ij}\ in\ network)\{$
          $\Delta w_{ij} \leftarrow (l) * Err_j * O_i$
          $w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$
      $\}$
      $for(each\ bias\ \theta_j\ in\ network)\{$
```

```
            $\Delta\theta_j \leftarrow (l) * Err_j$
            $\theta_j \leftarrow \theta_j + \Delta\theta_j$
        }
      }
  }
}
```

The R implementation

Please look up the R codes file `ch_04_bp.R` from the bundle of R codes for the BP algorithm. The codes can be tested with the following command:

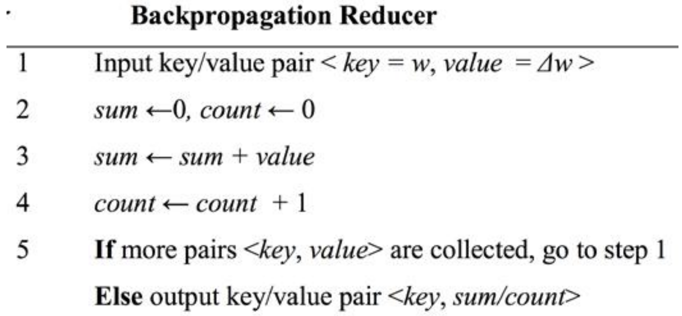
```
> source("ch_04_bp.R")
```

Parallel version with MapReduce

Massive data makes the training process of BP dramatically slow. The parallelized version of BP is proven to accelerate the speed in an amazing way. There are many versions of parallelized BPNN algorithms implemented on the MapReduce architecture. Here is one implementation, the **MapReduce-based Backpropagation Neural Network** (**MBNN**) algorithm.

Given the training dataset, each mapper is fed with one single training item. During the mapper tasks stage, new values for weights are calculated. Then, within the reducer tasks stage, new values for one weight are collected, and this results in an average value of these values to output for the weights. After the new values are provided, all the weights are updated, batch-wise. These steps are executed iteratively until the termination condition is matched.

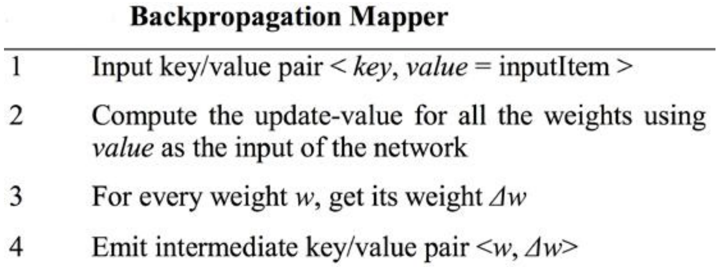
The backpropagation main algorithm is listed as follows:



Backpropagation Reducer

```
1  Input key/value pair < $key = w$, $value = \Delta w$ >
2  $sum \leftarrow 0$, $count \leftarrow 0$
3  $sum \leftarrow sum + value$
4  $count \leftarrow count + 1$
5  If more pairs <key, value> are collected, go to step 1
   Else output key/value pair <key, sum/count>
```

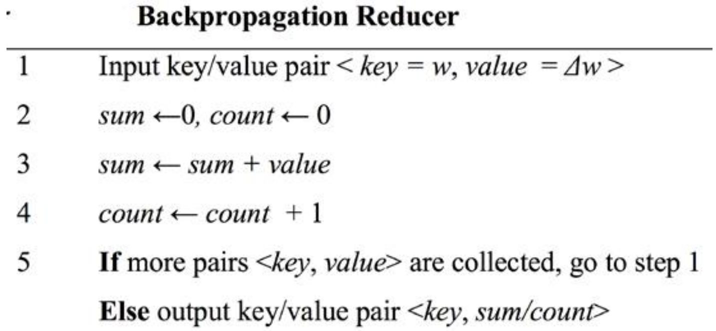
Four steps compose the backpropagation mapper algorithm. They are listed here:



Backpropagation Mapper

```
1  Input key/value pair < key, value = inputItem >
2  Compute the update-value for all the weights using
   value as the input of the network
3  For every weight w, get its weight $\Delta w$
4  Emit intermediate key/value pair <w, $\Delta w$>
```

Five steps compose the backpropagation reducer algorithm. They are listed here:



Backpropagation Reducer

```
1  Input key/value pair < $key = w$, $value = \Delta w$ >
2  $sum \leftarrow 0$, $count \leftarrow 0$
3  $sum \leftarrow sum + value$
4  $count \leftarrow count + 1$
5  If more pairs <key, value> are collected, go to step 1
   Else output key/value pair <key, sum/count>
```

Time for action

Here are some practice questions for you to check your understanding about the concepts covered:

- Find one application of the classification using frequent patterns
- What is the BBN algorithm?
- What is the kNN algorithm?
- What is the backpropagation algorithm?
- What is SVM?

Summary

In this chapter, we looked at the following topics:

- Ensemble methods
- Bagging
- AdaBoost
- Random forests
- BBN, which provides a topology model of causal relationship; it is an eager learner.

> An eager learner is one that behaves in an opposite way to the lazy learner. The lazy learner postpones the learning until the test tuple or test instance is provided.

- The kNN algorithm, which is a lazy learner
- SVM, by which the original data is transformed to a higher dimension, and is separated with a hyper-plane; it is an eager learner
- Classification using frequent patterns; this is an eager learner
- Classification using the BP algorithm, which is a neural network trained with a descent gradient

In the next chapter, we will learn the clustering algorithm, which is also a kind of unsupervised classification with no predefined labels.

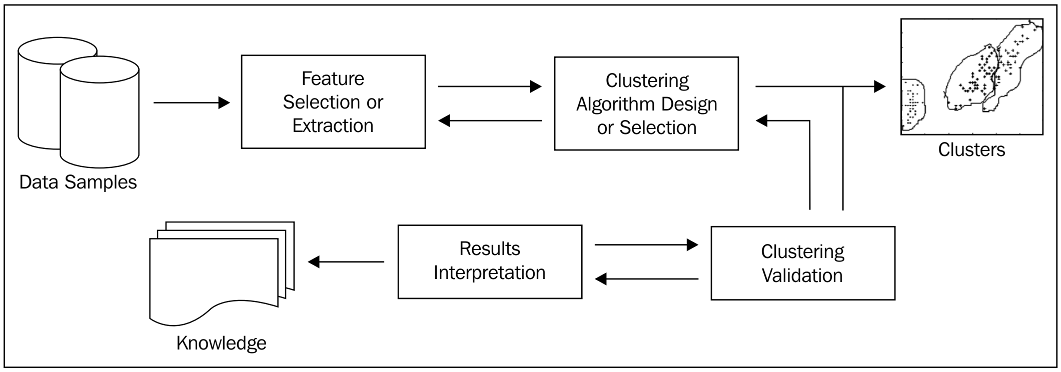
5
Cluster Analysis

Clustering is defined as an unsupervised classification of a dataset. The objective of the clustering algorithm is to divide the given dataset (a set of points or objects) into groups of data instances or objects (or points) with distance or probabilistic measures. Members in the same groups are closer by distance or similarity or by other measures. In other words, maximize the similarity of the intracluster (internal homogeneity) and minimize the similarity of the intercluster (external separation).

In this chapter, you will learn how to implement the top algorithms for clusters with R; these algorithms are listed here:

- Search engine and the k-means algorithm
- Automatic abstracting of document texts and the k-medoids algorithm
- The CLARA algorithm
- CLARANS
- Unsupervised image categorization and affinity propagation clustering
- Web page clustering and spectral clustering
- News categorization and hierarchical clustering

A clustering algorithm is used for the preparation of further analysis, while the other objective is to understand the nature of the dataset. The most common clustering process is illustrated in the following diagram:



The key steps in this process include the following:

- **Feature selection**: This step chooses distinguished features from the original dataset
- **Clustering algorithm design**: This step designs an appropriate algorithm based on the currently available clustering algorithms, or it just builds one from scratch
- **Cluster validation**: This step evaluates the clusters and provides a degree of confidence about the result
- **Result interpretation**: This step gives an intrinsic idea of the input dataset

There are many categorization methods to categorize clustering algorithms; the major categories are partition-based methods, density-based methods, hierarchical methods, spectral methods, grid-based methods, and so on.

Every clustering algorithm has its limitations and best practices for certain conditions or datasets. Once an algorithm is chosen, the parameters and distance measures (only for some algorithms) related to that algorithm need careful consideration as well. Then, we will list the most popular clustering algorithms and their corresponding parallel versions (if available).

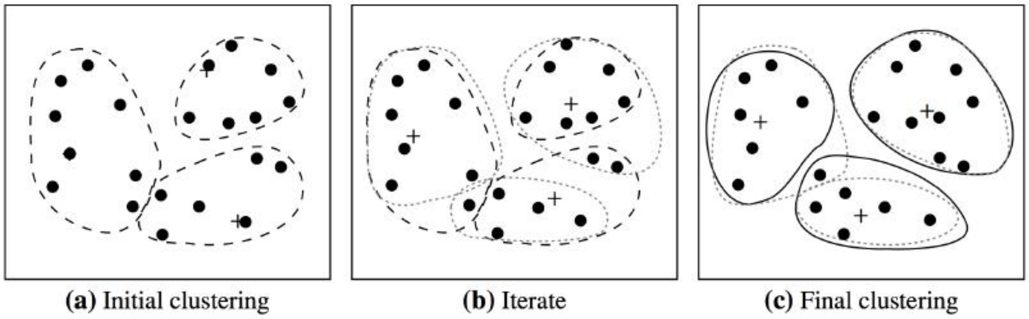
Here is the short list of clustering algorithms and their complexities:

Cluster algorithm	**Capability of tracking high dimensional data**
k-means	No
Fuzzy c-means	No
Hierarchical clustering	No
CLARA	No
CLARANS	No
BIRCH	No
DBSCAN	No
CURE	Yes
WaveCluster	No
DENCLUE	Yes
FC	Yes
CLIQUE	Yes
OptiGrid	Yes
ORCLUS	Yes

The difficulties or targets in another aspect of clustering algorithms are the arbitrary shape of clusters, large volume of datasets, high-dimensional datasets, insensitivity to noise or outlier, low reliance to user-defined parameters, capability of dealing with new data without learning after the initial learning or clustering, sensitivity to the intrinsic, or nature of the number of clusters, nice data visualization, and application to both numeric or nominal data types.

Search engines and the k-means algorithm

The general process of partition-based clustering is iterative. The first step defines or chooses a predefined number of representatives of the cluster and updates the representative after each iteration if the measure for the clustering quality has improved. The following diagram shows the typical process, that is, the partition of the given dataset into disjoint clusters:


(a) Initial clustering

(b) Iterate

(c) Final clustering

The characteristics of partition-based clustering methods are as follows:

- The resulting clusters are exclusive in most of the circumstances
- The shape of the clusters are spherical, because of most of the measures adopted are distance-based measures
- The representative of each cluster is usually the mean or medoid of the corresponding group (cluster) of points
- A partition represents a cluster
- These clusters are applicable for small-to-medium datasets
- The algorithm will converge under certain convergence object functions, and the result clusters are often local optimum

The k-means clustering algorithm is basically a greedy approach that defines the centroid of each cluster with its mean value. It is efficient for dealing with a large dataset. The k-means algorithm is a kind of exclusive clustering algorithm in which data is clustered in an exclusive way, and one object only belongs to, at the most, one group or cluster. It is also a type of partitional clustering algorithm in which clusters are created in one step, in contrast to a couple of steps.

The value of k is often determined by the domain knowledge, the dataset, and so on. At the start, k objects in the initial dataset D (the size of D is *n*, where $k \le n$) are randomly selected as the initial centers for the initial k clusters. In the iteration of the k-means algorithm, each object is assigned to the most similar or closest (various measures are used for distance or similarity) cluster (mean). Once the iteration ends, the mean for each cluster is updated or relocated. The k-means algorithm performs as much iteration as possible until there is no change that the clusters get from the previous clustering.

The quality of the specific cluster of the clustering algorithm is measured by various merits. One of them is formulated in the following equation. This is within the cluster variation measure, where c_i stands for the centroid of the cluster C_i. Here, *k* is the number of clusters, whereas $dist(p, c_i)$ represents the Euclidean distance of the two points. The minimum value of E is the needed value and depicts the best quality clusters. This evaluation or assessing objective function serves as the ideal one, though not practical for concrete problems. The k-means algorithm provides easy methods for an approximate-to-ideal value. The k-means algorithm is also known as the **squared error-based clustering algorithm**.

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} dist^2 \left(p, c_i \right)$$

$$\frac{1}{n} \sum_{j=1}^{n} \left[\min_j d^2 \left(p, c_i \right) \right]$$

In practice, the k-means algorithm can run several times with a different initial set of centroids to find a relatively better result.

The varieties of the k-means clustering algorithm are designed with different solutions for the selection of initial k centroids or means. The similarity or dissimilarity is measured, and the way to calculate the means is established.

The shortages of the k-means clustering method are listed as follows:

- The means of clusters must be defined with a function
- It is only applicable to numeric data type
- The value of k needs to be predefined by users, and it is difficult

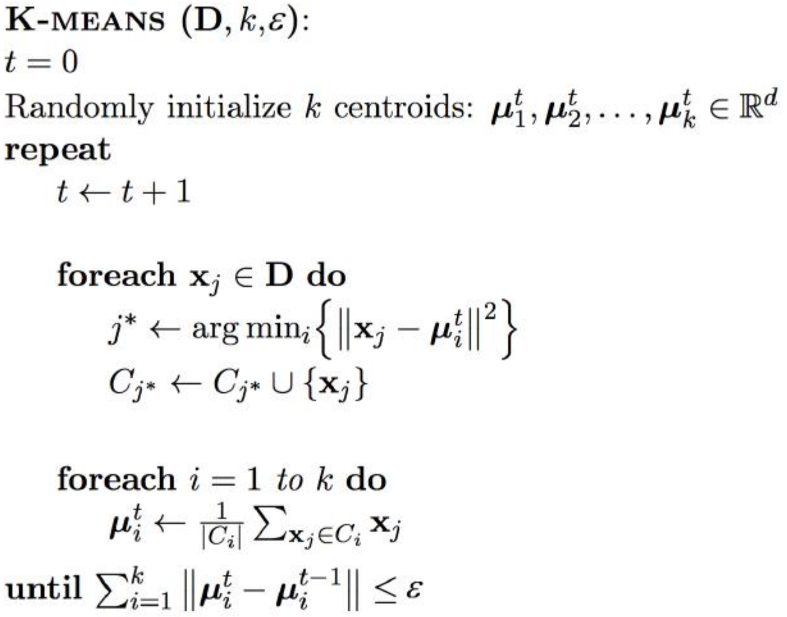
> The guidelines or thumb rules for the k-means clustering method are as follows:
>
> - Remember this method is sensitive to noise and outlier
> - This method is only applicable to clusters with closer sizes
> - This method finds it difficult to find nonconvex shapes

The k-means clustering algorithm

The input parameters for a dual SVM algorithm is as follows:

- D, which is the set of training objects
- K
- $\mathcal{E}$

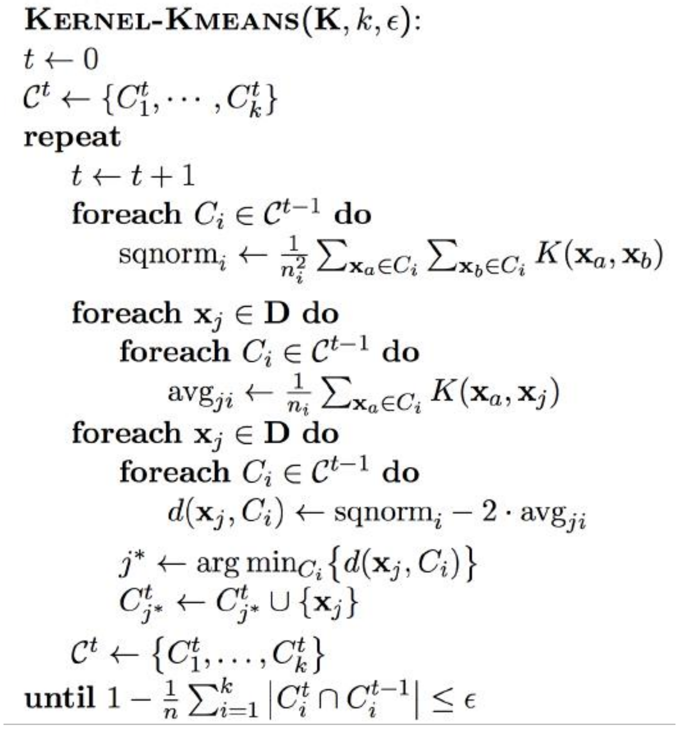
These parameters are used as depicted in the summarized pseudocodes of the k-means clustering algorithm given as follows:



```
K-MEANS $(\mathbf{D}, k, \varepsilon)$:
$t = 0$
Randomly initialize $k$ centroids: $\boldsymbol{\mu}_1^t, \boldsymbol{\mu}_2^t, \ldots, \boldsymbol{\mu}_k^t \in \mathbb{R}^d$
repeat
    $t \leftarrow t + 1$

    foreach $\mathbf{x}_j \in \mathbf{D}$ do
        $j^* \leftarrow \arg\min_i \left\{ \left\| \mathbf{x}_j - \boldsymbol{\mu}_i^t \right\|^2 \right\}$
        $C_{j^*} \leftarrow C_{j^*} \cup \{\mathbf{x}_j\}$

    foreach $i = 1$ to $k$ do
        $\boldsymbol{\mu}_i^t \leftarrow \frac{1}{|C_i|} \sum_{\mathbf{x}_j \in C_i} \mathbf{x}_j$
until $\sum_{i=1}^{k} \left\| \boldsymbol{\mu}_i^t - \boldsymbol{\mu}_i^{t-1} \right\| \leq \varepsilon$
```

The kernel k-means algorithm

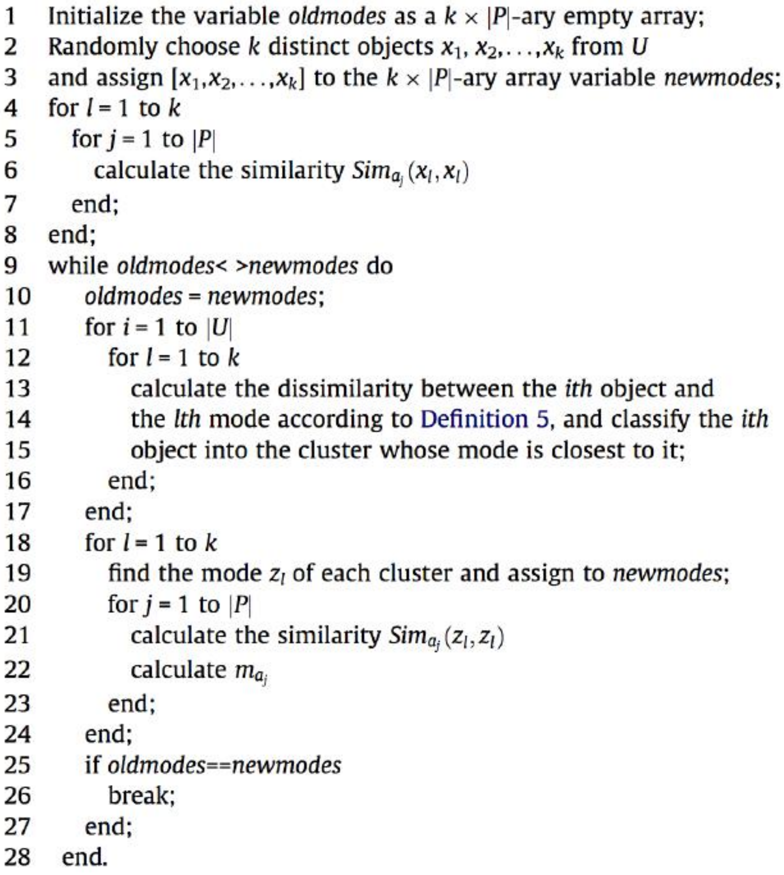
The summarized pseudocode for the kernel k-means clustering algorithm is as follows:



```
KERNEL-KMEANS($\mathbf{K}, k, \epsilon$):
$t \leftarrow 0$
$\mathcal{C}^t \leftarrow \{C_1^t, \cdots, C_k^t\}$
repeat
    $t \leftarrow t + 1$
    foreach $C_i \in \mathcal{C}^{t-1}$ do
        $\text{sqnorm}_i \leftarrow \frac{1}{n_i^2} \sum_{\mathbf{x}_a \in C_i} \sum_{\mathbf{x}_b \in C_i} K(\mathbf{x}_a, \mathbf{x}_b)$
    foreach $\mathbf{x}_j \in \mathbf{D}$ do
        foreach $C_i \in \mathcal{C}^{t-1}$ do
            $\text{avg}_{ji} \leftarrow \frac{1}{n_i} \sum_{\mathbf{x}_a \in C_i} K(\mathbf{x}_a, \mathbf{x}_j)$
    foreach $\mathbf{x}_j \in \mathbf{D}$ do
        foreach $C_i \in \mathcal{C}^{t-1}$ do
            $d(\mathbf{x}_j, C_i) \leftarrow \text{sqnorm}_i - 2 \cdot \text{avg}_{ji}$
        $j^* \leftarrow \arg\min_{C_i} \left\{d(\mathbf{x}_j, C_i)\right\}$
        $C_{j*}^t \leftarrow C_{j*}^t \cup \{\mathbf{x}_j\}$
    $\mathcal{C}^t \leftarrow \left\{C_1^t, \ldots, C_k^t\right\}$
until $1 - \frac{1}{n} \sum_{i=1}^k \left|C_i^t \cap C_i^{t-1}\right| \leq \epsilon$
```

The k-modes algorithm

This algorithm is a variant of the k-means algorithm; it can deal with categorical data types and large datasets. It can be integrated with the k-means method to deal with the clustering of the dataset that contains all data types. The algorithm is mentioned as follows:



```
Initialize the variable oldmodes as a $k \times |P|$-ary empty array;
Randomly choose $k$ distinct objects $x_1, x_2, \ldots, x_k$ from $U$
and assign $[x_1, x_2, \ldots, x_k]$ to the $k \times |P|$-ary array variable newmodes;
for $l = 1$ to $k$
  for $j = 1$ to $|P|$
    calculate the similarity $Sim_{a_j}(x_l, x_l)$
  end;
end;
while oldmodes< >newmodes do
   oldmodes = newmodes;
   for $i = 1$ to $|U|$
     for $l = 1$ to $k$
       calculate the dissimilarity between the ith object and
       the lth mode according to Definition 5, and classify the ith
       object into the cluster whose mode is closest to it;
     end;
   end;
   for $l = 1$ to $k$
     find the mode $z_l$ of each cluster and assign to newmodes;
     for $j = 1$ to $|P|$
       calculate the similarity $Sim_{a_j}(z_l, z_l)$
       calculate $m_{a_j}$
     end;
   end;
   if oldmodes==newmodes
     break;
   end;
 end.
```

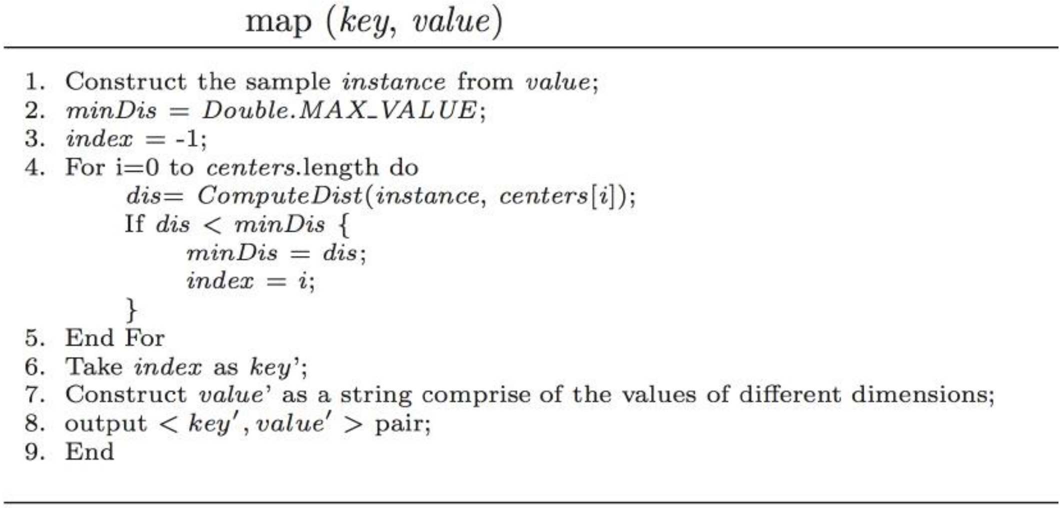
The R implementation

Take a look at the `ch_05_kmeans.R`, `ch_05_kernel_kmeans.R`, and `ch_05_kmedoids.R` R code files from the bundle of R code for the previously mentioned algorithms. The code can be tested using the following commands:

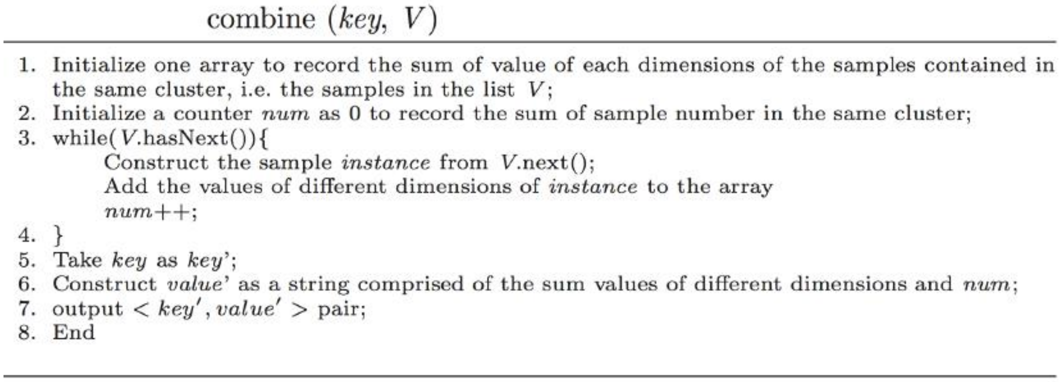
```
> source("ch_05_kmeans.R")
> source("ch_05_kernel_kmeans.R")
> source("ch_05_kmedoids.R")
```

Parallel version with MapReduce

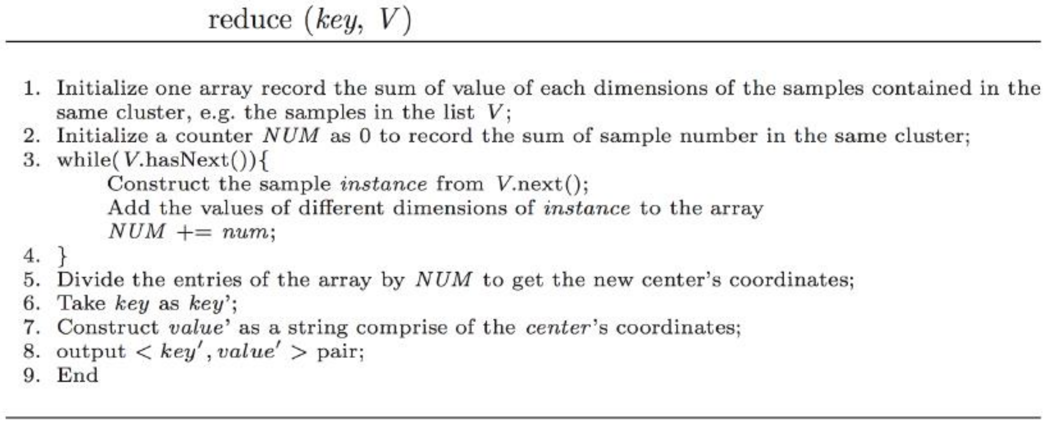
The parallelized k-means is listed as follows:



```
map (key, value)

1. Construct the sample instance from value;
2. $minDis = Double.MAX\_VALUE$;
3. $index = -1$;
4. For i=0 to centers.length do
       $dis = ComputeDist(instance, centers[i])$;
       If $dis < minDis$ {
           $minDis = dis$;
           $index = i$;
       }
5. End For
6. Take index as key';
7. Construct value' as a string comprise of the values of different dimensions;
8. output $< key', value' >$ pair;
9. End
```



```
combine (key, V)

1. Initialize one array to record the sum of value of each dimensions of the samples contained in
   the same cluster, i.e. the samples in the list V;
2. Initialize a counter num as 0 to record the sum of sample number in the same cluster;
3. while(V.hasNext()){
       Construct the sample instance from V.next();
       Add the values of different dimensions of instance to the array
       num++;
4. }
5. Take key as key';
6. Construct value' as a string comprised of the sum values of different dimensions and num;
7. output $< key', value' >$ pair;
8. End
```



```
reduce ($key$, $V$)

Initialize one array record the sum of value of each dimensions of the samples contained in the same cluster, e.g. the samples in the list $V$;
Initialize a counter $NUM$ as 0 to record the sum of sample number in the same cluster;
while($V$.hasNext()){
    Construct the sample $instance$ from $V$.next();
    Add the values of different dimensions of $instance$ to the array
    $NUM$ += $num$;
}
Divide the entries of the array by $NUM$ to get the new center's coordinates;
Take $key$ as $key$';
Construct $value$' as a string comprise of the $center$'s coordinates;
output $< key', value' >$ pair;
End
```

Search engine and web page clustering

Along with the nonstop accumulation of Internet documents, the difficulties in finding some useful information keeps increasing. To find information in these documents or web pages or from the Web, four search methodologies are provided to us:

- The unassisted keyword search
- The assisted keyword search
- The directory-based search
- The query-by-example search

Web page clustering is an important preprocessing step to web data mining, which is one solution among the many possible solutions. The document clustering occurs in the progress of IR and text clustering. Many web clustering criteria are provided, such as the semantic, the structure, the usage-based criteria, and so on. Domain knowledge plays an important role in web page clustering.

Term Frequency-Inverse Document Frequency (**TF-IDF**) is applied during the preprocessing of the document dataset. One modeling method to represent a data instance for document clustering is the **vector-space model**. Given a term space, each dimension with a specific term in the documents and any document in the original document dataset can be represented by a vector, as depicted in the following equation. This definition implies that mostly, frequently used terms play an important role in the document:

$$d_{tf} = \left(tf_1, tf_2, ..., tf_n\right)$$

Each value in a dimension denotes the frequency of the term that labels the dimension appearing in the document. For simplicity, the vector needs to be normalized as a unit length before processing it further, the stop words should be removed, and so on. As a potential and popular solution, the TF often weighs every term using the inverse document frequency among the document datasets. The inverse document frequency is denoted by the following formula in which *n* is the dataset size, and $\mathrm{df}\left(t_i\right)$ is the document's number that contains the term, t_i:

$$idf\left(t_i\right) = \log\left(\frac{n}{df(t_i)}\right)$$

Given the definition of inverse document frequency, there is another popular definition of the vector model to denote a document in the document collection for further processing using a clustering algorithm; $tf\left(d_i, t_i\right)$ is the frequency of term t_i in the document d_i:

$$d_j = (w_{j1}, \ldots, w_{jm})$$

$$w_{ji} = tf(d_j, t_i) \cdot idf(t_i)$$

The measures used in the document clustering are versatile and tremendous. Cosine method is one among them, and we will use the vector dot production in the equation while the corresponding centroid is defined as *c* in the successive equation:

$$cosine(d_1, d_2) = d_1 \cdot d_2 / \|d_1\| \|d_2\|$$

$$c = \frac{1}{|\mathrm{S}|}\sum_{d \in s} d$$

Reuters-21578 is one publicly-available document dataset for further research. TREC-5, 6, and 7 are also open source datasets.

With this measure definition, the k-means algorithm is used for web page clustering, as it displays its efficiency and always acts as a component for a practical web search engine.

Automatic abstraction of document texts and the k-medoids algorithm

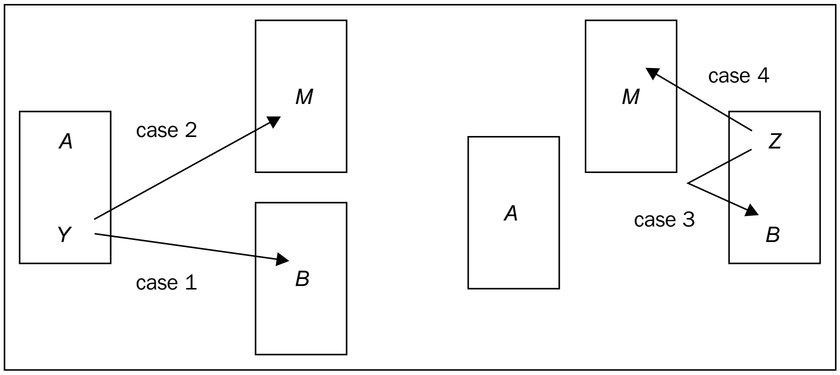
The k-medoids algorithm is extended from the k-means algorithm to decrease the sensitivity to the outlier data points.

Given the dataset *D* and the predefined parameter *k*, the k-medoids algorithm or the PAM algorithm can be described as shown in the upcoming paragraphs.

As per a clustering related to a set of k medoids, the quality is measured by the average distance between the members in each cluster and the corresponding representative or medoids.

An arbitrary selection of *k* objects from the initial dataset of objects is the first step to find the k medoids. In each step, for a selected object O_i and a nonselected node O_h, if the quality of the cluster is improved as a result of swapping them, then a swap is performed.

The cluster quality should be the sum of all the differences in the distance between the members and medoids, before and after the swap. For each nonselected object O_h, there are four difference cases under consideration (they are marked out in the following diagram). Given a set of medoids, one of them is O_i, and the cluster related to it is C_i.



- **Case 1**: The first difference case is $O_j \in C_i$, given that $O_{j,2}$ is the representative of the second medoid that is closer or more similar to O_j:

$$d\left(O_j, O_h\right) \geq d\left(O_j, O_{j,2}\right)$$

After swapping, O_j will be relocated to $c_{j,2}$, and the cost of swapping will be defined as follows:

$$C_{jih} = d\left(O_j, O_{j,2}\right) - d\left(O_j, O_i\right)$$

- **Case 2**: The second difference case is $O_j \in C_i$, $d\left(O_j, O_h\right) < d\left(O_j, O_{j,2}\right)$. After swapping, O_j will be relocated to C_h, and the cost of swapping is defined as follows:

$$C_{jih} = d\left(O_j, O_h\right) - d\left(O_j, O_i\right)$$

- **Case 3**: Here, $O_j \in C_t, t \neq i, O_t \, represents \, C_t$ and $d\left(O_j, O_h\right) \geq d\left(O_j, O_t\right)$. After swapping, the cost of swapping is defined as follows:

$$C_{jih} = 0$$

- **Case 4**: Here, $O_j \in C_t, t \neq i, O_t \, represents \, C_t$ and $d\left(O_j, O_h\right) < d\left(O_j, O_t\right)$. After swapping, O_j will be relocated to C_h, and the cost of swapping is defined as follows:

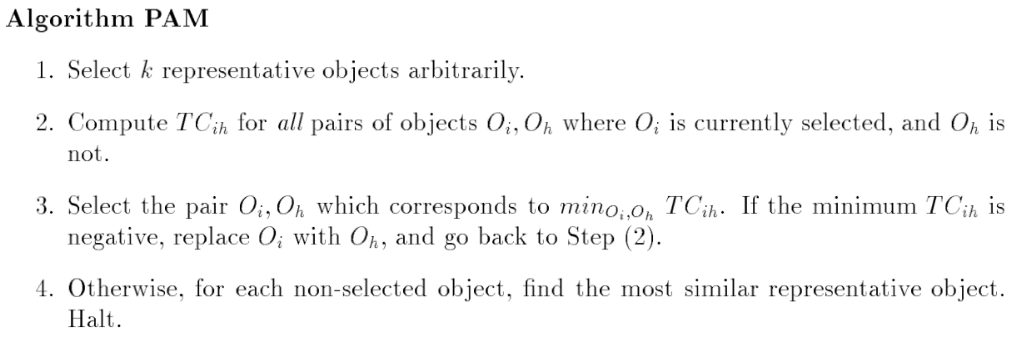
$$C_{jih} = d\left(O_j, O_h\right) - d\left(O_j, O_t\right)$$

At the end of each swapping step, the total cost of swapping is defined as follows:

$$\sum_j C_{jih}$$

The PAM algorithm

The **PAM** (**Partitioning Around Medoids**) algorithm is a partitional clustering algorithm. The summarized pseudocode for the PAM algorithm is as follows:



```
Algorithm PAM

1. Select $k$ representative objects arbitrarily.
2. Compute $TC_{ih}$ for *all* pairs of objects $O_i, O_h$ where $O_i$ is currently selected, and $O_h$ is
   not.
3. Select the pair $O_i, O_h$ which corresponds to $min_{O_i,O_h}\ TC_{ih}$. If the minimum $TC_{ih}$ is
   negative, replace $O_i$ with $O_h$, and go back to Step (2).
4. Otherwise, for each non-selected object, find the most similar representative object.
   Halt.
```

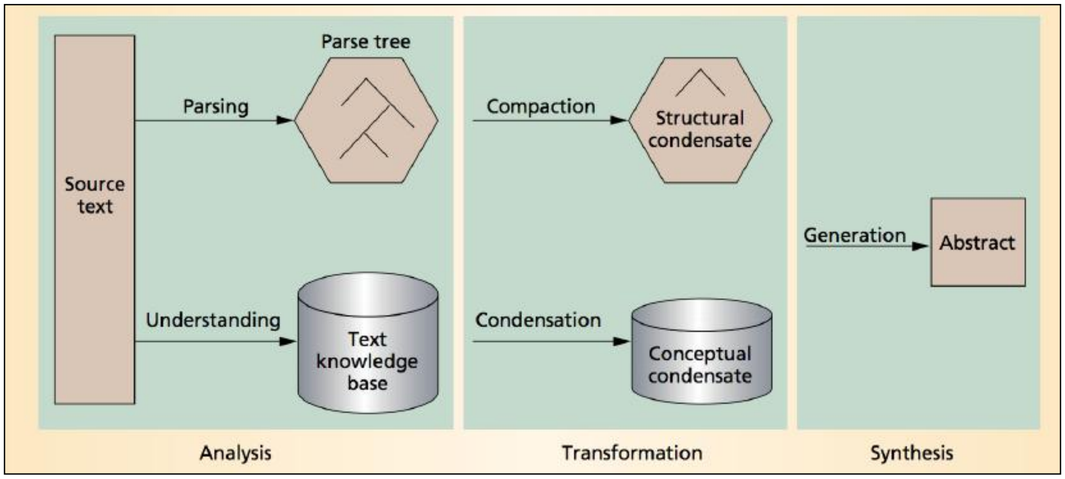
The R implementation

Take a look at the `ch_05_kmedoids.R` R code file from the bundle of R code for the previous algorithms. The code can be tested with the following command:

```
> source("ch_05_kmedoids.R")
```

Automatic abstraction and summarization of document text

Along with the increase in the size and quantity of documents on the Internet, the efficient algorithms are always in urgent need to get usable summarization or distill the key information. The documents will in versatile formats, structured or unstructured. The tasks include summarization of a single document or multiple documents. More extended target to extract summarization from multimedia files. Other challenges include summarizing multilingual documents. Abstraction requires the tool support from KNLP for grammar and lexicons for analyses and generation. One possible process for the abstraction is illustrated as follows:



Many approaches are suggested for automatic abstraction of document text, such as automatic extraction, automatic abstraction based on understanding, information extraction, and automatic abstraction based on structures. Possible features to be adapted to design the summarization system include the sentence length cutoff, fixed-phrase, paragraph, thematic word, and uppercase word features.

Abstraction or summarization popularly has been defined as a process with two steps. An intermediate representation of some sort is retrieved by the extraction of important concepts from the source texts. Given the intermediate representation, the summary is generated.

For the first step of the summarization process, it can largely be treated as part of automatic indexing. Lexical chains to extract important concepts from a document are one possible solution. Lexical chains exploit the cohesion among an arbitrary number of related words, and they are calculated by grouping (chaining) sets of words that are semantically related.

The CLARA algorithm

Instead of taking the whole set of data into consideration, the **CLARA** (**Clustering LARge Application**) algorithm randomly chooses a small portion of the actual data as a representative of the data. Medoids are then chosen from this sample using PAM. If the sample is selected in a fairly random manner, it should closely represent the original dataset.

CLARA draws multiple samples of the dataset, applies PAM to each sample, finds the medoids, and then returns its best clustering as the output. At first, a sample dataset D' is drawn from the original dataset D, and the PAM algorithm is applied to D' to find the k medoids. Use these k medoids and the dataset D to calculate the current dissimilarity. If it is smaller than the one you get in the previous iteration, then these k medoids are kept as the best k medoids. The whole process is performed a specified number of times.

The CLARA algorithm

The summarized pseudocode for the CLARA algorithm is as follows:

```
CLARA($X, d, k$)
    $bestDissim \leftarrow \infty$
    for $t \leftarrow 1$ to $S$
    do $X' \leftarrow$ RANDOM-SUBSET($X, s$)
       $D \leftarrow$ BUILD-DISSIM-MATRIX($X', d$)
       $(C', M) \leftarrow$ PAM($X', D, k$)
       $C \leftarrow$ ASSIGN-MEDOIDS($X, M, D$)
       $dissim \leftarrow$ TOTAL-DISSIM($C, M, D$)
       if $dissim < bestDissim$
          then $bestDissim \leftarrow dissim$
               $C_{best} \leftarrow C$
               $M_{best} \leftarrow M$
    return $(C_{best}, M_{best})$
```

The R implementation

Take a look at the `ch_05_clara.R` R code file from the bundle of R code for the previously mentioned algorithms. The codes can be tested with the following command:

```
> source("ch_05_clara.R")
```

CLARANS

CLARANS (**Clustering Large Applications based on RANdomized Search**) is efficient and effective and is the best practice for spatial data mining. CLARANS applies a strategy to search in a certain graph. A node in this graph, denoting it as $G_{n,k}$, is represented by a set of objects, $\left\{O_{m_1},\ldots,O_{m_k}\right\}$, $O_{m_1},\ldots,O_{m_k} \in D$. Here, *k* is the predefined value to choose the k medoids; as a result, the nodes in the graph are a set of $\left\{\left\{O_{m_1},\ldots,O_{m_k}\right\} \mid O_{m_1},\ldots,O_{m_k} \in D\right\}$. If two nodes, $S_1 = \left\{O_{m_1},\ldots,O_{m_k}\right\}$, and $S_2 = \left\{O_{w_1},\ldots,O_{w_k}\right\}$ are neighbors, then $\left|S_1 \cap S_2\right| = k-1$. Each node in the $G_{n,k}$ graph represents a set of medoids and the cluster related to it. As a result, a cost is related to each node; this cost is the total distance between any objects and the medoid represents its cluster. The cost differential of two neighbors can be calculated with the cost measure function introduced in the PAM algorithm.

The CLARANS algorithm

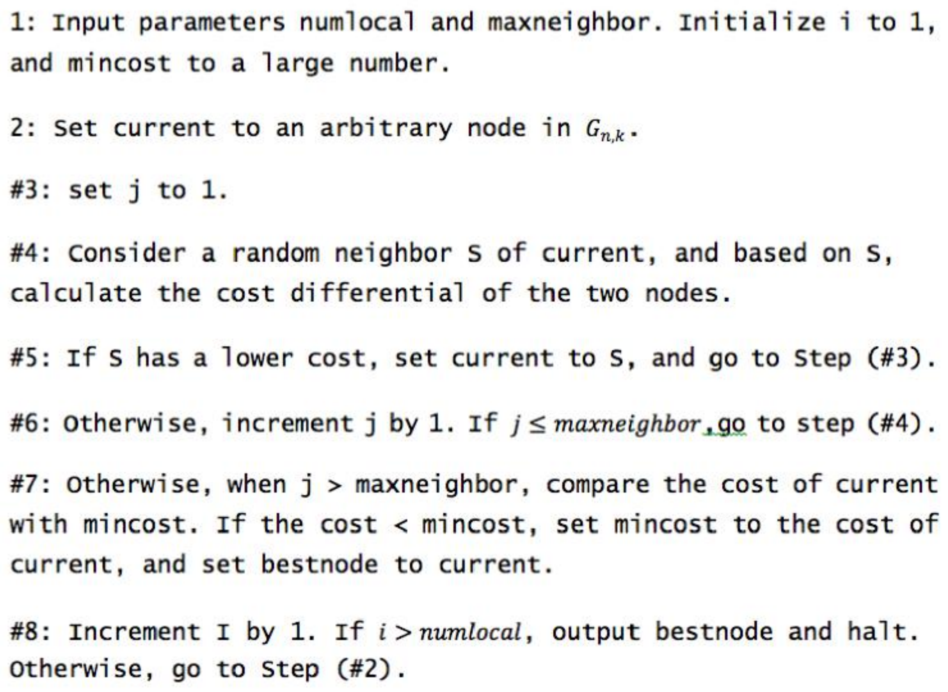
The input parameters for the CLARANS algorithm are as follows:

- `D`, which is the training tuples dataset
- `numlocal`
- `maxneighbor`

The output of the algorithm is:

- `bestnode`

The summarized pseudocode for the CLARANS algorithm is as follows:



```
1: Input parameters numlocal and maxneighbor. Initialize i to 1, and mincost to a large number.
2: Set current to an arbitrary node in $G_{n,k}$.
#3: set j to 1.
#4: Consider a random neighbor S of current, and based on S, calculate the cost differential of the two nodes.
#5: If S has a lower cost, set current to S, and go to Step (#3).
#6: Otherwise, increment j by 1. If $j \le maxneighbor$, go to step (#4).
#7: Otherwise, when j > maxneighbor, compare the cost of current with mincost. If the cost < mincost, set mincost to the cost of current, and set bestnode to current.
#8: Increment I by 1. If $i > numlocal$, output bestnode and halt. Otherwise, go to Step (#2).
```

The R implementation

Take a look at the `ch_05_clarans.R` R code file from the bundle of R code for the previously mentioned algorithms. The codes can be tested with the following command:

```
> source("ch_05_clarans.R")
```

Unsupervised image categorization and affinity propagation clustering

AP (**Affinity Propagation**) finds a set of exemplars, $X_e = \{x_{e_1}, \ldots, x_{e_k}\}$, in the dataset and assigns nonselected points to the exemplars. An exemplar is the representative of a cluster.

Two types of messages are exchanged between data objects or points; they are explained here:

- The *r(i,k)* message, which is called **responsibility**, represents the accumulated evidence sent from x_i to x_k. It informs us that x_k is suitable to serve as the exemplar of point x_i. Every candidate is counted in.
- The *a(i,k)* message, which is called **availability**, denotes the accumulated evidence sent from x_i to x_k. It informs us that x_k should be the exemplar. Every support from other points is well considered.

Both *r(i, k)* and *a(i,k)* are initialized as 0 at the beginning of the algorithm:

$$r(i,k) = s(i,k) - \max_{k',k' \neq k}\{a(i,k') + s(i,k')\}$$

$$r(k,k) = s(k,k) - \max_{k',k' \neq k}\{s(k,k')\}$$

$s(i,k) = -\|x_i - x_k\|^2$, $i \neq k$, for `s(k, k)` is initialized with the same value (typically defined with heuristic knowledge) for each point at the start and updated in the following description to recur the affection. `s(i, k)` denotes the extent to which x_j is suited to be the exemplar of x_i. Here is a possible value for `s(i, i)` to be set as a constant:

$$s(l,l) = \frac{\sum_{i,j=1;i \neq j}^{N} s(i,j)}{N*(N-1)}, 1 \leq l \leq N$$

$$a(i,k) = \min\{0, r(k,k) + \sum_{i',i' \neq i,k} \max\{0, r(i',k)\}\}$$

$$a(k,k) = \sum_{i',i' \neq i} \max\{0, r(i',k)\}$$

The index of exemplar, $e(x_i)$, which is for data point x_i, is defined with the following formula:

```
arg max {a(i,k) + r(i,k), k = 1,…, N}
```

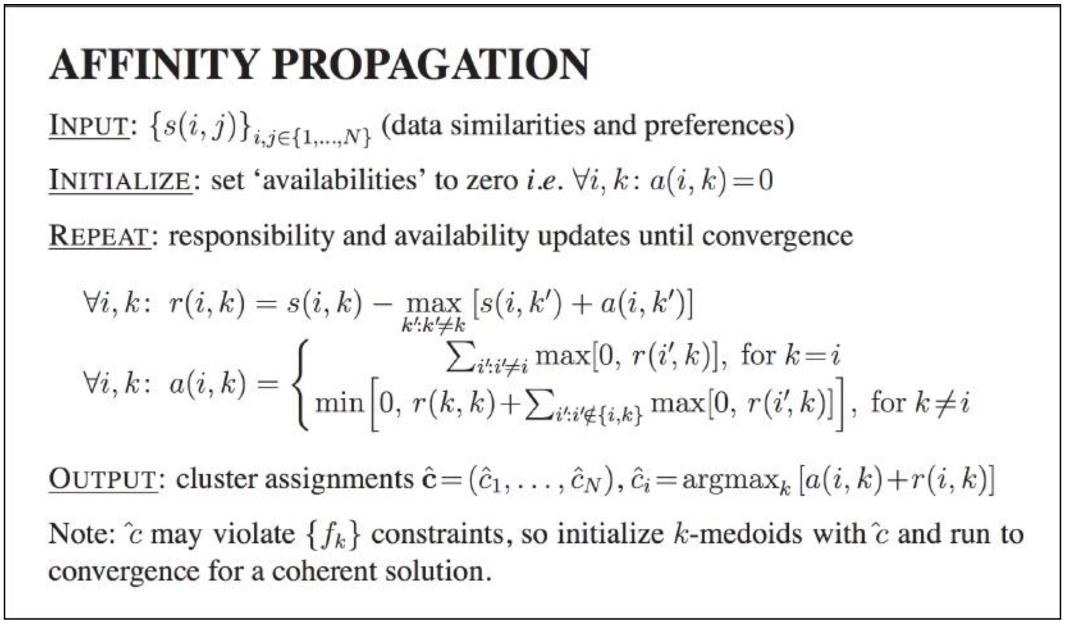
Given `R = (r(i, j))` as the responsibility matrix and `A = (a(i, j))` as the availability matrix, *t* represents the iteration counts, where a damping factor $\lambda \in [0,1]$ is set to depress numerical oscillations that might arise:

$$R_{t+1} = (1-\lambda)R_t + \lambda R_{t-1}$$

$$A_{t+1} = (1-\lambda)A_t + \lambda A_{t-1}$$

Affinity propagation clustering

Affinity propagation itself is summarized as follows:



AFFINITY PROPAGATION

INPUT: $\{s(i,j)\}_{i,j\in\{1,\ldots,N\}}$ (data similarities and preferences)

INITIALIZE: set 'availabilities' to zero *i.e.* $\forall i,k\colon a(i,k)=0$

REPEAT: responsibility and availability updates until convergence

$$\forall i,k\colon\ r(i,k) = s(i,k) - \max_{k':k'\neq k}\left[s(i,k') + a(i,k')\right]$$

$$\forall i,k\colon\ a(i,k) = \begin{cases} \sum_{i':i'\neq i}\max[0,\, r(i',k)], & \text{for } k=i \\ \min\left[0,\, r(k,k)+\sum_{i':i'\notin\{i,k\}}\max[0,\, r(i',k)]\right], & \text{for } k\neq i \end{cases}$$

OUTPUT: cluster assignments $\hat{\mathbf{c}}=(\hat{c}_1,\ldots,\hat{c}_N)$, $\hat{c}_i=\text{argmax}_k\left[a(i,k)+r(i,k)\right]$

Note: $\hat{c}$ may violate $\{f_k\}$ constraints, so initialize k-medoids with $\hat{c}$ and run to convergence for a coherent solution.

The R implementation

Take a look at the `ch_05_affinity_clustering.R` R code file from the bundle of R code for the previously mentioned algorithms. The codes can be tested with the following command:

```
> source("ch_05_affinity_clustering.R")
```

Unsupervised image categorization

Due to the massive number of images and other multimedia documents, the task to classify images becomes even harder than before. Unsupervised image categorization is frequently utilized by image and video summarization, or it just serves as a preprocessing step in supervised methods for classification.

One major issue related to unsupervised image categorization is to estimate the distribution of image categories. Further on, finding the most descriptive prototypes of the image categories is another main issue of image categorization.

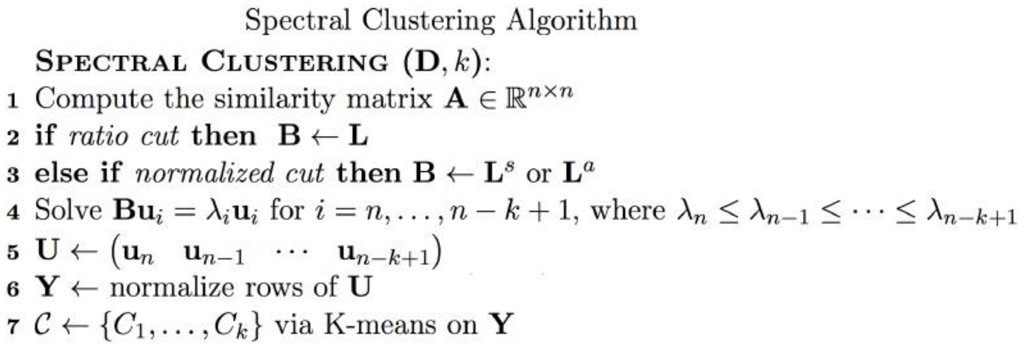
Each image can be represented as a high-dimensional data instance, including features related to color, texture, and shape. The exemplar technique is applied here; it represents image categories by a small set of image or its fragments. Given exemplar concepts, the dimension of an image data instance reduces to a relatively small size and eases further processing. The measures applied here include the Chamfer, Hausdorff, and shuffle distances.

Natural categories of the dataset can be of various complex types; overlapping might be a frequent shape.

Unsupervised image categorization or classification is a clustering problem. Image clustering is to identify a set of similar image primitives, such as pixels, line elements, regions, and so on. Given the complex dataset, the recommended way is to use the prototype-based clustering algorithm. Affinity propagation algorithms can be applied to unsupervised categorization by finding a subset of representative exemplars.

The spectral clustering algorithm

The summarized pseudocode for the spectral clustering algorithm is as follows:



```
Spectral Clustering Algorithm
SPECTRAL CLUSTERING ($\mathbf{D}, k$):
  Compute the similarity matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$
  if ratio cut then $\mathbf{B} \leftarrow \mathbf{L}$
  else if normalized cut then $\mathbf{B} \leftarrow \mathbf{L}^s$ or $\mathbf{L}^a$
  Solve $\mathbf{B}\mathbf{u}_i = \lambda_i \mathbf{u}_i$ for $i = n, \dots, n-k+1$, where $\lambda_n \leq \lambda_{n-1} \leq \cdots \leq \lambda_{n-k+1}$
  $\mathbf{U} \leftarrow \begin{pmatrix} \mathbf{u}_n & \mathbf{u}_{n-1} & \cdots & \mathbf{u}_{n-k+1} \end{pmatrix}$
  $\mathbf{Y} \leftarrow$ normalize rows of $\mathbf{U}$
  $\mathcal{C} \leftarrow \{C_1, \dots, C_k\}$ via K-means on $\mathbf{Y}$
```

The R implementation

Take a look at the `ch_05_ spectral_clustering.R` R code file from the bundle of R code for the previously mentioned algorithms. The codes can be tested with the following command:

```
> source("ch_05_ spectral_clustering.R")
```

News categorization and hierarchical clustering

Hierarchical clustering divides the target dataset into multilevels or a hierarchy of clusters. It segments data points along with successive partitions.

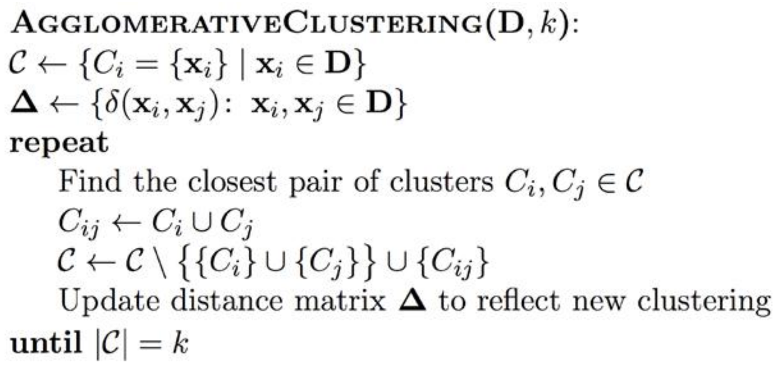
There are two strategies for hierarchical clustering. **Agglomerative clustering** starts with each data object in the input dataset as a cluster, and then in the following steps, clusters are merged according to certain similarity measures that end in only one cluster. **Divisive clustering**, in contrast, starts with one cluster with all the data objects in the input dataset as members, and then, the cluster splits according to certain similarity measures in the following steps, and at the end, singleton clusters of individual data objects are left.

The characteristics of hierarchical clustering methods are as follows:

- Multilevel decomposition.
- The merges or splits cannot perform a rollback. The error in the algorithm that is introduced by merging or splitting cannot be corrected.
- The hybrid algorithm.

Agglomerative hierarchical clustering

The summarized pseudocode for the agglomerative hierarchical clustering algorithm is as follows:



```
AGGLOMERATIVECLUSTERING($\mathbf{D}, k$):
$\mathcal{C} \leftarrow \{C_i = \{\mathbf{x}_i\} \mid \mathbf{x}_i \in \mathbf{D}\}$
$\mathbf{\Delta} \leftarrow \{\delta(\mathbf{x}_i, \mathbf{x}_j) \colon \mathbf{x}_i, \mathbf{x}_j \in \mathbf{D}\}$
repeat
    Find the closest pair of clusters $C_i, C_j \in \mathcal{C}$
    $C_{ij} \leftarrow C_i \cup C_j$
    $\mathcal{C} \leftarrow \mathcal{C} \setminus \{\{C_i\} \cup \{C_j\}\} \cup \{C_{ij}\}$
    Update distance matrix $\mathbf{\Delta}$ to reflect new clustering
until $|\mathcal{C}| = k$
```

The BIRCH algorithm

The **BIRCH** (**Balanced Iterative Reducing and Clustering using Hierarchies**) algorithm is designed for large dynamic datasets and can also be applied to incremental and dynamic clustering. Only one pass of dataset is required, and this means that there is no need to read the whole dataset in advance.

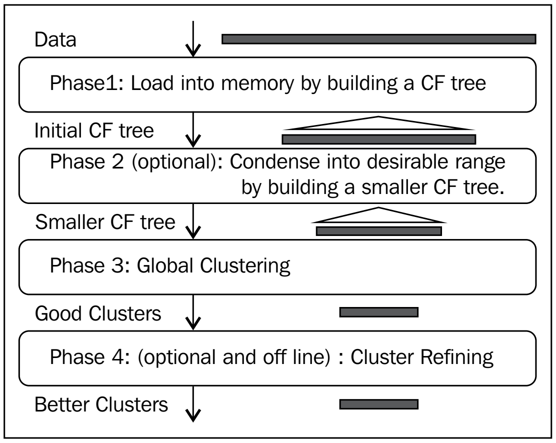
CF-Tree is a helper data structure that is used to store the summary of clusters, and it is also a representative of a cluster. Each node in the CF-Tree is declared as a list of entries, $[CF_i, child_i], i \in [1, B]$, and B is the predefined entry limitation. $child_i$ denotes the link to the *i*th child.

Given CF_i as the representative of cluster *i*, is defined as $CF_i = (N_i, LS, SS)$, where N_i is the member count of the cluster, *LS* is the linear sum of the member objects, and *SS* is the squared sum of the member objects.

The leave of the CF-Tree must conform to the diameter limitation. The diameter of the $\left(\sum_{l=1}^{N_1}\sum_{m=1}^{N_1}(x_l - x_m)^2 / N_i(N_i - 1)\right)^{1/2}$ cluster must be less than the predefined threshold value, *T*.

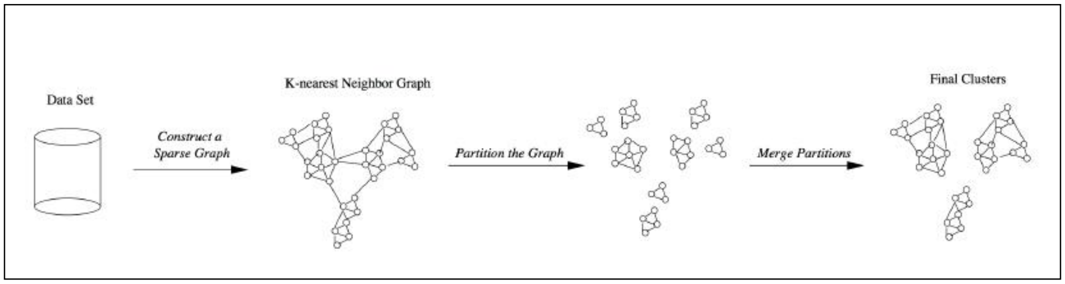
The main helper algorithms in BIRCH are **CF-Tree insertion** and **CF-Tree rebuilding**.

The summarized pseudocode for the BIRCH algorithm is as follows:


Data
Phase1: Load into memory by building a CF tree
Initial CF tree
Phase 2 (optional): Condense into desirable range by building a smaller CF tree.
Smaller CF tree
Phase 3: Global Clustering
Good Clusters
Phase 4: (optional and off line) : Cluster Refining
Better Clusters

The chameleon algorithm

The overall framework for the chameleon algorithm is illustrated in the following diagram:



There are three main steps in the chameleon algorithm, which are as follows:

- Sparsification: This step is to generate a k-Nearest Neighbor graph, which is derived from a proximity graph.
- Graph partitioning: This step applies a multilevel graph-partitioning algorithm to partition the dataset. The initial graph is an all-inclusive graph or cluster. This then bisects the largest graph in each successive step, and finally, we get a group of roughly, equally-sized clusters (with high intrasimilarity). Each resulting cluster has a size not more than a predefined size, that is, `MIN_SIZE`.

- Agglomerative hierarchical clustering: This last step is where the clusters are merged based on self-similarity. The notion of self-similarity is defined with the RI and RC; they can be combined in various ways to be a measure of self-similarity.

Relative Closeness (**RC**), given the C_i and C_j clusters with sizes m_i and m_j, respectively. $\overline{S}_{EC}\left(C_i, C_j\right)$ denotes the average weight of the edges that connect the two clusters. $EC\left(C_i\right)\left(EC\left(C_j\right)\right)$ denotes the average weight of the edges that bisect the $C_i\left(C_j\right)$ cluster:

$$RC(C,C) = \overline{S}_{EC}\left(C_i, C_j\right) / \left(\frac{m_i}{m_i + m_j} S_{EC}\left(C_i\right) + \frac{m_i}{m_i + m_j} S_{EC}\left(C_j\right)\right)$$

Relative Interconnectivity (**RI**) is defined in the following equation. Here, $EC\left(C_i, C_j\right)$ is the sum of the edges to connect the two clusters, and $EC\left(C_i\right)\left(EC\left(C_j\right)\right)$ is the minimum sum of the cut edges that bisect the $C_i\left(C_j\right)$ cluster:

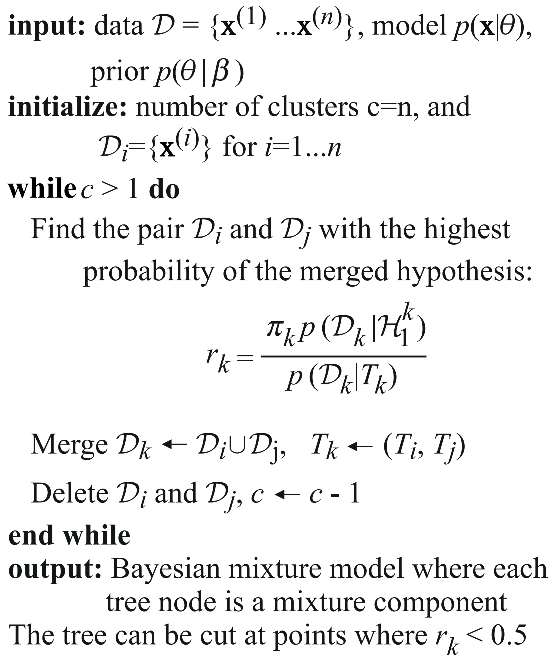
$$RI\left(C_i, C_j\right) = EC(C_{i,} C_j) / \frac{1}{2}\left(EC(C_i) + EC(C_j)\right)$$

The summarized pseudocode for the chameleon algorithm is as follows:

```
1: Build a k-nearest neighbor graph.

2: Partition the graph using a multilevel graph-partitioning
algorithm.

3: do {

4:   Merge the clusters that best preserve the cluster
self-similarity with respect to relative interconnectivity and
relative closeness.

5:} while No more clusters can be merged.
```

The Bayesian hierarchical clustering algorithm

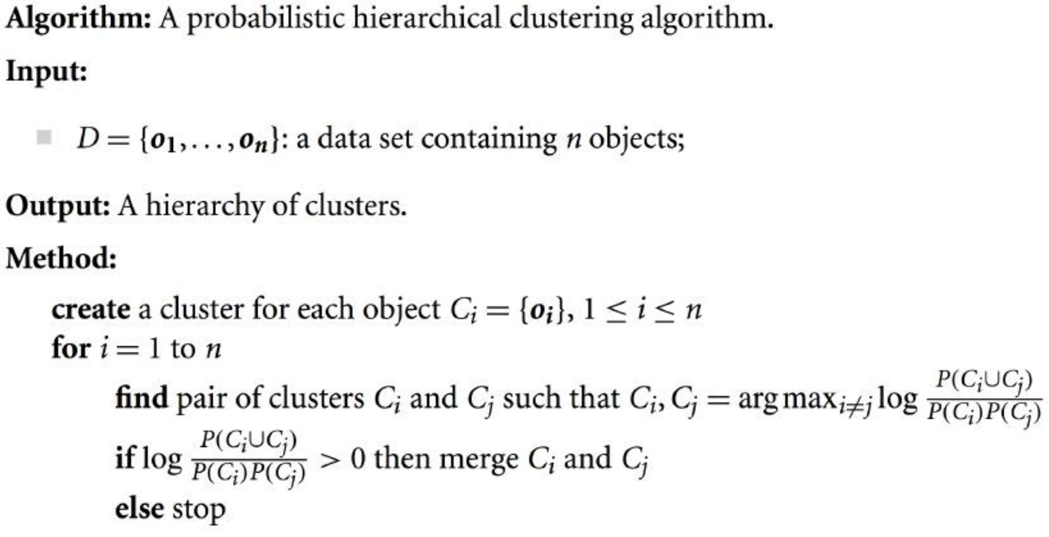
The summarized pseudocode for the Bayesian hierarchical clustering algorithm is as follows:



```
input: data $\mathcal{D} = \{\mathbf{x}^{(1)} \ldots \mathbf{x}^{(n)}\}$, model $p(\mathbf{x}|\theta)$,
        prior $p(\theta \mid \beta)$
initialize: number of clusters c=n, and
        $\mathcal{D}_i=\{\mathbf{x}^{(i)}\}$ for $i$=1...$n$
while $c > 1$ do
  Find the pair $\mathcal{D}_i$ and $\mathcal{D}_j$ with the highest
       probability of the merged hypothesis:
            $r_k = \frac{\pi_k p(\mathcal{D}_k|\mathcal{H}_1^k)}{p(\mathcal{D}_k|T_k)}$
  Merge $\mathcal{D}_k \leftarrow \mathcal{D}_i \cup \mathcal{D}_j$,   $T_k \leftarrow (T_i, T_j)$
  Delete $\mathcal{D}_i$ and $\mathcal{D}_j$, $c \leftarrow c - 1$
end while
output: Bayesian mixture model where each
        tree node is a mixture component
The tree can be cut at points where $r_k < 0.5$
```

The probabilistic hierarchical clustering algorithm

The probabilistic hierarchical clustering algorithm is a hybrid of hierarchical clustering and probabilistic clustering algorithms. As a probabilistic clustering algorithm, it applies a completely probabilistic approach, as depicted here:



Algorithm: A probabilistic hierarchical clustering algorithm.

Input:

- $D = \{\boldsymbol{o}_1, \ldots, \boldsymbol{o}_n\}$: a data set containing n objects;

Output: A hierarchy of clusters.

Method:

```
create a cluster for each object $C_i = \{\boldsymbol{o}_i\}, 1 \le i \le n$
for $i = 1$ to $n$
    find pair of clusters $C_i$ and $C_j$ such that $C_i, C_j = \arg\max_{i \neq j} \log \frac{P(C_i \cup C_j)}{P(C_i)P(C_j)}$
    if $\log \frac{P(C_i \cup C_j)}{P(C_i)P(C_j)} > 0$ then merge $C_i$ and $C_j$
    else stop
```

The R implementation

Take a look at the `ch_05_ahclustering.R` R code file from the bundle of R code for the previously mentioned algorithms. The codes can be tested with the following command:

```
> source("ch_05_ahclustering.R")
```

News categorization

News portals provide visitors with tremendous news categorized with a predefined set of topics. With the exponential growth of information we receive from the Internet, clustering technologies are widely used for web documents categorization, which include online news. These are often news streams or news feeds.

One advantage of a clustering algorithm for this task is that there is no need for prior domain knowledge. News can be summarized by clustering news that covers specific events. Unsupervised clustering can also play a pivotal role in discovering the intrinsic topical structure of the existing text collections, while new categorization is used to classify the documents according to a predefined set of classes.

Services such as Google News are provided. For a story published by a couple of new agencies or even the same agency, there are often various versions that exist at the same time or span the same range of days. Here, clustering can help aggregate the news related to the same story, and the result is a better overview of the current story for visitors.

As preprocessing steps, plain texts are extracted from the web pages or news pages. Reuters-22173 and -21578 are two popular documents dataset used for research. The representation of a data instance in the dataset can either be classical term-document vectors or term-sentence vectors, as the dataset is a collection of documents, and the cosine-like measures are applied here.

Time for action

Here are some practice questions for you to check whether you understood the concepts we covered in this chapter:

1. What is the PAM clustering algorithm?
2. What is the k-means algorithm?
3. What is the k-medoids algorithm?
4. What is the CLARA algorithm?
5. What is the CLARANS algorithm?

Summary

In this chapter, we looked at:

- Partition-based clustering.
- The k-means algorithm is a partition-based clustering algorithm. The centroids of clusters are defined as a representative of each cluster. In k-means clustering, a set of *n* data points in a D-dimensional space and an integer *k* are given. The problem is to distribute a set of *k* points in the centers to minimize the SSE.
- The k-medoids algorithm is a partition-based clustering algorithm. The representatives of each resulting clusters are chosen from the dataset itself, that is, the data objects belong to it.
- CLARA depends on sampling. It draws a sample from the original dataset instead of the entire dataset. PAM is then applied to each sampling. Then, the best result is kept during all the iterations.
- CLARANS is a clustering algorithm based on randomized search.

- The affinity propagation clustering algorithm recursively passes affinity messages between objects or points and converges to exemplars adaptively.
- Spectral clustering is used to construct graph partitions based on eigenvectors of the adjacency matrix.
- Hierarchical clustering decomposes a dataset *D* into levels of nested clusters; this is represented by a **dendrogram**, a tree that iteratively splits the dataset *D* into smaller subsets. The process stops only after each subset consists of only one object.

The next chapter will cover more advanced topics related to clustering algorithms, density-based algorithms, grid-based algorithms, the EM algorithm, high-dimensional algorithms, constraint-based clustering algorithms, and so on.

6
Advanced Cluster Analysis

In this chapter, you will learn how to implement the top algorithms for clusters with R. The evaluation/benchmark/measure tools are also provided.

In this chapter, we will cover the following topics:

- Customer categorization analysis of e-commerce and DBSCAN
- Clustering web pages and OPTICS
- Visitor analysis in the browser cache and DENCLUE
- Recommendation system and STING
- Web sentiment analysis and CLIQUE
- Opinion mining and WAVE CLUSTER
- User search intent and the EM algorithm
- Customer purchase data analysis and clustering high-dimensional data
- SNS and clustering graph and network data

Customer categorization analysis of e-commerce and DBSCAN

By defining the density and density measures of data point space, the clusters can be modeled as sections with certain density in the data space.

Density Based Spatial Clustering of Applications with Noise (**DBSCAN**) is one of the most popular density-based clustering algorithms. The major characteristics of DBSCAN are:

- Good at dealing with large datasets with noises
- Clusters of various shapes can be dealt with

DBSCAN is based on the classification of the data points in the dataset as core data points, border data points, and noise data points, with the support of the use of density relations between points, including directly density-reachable, density-reachable, density-connected points. Before we provide a detailed description of DBSCAN, let's illustrate these ideas.

A point is defined as a core point if the number of data points within the distance of the predefined parameter, **Eps** or $\in$, is greater than that of the predefined parameter, **MinPts**. The space within the Eps is called Eps-neighborhood or $N_{\in}(q)$. An object, *o*, is noise only if there is no cluster that contains *o*. A border data object is any object that belongs to a cluster, but not a core data object.

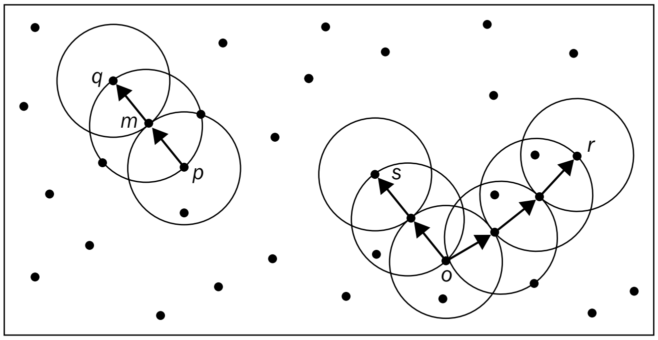
Given a core object, *p*, and an object, *q*, the object, *q*, is **directly density-reachable** from *p* if $q \in N_{\in}(p)$.

Given a core object, *p*, and an object, *q*, the object *q* is **density-reachable** from *p* if $\exists\, a\, chain\, of\, data\, object\ p_1,..,p_n, p_1 = q, p_n = p, \forall 1 \le i < n$ and $p_i\ is\ directly\ density - reachable\ from\ p$.

For two data objects, q_1 and q_2, they are **density-connected** if $\exists\, a\, core\, object\ p, q_1\ and\ q_2\ are\ density - reachable\ to\ p$.

The **density-based cluster** denotes a set of density-connected data objects that are maximal according to density reachability.

Here is an example illustrated in the following diagram:

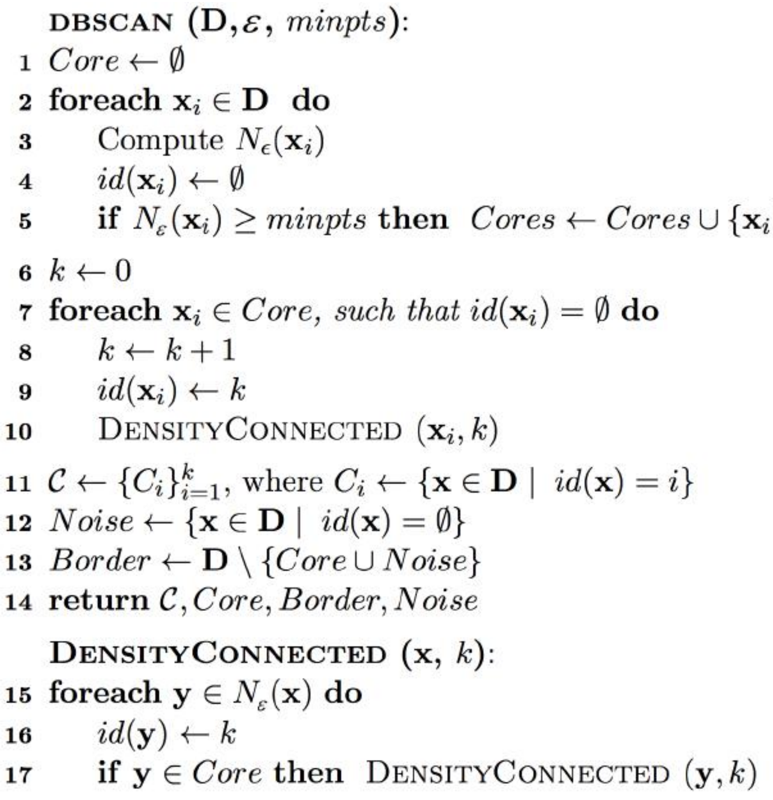


The DBSCAN algorithm

The summarized pseudocode for the DBSCAN algorithm is as follows, with the following input/output parameters defined.

The input parameters for the DBSCAN algorithm are as follows:

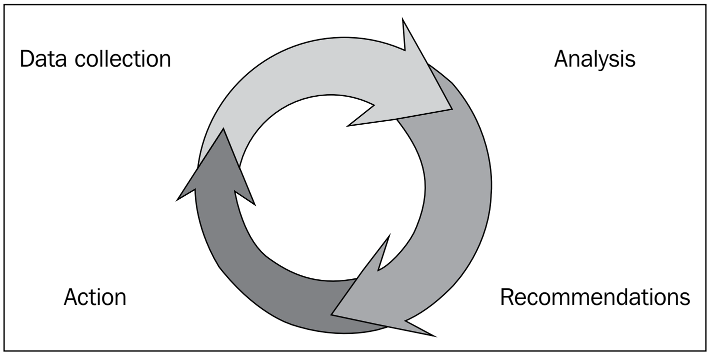
- *D*, which is the training tuples dataset
- *k*, which is the neighbor list size
- *Eps*, which is the radius parameter that denotes the neighborhood area of a data point
- *MinPts*, which is the minimum number (the neighborhood density threshold) of points that must exist in the Eps-neighborhood
- The output of the algorithm
- A set of density-based clusters



```
DBSCAN ($\mathbf{D}, \varepsilon$, $minpts$):
$Core \leftarrow \emptyset$
foreach $\mathbf{x}_i \in \mathbf{D}$ do
    Compute $N_\epsilon(\mathbf{x}_i)$
    $id(\mathbf{x}_i) \leftarrow \emptyset$
    if $N_\varepsilon(\mathbf{x}_i) \geq minpts$ then $Cores \leftarrow Cores \cup \{\mathbf{x}_i\}$
$k \leftarrow 0$
foreach $\mathbf{x}_i \in Core$, such that $id(\mathbf{x}_i) = \emptyset$ do
    $k \leftarrow k + 1$
    $id(\mathbf{x}_i) \leftarrow k$
    DensityConnected $(\mathbf{x}_i, k)$
$\mathcal{C} \leftarrow \{C_i\}_{i=1}^{k}$, where $C_i \leftarrow \{\mathbf{x} \in \mathbf{D} \mid id(\mathbf{x}) = i\}$
$Noise \leftarrow \{\mathbf{x} \in \mathbf{D} \mid id(\mathbf{x}) = \emptyset\}$
$Border \leftarrow \mathbf{D} \setminus \{Core \cup Noise\}$
return $\mathcal{C}, Core, Border, Noise$

DensityConnected ($\mathbf{x}$, $k$):
foreach $\mathbf{y} \in N_\varepsilon(\mathbf{x})$ do
    $id(\mathbf{y}) \leftarrow k$
    if $\mathbf{y} \in Core$ then DensityConnected $(\mathbf{y}, k)$
```

Customer categorization analysis of e-commerce

Customers of e-commerce can be categorized by the psychographic, culturally specific purchasing behavior. The result of the customer categorization can make the storeowner efficiently and effectively respond to the customer. The e-commerce general analysis process is illustrated as follows:

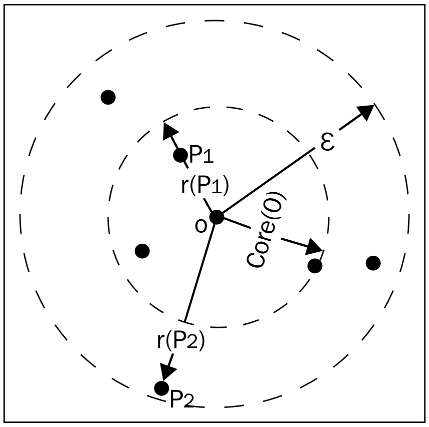


Clustering web pages and OPTICS

Ordering points to identify the clustering structure, OPTICS, extends the DBSCAN algorithm and is based on the phenomenon that density-based clusters, with respect to a higher density, are completely contained in density-connected sets with respect to lower density.

To construct density-based clusters with different densities simultaneously, the objects are dealt with in a specific order when expanding a cluster, that is, according to the order, an object that is density-reachable with respect to the lowest $\in$ for the higher density clusters will be finished first.

Two major concepts are introduced to illustrate the OPTICS algorithm: **core-distance** of an object, *p*, and **reachability-distance** of object, *p*. An example is illustrated in the following image. During which, core-distance (○), reachability-distances, $r(p_1, o)$, $r(p_2, o)$, for **MinPts=4**.



The **core-distance** of an object, `p`, is denoted as the minimal value, ∈′, considering that the ∈′-neighborhood at least contains `MinPts` data objects; otherwise, this is undefined.

Given two data objects, *p* and *q*, the **reachability-distance** of object, *p*, from *q* represents the smallest radius value that makes *p* density-reachable from *q* if *q* is a core data object; otherwise, this is undefined.

OPTICS outputs an ordering of all data objects in a given dataset in which the data objects are processed. For each object, the core-distance and a suitable reachability-distance for each data object are calculated and output together.

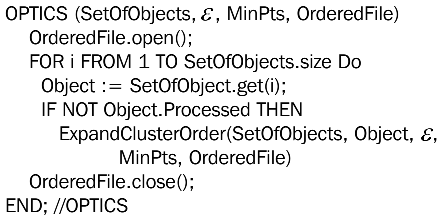
The OPTICS algorithm

OPTICS randomly selects an object from the input dataset as the current data object, *p*. In addition, for the object, *p*, the ∈-neighborhood is fetched, the core-distance is calculated, and the reachability-distance is undefined.

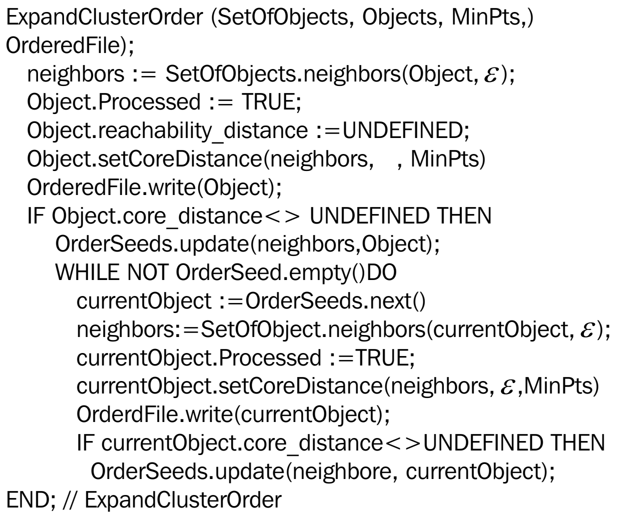
Given *p* as a core data object, OPTICS recalculates the reachability-distance for any object, *q* (in the neighborhood of *p*), from *p* and inserts *q* into OrderSeeds if *q* has not been processed.

Once the augmented cluster ordering of the input dataset is generated with respect to ∈, `MinPts` and a clustering-distance $\in' \leq \in$, the density-based clustering is performed with this order.

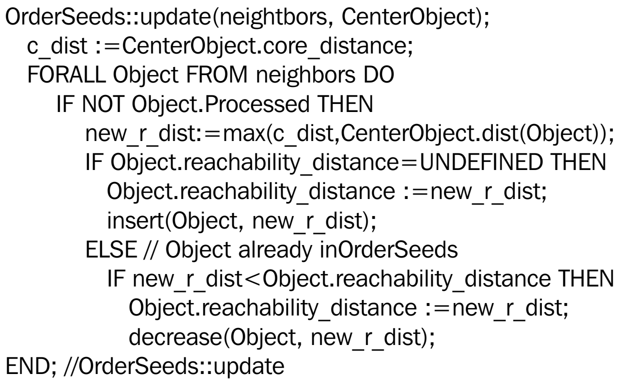
The summarized pseudocode for the OPTICS algorithm with a couple of supporter functions are as follows:



```
OPTICS (SetOfObjects, $\varepsilon$, MinPts, OrderedFile)
    OrderedFile.open();
    FOR i FROM 1 TO SetOfObjects.size Do
      Object := SetOfObject.get(i);
      IF NOT Object.Processed THEN
         ExpandClusterOrder(SetOfObjects, Object, $\varepsilon$,
                   MinPts, OrderedFile)
    OrderedFile.close();
END; //OPTICS
```



```
ExpandClusterOrder (SetOfObjects, Objects, MinPts,)
OrderedFile);
   neighbors := SetOfObjects.neighbors(Object, $\varepsilon$);
   Object.Processed := TRUE;
   Object.reachability_distance :=UNDEFINED;
   Object.setCoreDistance(neighbors,    , MinPts)
   OrderedFile.write(Object);
   IF Object.core_distance<> UNDEFINED THEN
       OrderSeeds.update(neighbors,Object);
       WHILE NOT OrderSeed.empty()DO
          currentObject :=OrderSeeds.next()
          neighbors:=SetOfObject.neighbors(currentObject, $\varepsilon$);
          currentObject.Processed :=TRUE;
          currentObject.setCoreDistance(neighbors, $\varepsilon$,MinPts)
          OrderdFile.write(currentObject);
          IF currentObject.core_distance<>UNDEFINED THEN
            OrderSeeds.update(neighbore, currentObject);
END; // ExpandClusterOrder
```



```
OrderSeeds::update(neightbors, CenterObject);
   c_dist :=CenterObject.core_distance;
   FORALL Object FROM neighbors DO
       IF NOT Object.Processed THEN
           new_r_dist:=max(c_dist,CenterObject.dist(Object));
           IF Object.reachability_distance=UNDEFINED THEN
              Object.reachability_distance :=new_r_dist;
              insert(Object, new_r_dist);
           ELSE // Object already inOrderSeeds
              IF new_r_dist<Object.reachability_distance THEN
                 Object.reachability_distance :=new_r_dist;
                 decrease(Object, new_r_dist);
END; //OrderSeeds::update
```

```
ExractDBSCAN-Clustering(ClusterOrderedObjs, $\varepsilon$', MinPts)
// Precondition: $\varepsilon$ '$\leq$ generating dist $\varepsilon$ for ClusterOrderedObjs
   ClusterId := NOISE;
   FOR i FROM 1 TO ClusterOrderedObjs.size Do
     Object :=ClusterOrderedObjs.get(i);
     IF Object.reachability_distance>$\varepsilon$' THEN
        // UNDEFINED >$\varepsilon$
        IF Object.core_distance $\leq$ $\varepsilon$' THEN
            ClusterId := nextId(ClusterId);
            Object.clusterId := ClusterId;
         ELSE
            Object.clusterId := NOISE;
       ELSE     // Object.reachability_distance $\leq$ $\varepsilon$'
            Object.clusterId := ClusterId;
 END; // ExtractDBSCAN-Clustering
```

The R implementation

Please take a look at the R codes file `ch_06_optics.R` from the bundle of R codes for the previously mentioned algorithm. The codes can be tested with the following command:

```
> source("ch_06_optics.R")
```

Clustering web pages

Clustering web pages can be used to group related texts/articles and serve as the preprocessing steps for supervised learning. It enables automated categorization.

Web pages are versatile and have a different structure (well-formed or badly formed without structure) and contents.

The Yahoo! industry web page data (CMUWeb KBProject, 1998) is used as the test data here.

Visitor analysis in the browser cache and DENCLUE

DENsity-based CLUstEring (**DENCLUE**) is a density-based clustering algorithm that depends on the support of density-distribution functions.

Before a detailed explanation on the DENCLUE algorithm, some concepts need to be introduced; they are **influence function**, **density function**, **gradient**, and **density attractor**.

The influence function of a specific data object can be any function for which the Gaussian kernel is usually used as the kernel at the data point.

The density function at a point, *x*, is defined as the sum of the influence functions of all the data objects at this data point.

A point is defined as a density attractor if it is a local maximum of the density function and is computed as

$$x^0 = x$$

$$x^{i+1} = x^i + \delta * \frac{\nabla f_B^D\left(x^i\right)}{\left\|\nabla f_B^D\left(x^i\right)\right\|}$$

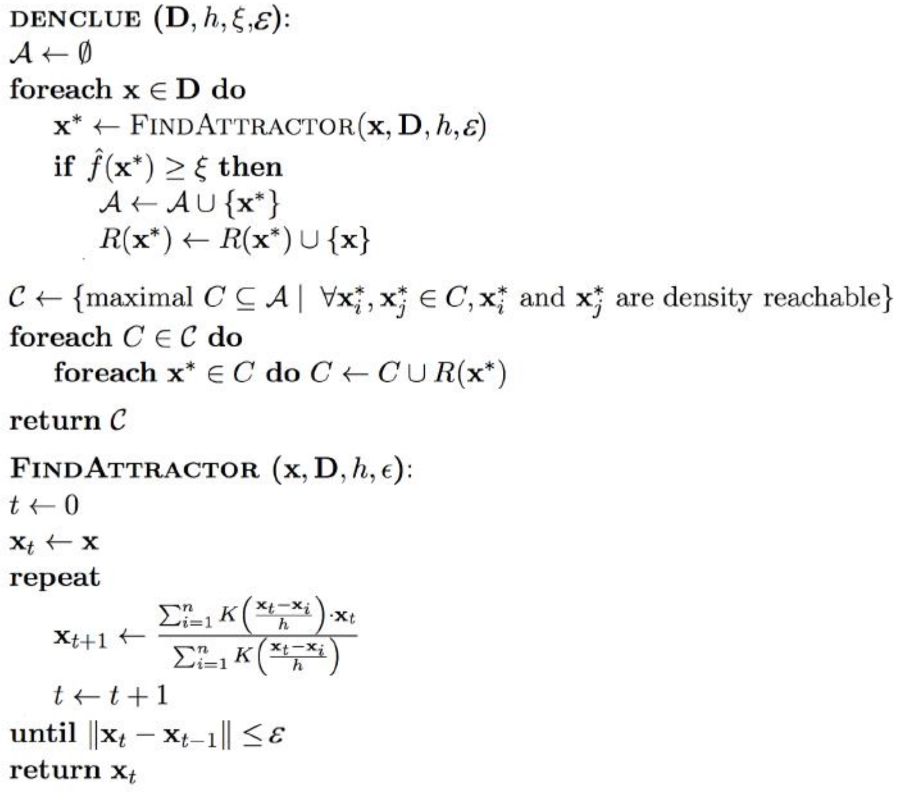
A gradient of the density function is defined in the following equation, given the density function, $f_B^D\left(x\right)$.

$$\nabla f_B^D\left(x\right) = \sum_{i=1}^{N}\left(x_i - x\right) * f_B^{x_i}\left(x\right)$$

DENCLUE defines a density function for the data point space at first. All the local maxima data points are searched and found. Assign each data point to the nearest local maxima point to maximize the density related to it. Each group of data points bound with a local maxima point is defined as a cluster. As a postprocess, the cluster is discarded if its bound local maxima density is lower than the user-predefined value. The clusters are merged if there exists a path such that each point on the path has a higher density value than the user-predefined value.

The DENCLUE algorithm

The summarized pseudocodes for the DENCLUE algorithm is as follows:



```
DENCLUE $(\mathbf{D}, h, \xi, \varepsilon)$:
$\mathcal{A} \leftarrow \emptyset$
foreach $\mathbf{x} \in \mathbf{D}$ do
    $\mathbf{x}^* \leftarrow$ FindAttractor$(\mathbf{x}, \mathbf{D}, h, \varepsilon)$
    if $\hat{f}(\mathbf{x}^*) \geq \xi$ then
        $\mathcal{A} \leftarrow \mathcal{A} \cup \{\mathbf{x}^*\}$
        $R(\mathbf{x}^*) \leftarrow R(\mathbf{x}^*) \cup \{\mathbf{x}\}$

$\mathcal{C} \leftarrow \{$maximal $C \subseteq \mathcal{A} \mid \forall \mathbf{x}_i^*, \mathbf{x}_j^* \in C, \mathbf{x}_i^*$ and $\mathbf{x}_j^*$ are density reachable$\}$
foreach $C \in \mathcal{C}$ do
    foreach $\mathbf{x}^* \in C$ do $C \leftarrow C \cup R(\mathbf{x}^*)$
return $\mathcal{C}$

FindAttractor $(\mathbf{x}, \mathbf{D}, h, \epsilon)$:
$t \leftarrow 0$
$\mathbf{x}_t \leftarrow \mathbf{x}$
repeat
    $\mathbf{x}_{t+1} \leftarrow \frac{\sum_{i=1}^{n} K\left(\frac{\mathbf{x}_t - \mathbf{x}_i}{h}\right) \cdot \mathbf{x}_t}{\sum_{i=1}^{n} K\left(\frac{\mathbf{x}_t - \mathbf{x}_i}{h}\right)}$
    $t \leftarrow t + 1$
until $\|\mathbf{x}_t - \mathbf{x}_{t-1}\| \leq \varepsilon$
return $\mathbf{x}_t$
```

The R implementation

Please take a look at the R codes file `ch_06_denclue.R` from the bundle of R codes for previously mentioned algorithm. The codes can be tested with the following command:

```
> source("ch_06_denclue.R")
```

Visitor analysis in the browser cache

The browser-cache analysis provides the website owner with the convenience that shows the best matched part to the visitors, and at the same time, it is related to their privacy protection. The data instances in this context are browser caches, sessions, cookies, various logs, and so on.

The possible factors included in certain data instances can be the Web address, IP address (denotes the position where the visitor comes from), the duration for which the visitor stayed on a specific page, the pages the user visited, the sequence of the visited pages, the date and time of every visit, and so on. The log can be specific to a certain website or to various websites. A more detailed description is given in the following table:

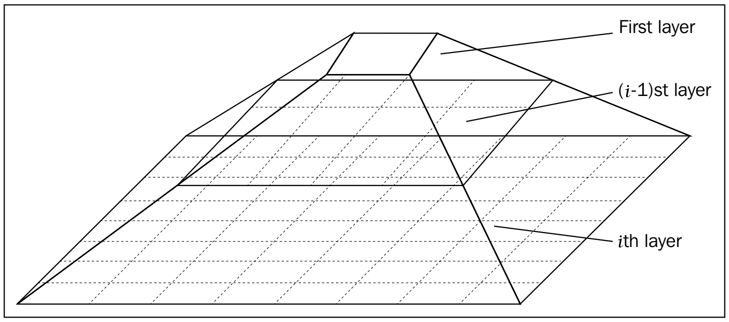
Hit	This refers to each element of a web page downloaded to a viewer's web browser (such as Internet Explorer, Mozilla, or Netscape). Hits do not correspond in any direct fashion to the number of pages viewed or number of visitors to a site. For example, if a viewer downloads a web page with three graphics, the web logfile will show four hits: one for the web page and one for each of the three graphics.
Unique Visitors	The actual number of viewers to the website that came from a unique IP address (see IP address in this table).
New/Return Visitors	The number of first-time visitors to the site compared to returning visitors.
Page views	The number of times a specified web page has been viewed; shows exactly what content people are (or are not) viewing at a website. Every time a visitor hits the page refresh button, another page view is logged.
Page views per visitor	The number of page views divided by the number of visitors; measures how many pages viewers look at each time they visit a website.
IP address	A numeric identifier for a computer. (The format of an IP address is a 32-bit numeric address written as four numbers separated by periods; each number can be zero to 255. For example, 1.160.10.240 could be an IP address.) The IP address can be used to determine a viewer's origin (that is, by country); it also can be used to determine the particular computer network a website's visitors are coming from.
Visitor location	The geographic location of the visitor.
Visitor language	The language setting on the visitor's computer.
Referring pages/sites (URLs)	Indicates how visitors get to a website (that is, whether they type the URL, or web address, directly into a web browser or if they click through from a link at another site).

Keywords	If the referring URL is a search engine, the keywords (search string) that the visitor used can be determined.
Browser type	The type of browser software a visitor is using (that is, Netscape, Mozilla, Internet Explorer, and so on)
Operating system version	The specific operating system the site visitor uses.
Screen resolution	The display settings for the visitor's computer.
Java or Flash-enabled	Whether or not the visitor's computer allows Java (a programming language for applications on the Web) and/or Flash (a software tool that allows web pages to be displayed with animation, or motion).
Connection speed	Whether visitors are accessing the website from a slower dial-up connection, high-speed broadband, or Tl.
Errors	The number of errors recorded by the server, such as a "404-file not found" error; can be used to identify broken links and other problems at the website.
Visit duration	Average time spent on the site (length the visitor stays on the site before leaving). Sites that retain visitors longer are referred to as "sticky" sites.
Visitor paths/navigation	How visitors navigate the website, by specific pages, most common entry pages (the first page accessed by a visitor at a website) and exit points (the page from which a visitor exits a Website), and so on. For example, if a large number of visitors leave the site after looking at a particular page, the analyst might infer that they either found the information they needed, or alternatively, there might be a problem with that page (is it the page where shipping and handling fees are posted, which maybe are large enough to turn visitors away?).
Bounce rate	The percentage of visitors who leave the site after the first page; calculated by the number of visitors who visit only a single page divided by the number of total visits. The bounce rate is sometimes used as another indicator of "stickiness."

The analysis of a visitor is basically history sniffing, which is used for user-behavior analysis.

Recommendation system and STING

STatistical Information Grid (**STING**) is a grid-based clustering algorithm. The dataset is recursively divided into a hierarchy structure. The whole input dataset serves as the root node in the hierarchy structure. Each cell/unit in a layer is composed of a couple of cells/units in the lower layer. An example is shown in the following diagram:



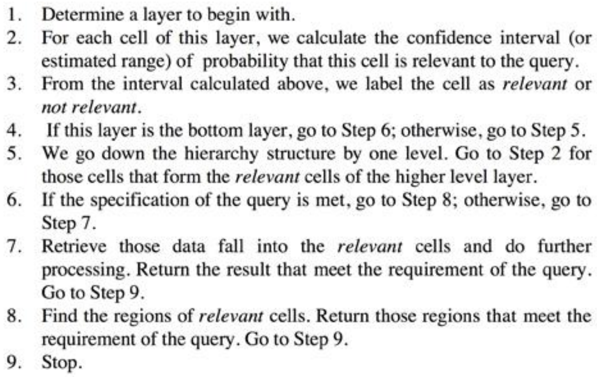
To support the query for a dataset, the statistical information of each unit is calculated in advance for further processing; this information is also called statistics parameters.

The characteristics of STING algorithms are (but not limited to) the following:

- A query-independent structure
- Intrinsically parallelizable
- Efficiency

The STING algorithm

The summarized pseudocodes for the STING algorithm are as follows:



```
1. Determine a layer to begin with.
2. For each cell of this layer, we calculate the confidence interval (or
   estimated range) of probability that this cell is relevant to the query.
3. From the interval calculated above, we label the cell as relevant or
   not relevant.
4. If this layer is the bottom layer, go to Step 6; otherwise, go to Step 5.
5. We go down the hierarchy structure by one level. Go to Step 2 for
   those cells that form the relevant cells of the higher level layer.
6. If the specification of the query is met, go to Step 8; otherwise, go to
   Step 7.
7. Retrieve those data fall into the relevant cells and do further
   processing. Return the result that meet the requirement of the query.
   Go to Step 9.
8. Find the regions of relevant cells. Return those regions that meet the
   requirement of the query. Go to Step 9.
9. Stop.
```

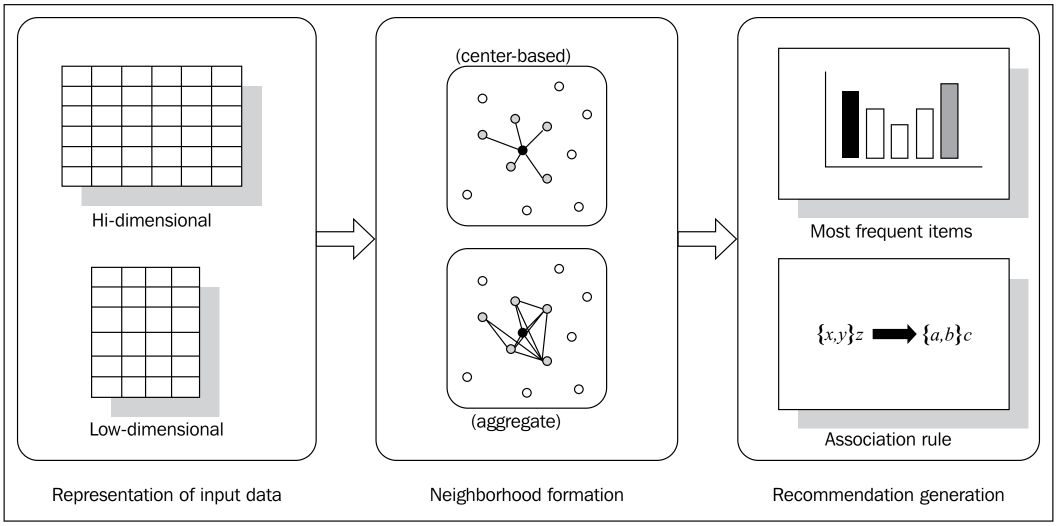
The R implementation

Please take a look at the R codes file `ch_06_sting.R` from the bundle of R codes for the previously mentioned algorithm. The codes can be tested with the following command:

```
> source("ch_06_sting.R")
```

Recommendation systems

Depending on statistical, data-mining, and knowledge-discovery techniques, recommendation systems are being used by most of the e-commerce sites to make it easy for consumers to find products to purchase. Three main parts: representation of input data, neighborhood formation, and recommendation generation are shown in the following diagram:



Web sentiment analysis and CLIQUE

CLustering In QUEst (**CLIQUE**) is a bottom-up and grid-based clustering algorithm. The idea behind this algorithm is the Apriori feature, that is, the monotonicity of dense units with respect to dimensionality. If a set of data points, *S*, is a cluster in a k-dimensional projection of the space, then *S* is also contained in a cluster in any (k-1)-dimensional projections of this space.

The algorithm proceeds by passes. The one-dimensional dense units are produced by one pass through the data. The candidate k-dimensional units are generated using the candidate-generation procedure and the determined (k-l)-dimensional dense units that are fetched at the (k-1) pass.

The characteristics of the CLIQUE algorithm are as follows:

- Efficient for high-dimensional datasets
- Interpretability of results
- Scalability and usability

The CLIQUE algorithm contains three steps to cluster a dataset. First, a group of subspaces is selected to cluster the dataset. Then, clustering is independently executed in every subspace. Finally, a concise summary of every cluster is produced in the form of a **disjunctive normal form** (**DNF**) expression.

The CLIQUE algorithm

The summarized pseudocode for the CLIQUE algorithm is as follows:

1. Indentification of subspaces that contain clusters.
2. Indentification of cluster.
3. Generation minimal description for the cluster.

The candidate-generation algorithm is illustrated as follows:

```
insert into $C_k$
Select $u_1.[l_1, h_1), u_1.[l_2, h_2), \ldots,$
    $u_1.[l_{k-1}, h_{k-1}), u_2.[l_{k-1}, h_{k-1})$
from $D_{k-1}\ u_1, D_{k-1}\ \mathrm{u}_2$
Where $u_1.a_1 = u_2.a_1, u_1.l_1 = u_2.l_1, u_1.h_1 = u_2.h_1,$
    $u_1.a_2 = u_2.a_2, u_1.l_2 = u_2.l_2, u_1.h_2 = u_2.h_2, \ldots,$
    $u_1.a_{k-2} = u_2.a_{k-2}, u_1.l_{k-2} = u_2.l_{k-2}, u_1.h_{k-2} = u_2.h_{k-2},$
    $u_1.a_{k-1} < u_2.a_{k-1}$
```

Here is the algorithm to find the connected components of the graph; this is equivalent to finding clusters:

```
input: starting unit $u$= $\{[l_1, h_1),...[l_k, h_k)\}$
          clusternumber $n$
dfs $(u, n)$
$u. num=n$
for ( $j = 1; j < k; j$ ++) do begin
    $ul = \{[ l1, h1 ),... [(l_j^l), (h_j^l)),..., [l_k , h_k )\}$
    if( $u^l$ is dense ) and ( $u^l. num$ is undefined )
    dfs $(u^l, n)$
    $u^r = \{[ l1, h1 ), ..., [( l_j^r), (h_j^r)),...,[ l_k , h_k )\}$
    if( $u^r$ is dense ) and ( $u^r. num$ is undefined )
    dfs $(u^r, n)$
end
```

The R implementation

Please take a look at the R codes file `ch_06_clique.R` from the bundle of R codes for the previously mentioned algorithm. The codes can be tested with the following command:

```
> source("ch_06_clique.R")
```

Web sentiment analysis

Web sentiment analysis is used to identify the idea or thought behind the text, for example, the sentiment analysis of microblogs such as Twitter. One simple example is comparing the post with a predefined labeled word list for sentiment judging. Another example is that we can judge a movie review as thumbs up or thumbs down.

Web sentiment analyses are used in news article analyses of biases pertaining to specific views, newsgroup evaluation, and so on.

Opinion mining and WAVE clustering

The WAVE clustering algorithm is a grid-based clustering algorithm. It depends on the relation between spatial dataset and multidimensional signals. The idea is that the cluster in a multidimensional spatial dataset turns out to be more distinguishable after a wavelet transformation, that is, after applying wavelets to the input data or the preprocessed input dataset. The dense part segmented by the sparse area in the transformed result represents clusters.

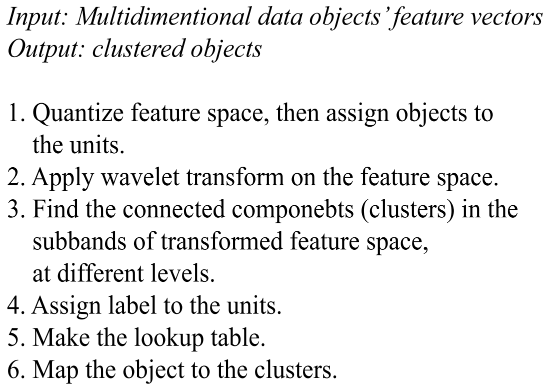
The characteristics of the WAVE cluster algorithm are as follows:

- Efficient for a large dataset
- Efficient for finding various shapes of clusters
- Insensitive to noise or outlier
- Insensitive with respect to the input order of a dataset
- Multiresolution, which is introduced by wavelet transforms
- Applicable to any numerical dataset

The WAVE cluster algorithm performs a few steps. In the first step, it creates a grid and assigns each data object from the input dataset to a unit in the grid. And then, transform the data to a new space by applying wavelet transform functions. Third, find the connected component in the new space. Map the cluster label for related data object to the original data space.

The WAVE cluster algorithm

The summarized pseudocodes for the WAVE cluster algorithm are as follows:



```
Input: Multidimentional data objects' feature vectors
Output: clustered objects

1. Quantize feature space, then assign objects to
   the units.
2. Apply wavelet transform on the feature space.
3. Find the connected componebts (clusters) in the
   subbands of transformed feature space,
   at different levels.
4. Assign label to the units.
5. Make the lookup table.
6. Map the object to the clusters.
```

The R implementation

Please take a look at the R codes file `ch_06_wave.R` from the bundle of R codes for the previously mentioned algorithm. The codes can be tested with the following command:

```
> source("ch_06_wave.R")
```

Opinion mining

An entity possesses a couple of features. Features might be explicit or implicit. If a person or organization expresses an opinion, then the person or organization should be an opinion holder. An opinion specific to a feature is a positive or negative view, attitude, emotion, or appraisal of the feature from the opinion holder. Whether the opinion on a feature is positive, negative, or neutral denotes opinion orientation.

Opinion mining means mining the opinion on certain features of the object or entity under research. The simplest case is to judge the opinion as positive or negative.

An opinion-orientation algorithm can be listed as follows:

- Identify opinion words and phrases
- Handle negation
- The But clause
- Aggregating opinion

$$Score(f_i, s) = \sum_{op_j \in s} \frac{op_j \cdot oo}{d(op_j, f_i)}$$

The preceding formula denotes the opinion orientation on a certain feature, f_i, where op_j is an opinion word in *s*, $d(op_j, f_i)$ is the distance between f_i and op_j. $op_j \cdot oo$ is the opinion orientation.

User search intent and the EM algorithm

The **Expectation Maximization** (**EM**) algorithm is a probabilistic-model-based clustering algorithm that depends on the mixture model in which the data is modeled by a mixture of simple models. The parameters related to these models are estimated by **Maximum Likelihood Estimation** (**MLE**).

Mixture models assume that the data is the result of the combination of various simple probabilistic distribution functions. Given *K* distribution functions and the *j*th distribution with the parameter, θ_j, Θ is the set of θ_j of all distributions:

$$p\left(x\middle|\Theta\right)=\sum_{j=1}^{K} w_j p_j(x\,|\,\theta_j)$$

The EM algorithm performs in the following way. In the first step, an initial group of model parameters are selected. The expectation step is the second step that performs the calculation of the probability:

$$P\left(distribution\ \ j\,|\,x_i,\theta\right)=\frac{P\left(x_i\,|\,\theta_j\right)}{\sum_{l=1}^{K} P\left(x_i\,|\,\theta_l\right)}$$

The previous equation represents the probability of each data object belonging to each distribution. Maximization is the third step. With the result of the expectation step, update the estimation of the parameters with the ones that maximize the expected likelihood:

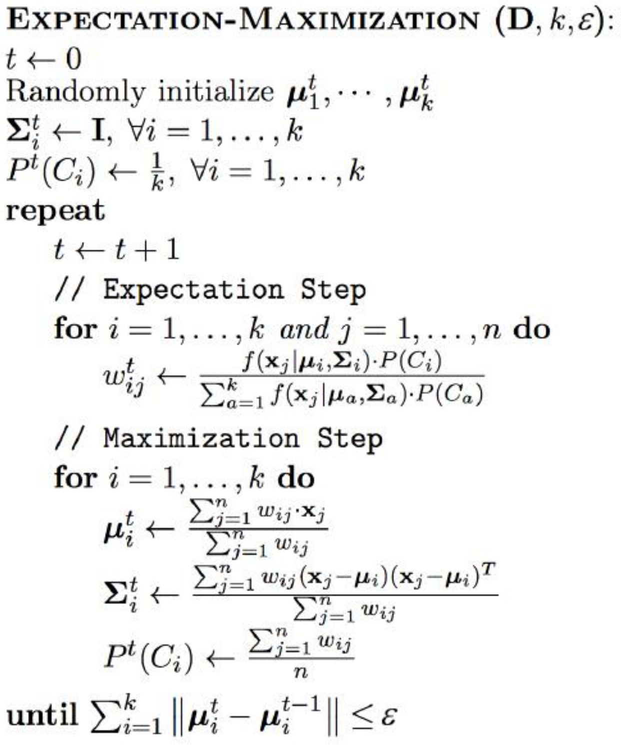
$$\mu_j = \frac{1}{k}\sum_{i=1}^{n} o_i \frac{P(\Theta_j\left|\mathrm{o}_i,\Theta\right.)}{\sum_{l=1}^{n} P(\Theta_j\left|\mathrm{o}_l,\Theta\right.)} = \frac{1}{k}\,\frac{\sum_{i=1}^{n} o_i P(\Theta_j\left|\mathrm{o}_i,\Theta\right.)}{\sum_{i=1}^{n} P(\Theta_j\left|\mathrm{o}_i,\Theta\right.)}$$

$$\sigma_j = \sqrt{\frac{\sum_{i=1}^{n} p(\Theta_j\left|\mathrm{o}_i,\Theta\right.)(\mathrm{o}_i - u_j)^2}{\sum_{i=1}^{n} p(\Theta_j\left|\mathrm{o}_i,\Theta\right.)}}$$

The expectation step and maximization step are performed repeatedly until the output matches the ending condition, that is, the changes of the parameter estimations below a certain threshold.

The EM algorithm

The summarized pseudocode for the EM algorithm are as follows:



```
EXPECTATION-MAXIMIZATION $(\mathbf{D}, k, \varepsilon)$:
$t \leftarrow 0$
Randomly initialize $\boldsymbol{\mu}_1^t, \cdots, \boldsymbol{\mu}_k^t$
$\boldsymbol{\Sigma}_i^t \leftarrow \mathbf{I}, \forall i = 1, \ldots, k$
$P^t(C_i) \leftarrow \frac{1}{k}, \forall i = 1, \ldots, k$
repeat
    $t \leftarrow t + 1$
    // Expectation Step
    for $i = 1, \ldots, k$ and $j = 1, \ldots, n$ do
        $w_{ij}^t \leftarrow \frac{f(\mathbf{x}_j | \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \cdot P(C_i)}{\sum_{a=1}^k f(\mathbf{x}_j | \boldsymbol{\mu}_a, \boldsymbol{\Sigma}_a) \cdot P(C_a)}$
    // Maximization Step
    for $i = 1, \ldots, k$ do
        $\boldsymbol{\mu}_i^t \leftarrow \frac{\sum_{j=1}^n w_{ij} \cdot \mathbf{x}_j}{\sum_{j=1}^n w_{ij}}$
        $\boldsymbol{\Sigma}_i^t \leftarrow \frac{\sum_{j=1}^n w_{ij} (\mathbf{x}_j - \boldsymbol{\mu}_i)(\mathbf{x}_j - \boldsymbol{\mu}_i)^T}{\sum_{j=1}^n w_{ij}}$
        $P^t(C_i) \leftarrow \frac{\sum_{j=1}^n w_{ij}}{n}$
until $\sum_{i=1}^k \left\| \boldsymbol{\mu}_i^t - \boldsymbol{\mu}_i^{t-1} \right\| \leq \varepsilon$
```

The R implementation

Please take a look at the R codes file `ch_06_em.R` from the bundle of R codes for the previously mentioned algorithm. The codes can be tested with the following command:

```
> source("ch_06_em.R")
```

The user search intent

Determining the user search intent is an important yet a difficult issue with respect to the sparse data available concerning the searcher and queries.

User intent has a wide range of applications, cluster query refinements, user intention profiles, and web search intent induction. Given Web search engine queries, finding user intention is also a key and requirement.

To determine the user interest and preferences, the clicks' sequence upon the search result can be used as a good base.

Web search personalization is another important application with the user search intention. It is related to user context and intention. With the user intention applied, more effective and efficient information will be provided.

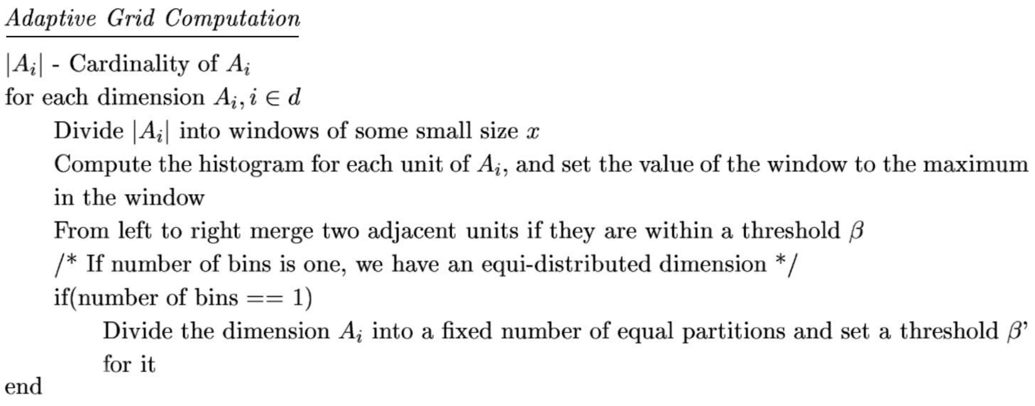
Customer purchase data analysis and clustering high-dimensional data

For high-dimensional data-space clustering, two major issues occur: efficiency and quality. New algorithms are needed to deal with this type of dataset. Two popular strategies are applied to it. One is the subspace-clustering strategy to find the cluster in the subspace of the original dataset space. Another is the dimensionality-reduction strategy, which is a lower dimensional data space created for further clustering.

MAFIA is an efficient and scalable subspace-clustering algorithm for high-dimensional and large datasets.

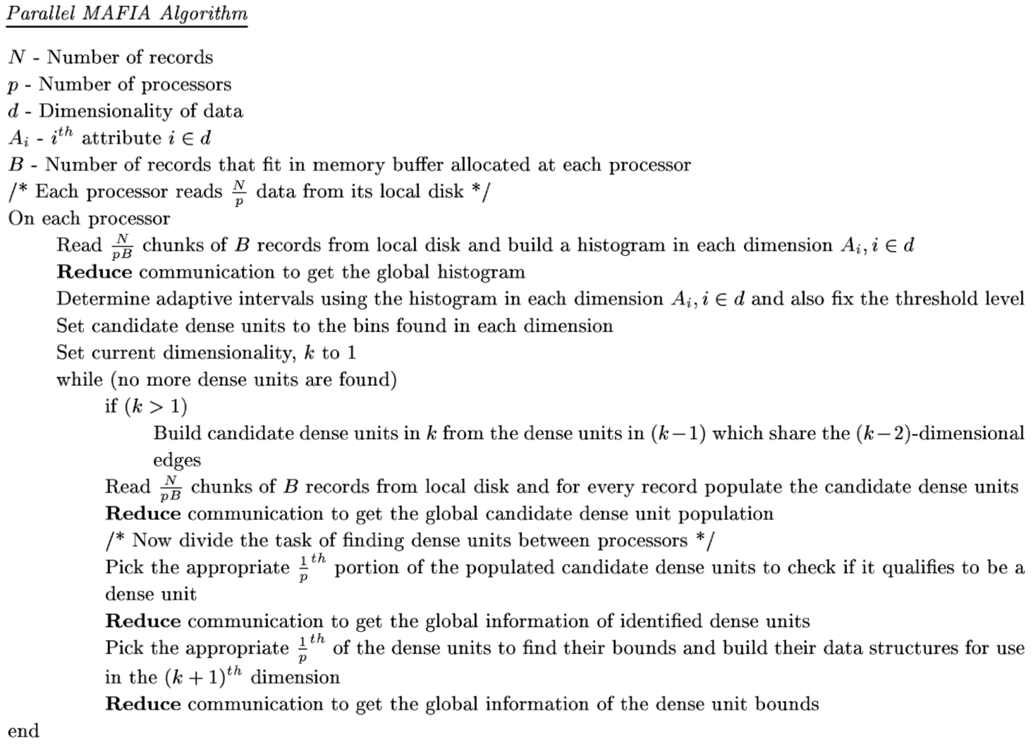
The MAFIA algorithm

The summarized pseudocode for the MAFIA algorithm is as follows:



```
Adaptive Grid Computation
$|A_i|$ - Cardinality of $A_i$
for each dimension $A_i, i \in d$
    Divide $|A_i|$ into windows of some small size $x$
    Compute the histogram for each unit of $A_i$, and set the value of the window to the maximum
    in the window
    From left to right merge two adjacent units if they are within a threshold $\beta$
    /* If number of bins is one, we have an equi-distributed dimension */
    if(number of bins == 1)
        Divide the dimension $A_i$ into a fixed number of equal partitions and set a threshold $\beta$'
        for it
end
```

The summarized pseudocode for the parallel MAFIA algorithm is as follows:

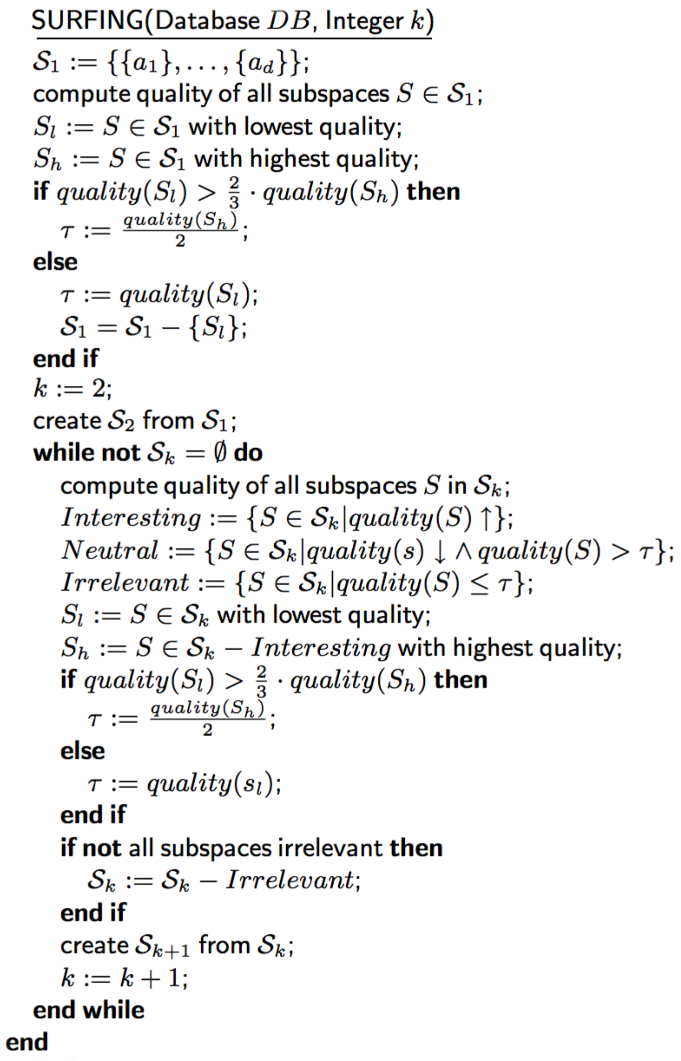


Parallel MAFIA Algorithm

```
$N$ - Number of records
$p$ - Number of processors
$d$ - Dimensionality of data
$A_i$ - $i^{th}$ attribute $i \in d$
$B$ - Number of records that fit in memory buffer allocated at each processor
/* Each processor reads $\frac{N}{p}$ data from its local disk */
On each processor
    Read $\frac{N}{pB}$ chunks of $B$ records from local disk and build a histogram in each dimension $A_i, i \in d$
    **Reduce** communication to get the global histogram
    Determine adaptive intervals using the histogram in each dimension $A_i, i \in d$ and also fix the threshold level
    Set candidate dense units to the bins found in each dimension
    Set current dimensionality, $k$ to 1
    while (no more dense units are found)
        if $(k > 1)$
            Build candidate dense units in $k$ from the dense units in $(k-1)$ which share the $(k-2)$-dimensional
            edges
        Read $\frac{N}{pB}$ chunks of $B$ records from local disk and for every record populate the candidate dense units
        **Reduce** communication to get the global candidate dense unit population
        /* Now divide the task of finding dense units between processors */
        Pick the appropriate $\frac{1}{p}^{th}$ portion of the populated candidate dense units to check if it qualifies to be a
        dense unit
        **Reduce** communication to get the global information of identified dense units
        Pick the appropriate $\frac{1}{p}^{th}$ of the dense units to find their bounds and build their data structures for use
        in the $(k+1)^{th}$ dimension
        **Reduce** communication to get the global information of the dense unit bounds
end
```

The SURFING algorithm

The summarized pseudocodes for the SURFING algorithm are as follows. It selects interesting features from the original attributes of the dataset.



```
SURFING(Database $DB$, Integer $k$)
$\mathcal{S}_1 := \{\{a_1\}, \ldots, \{a_d\}\}$;
compute quality of all subspaces $S \in \mathcal{S}_1$;
$S_l := S \in \mathcal{S}_1$ with lowest quality;
$S_h := S \in \mathcal{S}_1$ with highest quality;
if $quality(S_l) > \frac{2}{3} \cdot quality(S_h)$ then
   $\tau := \frac{quality(S_h)}{2}$;
else
   $\tau := quality(S_l)$;
   $\mathcal{S}_1 = \mathcal{S}_1 - \{S_l\}$;
end if
$k := 2$;
create $\mathcal{S}_2$ from $\mathcal{S}_1$;
while not $\mathcal{S}_k = \emptyset$ do
   compute quality of all subspaces $S$ in $\mathcal{S}_k$;
   $Interesting := \{S \in \mathcal{S}_k | quality(S) \uparrow\}$;
   $Neutral := \{S \in \mathcal{S}_k | quality(s) \downarrow \wedge quality(S) > \tau\}$;
   $Irrelevant := \{S \in \mathcal{S}_k | quality(S) \leq \tau\}$;
   $S_l := S \in \mathcal{S}_k$ with lowest quality;
   $S_h := S \in \mathcal{S}_k - Interesting$ with highest quality;
   if $quality(S_l) > \frac{2}{3} \cdot quality(S_h)$ then
      $\tau := \frac{quality(S_h)}{2}$;
   else
      $\tau := quality(s_l)$;
   end if
   if not all subspaces irrelevant then
      $\mathcal{S}_k := \mathcal{S}_k - Irrelevant$;
   end if
   create $\mathcal{S}_{k+1}$ from $\mathcal{S}_k$;
   $k := k + 1$;
end while
end
```

The R implementation

Please take a look at the R codes file `ch_06_surfing.R` from the bundle of R codes for the previously mentioned algorithm. The codes can be tested with the following command:

```
> source("ch_06_surfing.R")
```

Customer purchase data analysis

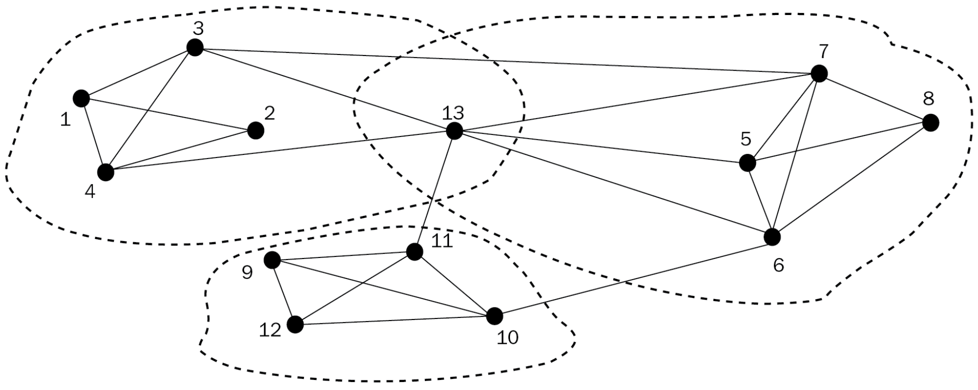
Customer purchase data analysis contains many applications such as the customer-satisfaction analysis.

From the customer purchase data analysis, one application helps find the unwanted consumption or user's purchase behavior.

SNS and clustering graph and network data

The clustering for graph and network data has a wide application in modern life, such as social networking. However, more challenges crop up along with the needs. High computational cost, sophisticated graphs, and high dimensionality and sparsity are the major concerns. With some special transformations, the issues can be transformed into graph cut issues.

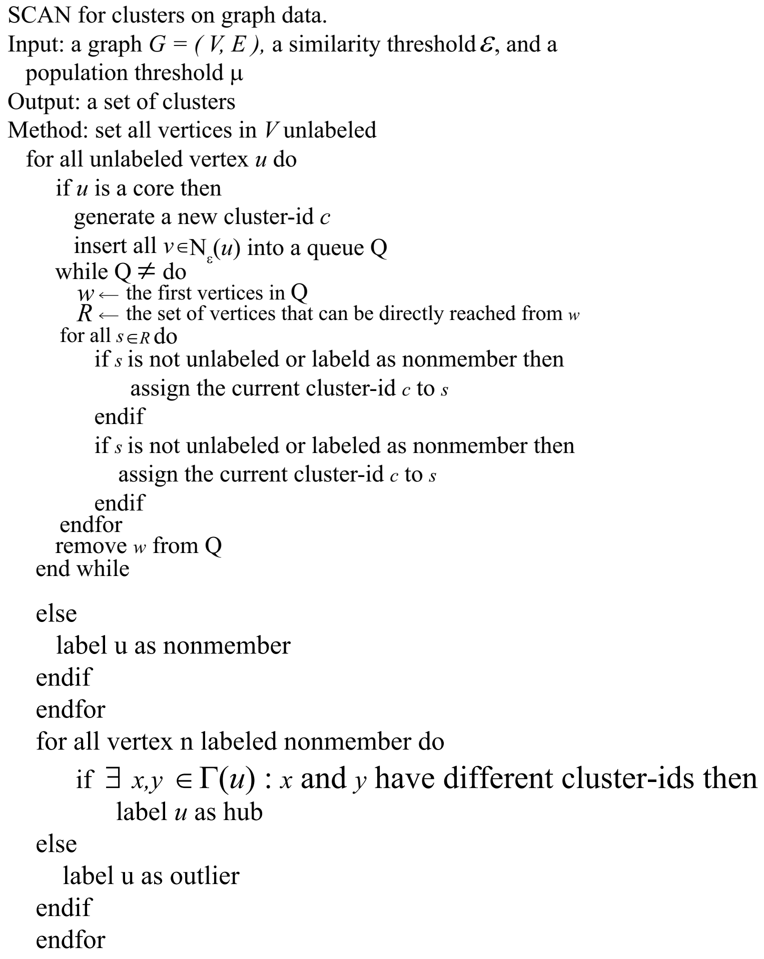
Structural Clustering Algorithm for Network (**SCAN**) is one of the algorithms that searches for well-connected components in the graph as clusters.



An example of overlaping communities [26]

The SCAN algorithm

The summarized pseudocodes for the SCAN algorithm are as follows:



```
SCAN for clusters on graph data.
Input: a graph $G = ( V, E )$, a similarity threshold $\varepsilon$, and a
  population threshold $\mu$
Output: a set of clusters
Method: set all vertices in $V$ unlabeled
  for all unlabeled vertex $u$ do
     if $u$ is a core then
       generate a new cluster-id $c$
       insert all $v \in N_\varepsilon(u)$ into a queue Q
     while Q $\neq$ do
       $w \leftarrow$ the first vertices in Q
       $R \leftarrow$ the set of vertices that can be directly reached from $w$
      for all $s \in R$ do
          if $s$ is not unlabeled or labeld as nonmember then
               assign the current cluster-id $c$ to $s$
          endif
          if $s$ is not unlabeled or labeled as nonmember then
             assign the current cluster-id $c$ to $s$
          endif
      endfor
     remove $w$ from Q
   end while
   else
     label u as nonmember
   endif
   endfor
   for all vertex n labeled nonmember do
       if $\exists\ x,y \in \Gamma(u)$ : $x$ and $y$ have different cluster-ids then
           label $u$ as hub
   else
      label u as outlier
   endif
   endfor
```

The R implementation

Please take a look at the R codes file `ch_06_scan.R` from the bundle of R codes for the previously mentioned algorithms. The codes can be tested with the following command:

```
> source("ch_06_scan.R")
```

Social networking service (SNS)

Social network has become the most popular online communication method nowadays. The analysis of SNS becomes important, because of the requirement of security, business, control, and so on.

The foundation of SNS is a graph theory, especially for SNS mining, such as finding social communities and abuse of SNS for bad purpose.

The clustering for SNS is an inherent application to find a community (or community detection). Random walk is another key technology for SNS analysis and is used to find communities.

Time for action

Here are some questions for you to know whether you have understood the concepts:

- What is the DBSCAN algorithm?
- What is the SCAN algorithm?
- What is the STING algorithm?
- What is the OPTICS algorithm?
- What is the constraint-based cluster method?

Summary

In this chapter, we covered the following facts:

- DBSCAN depends on the density-based description of clusters. With searching and measuring the density of data points, high density means high possibility of the existence of clusters, and others mean outliers or noise.
- OPTICS produces the cluster ordering that consists of the order of the data objects together with the corresponding reachability values and core values.
- You learned that DENCLUE is a clustering algorithm based on a set of specific density-distribution functions and can find arbitrary shape clusters. It first segments the dataset as cubes and identifies the local density-distribution function. You also learned that a hill-climbing algorithm is performed to retrieve the local maximum for each item in the related cube with which a cluster will be built.
- We saw that STING is based on the grid-like data structure, and it segments the embedding spatial area of the input data points into rectangular units. It is mainly used for a spatial dataset.

- You learned that CLIQUE is a grid-based clustering algorithm that finds subspaces of high-dimensional data, and it can find dense units in the high-dimensional data too.
- You know that the WAVE cluster is a grid-based clustering algorithm based on the wavelet transformations. It has a multiresolution and is efficient for large dataset.
- You learned that EM is a probabilistic-model-based clustering algorithm, where each data point with a probability indicates that it belongs to a cluster. It is based on the assumption that the attribute of the data point has values that is the linear combination of simple distributions.
- Clustering high-dimensional data.
- Clustering graph and network data.

In the next chapter, we will cover the major topics related to outlier detection and algorithms, and look at some of their examples.

7

Outlier Detection

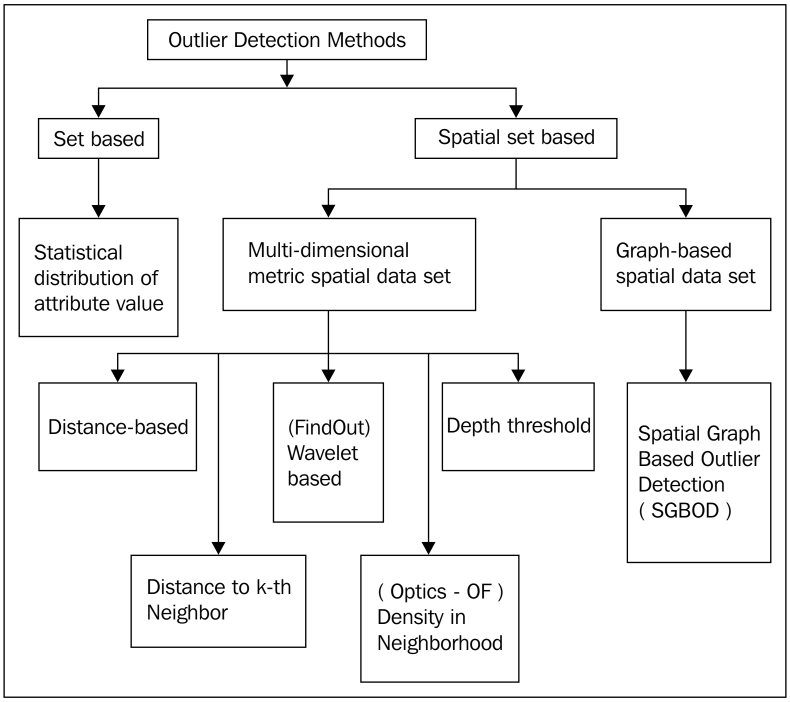
In this chapter, you will learn how to write R codes to detect outliers in real-world cases. Generally speaking, outliers arise for various reasons, such as the dataset being compromised with data from different classes and data measurement system errors.

As per their characteristics, outliers differ dramatically from the usual data in the original dataset. Versatile solutions are developed to detect them, which include model-based methods, proximity-based methods, density-based methods, and so on.

In this chapter, we will cover the following topics:

- Credit card fraud detection and statistical methods
- Activity monitoring—the detection of fraud of mobile phones and proximity-based methods
- Intrusion detection and density-based methods
- Intrusion detection and clustering-based methods
- Monitoring the performance of network-based and classification-based methods
- Detecting novelty in text, topic detection, and mining contextual outliers
- Collective outliers on spatial data
- Outlier detection in high-dimensional data

Here is a diagram illustrating a classification of outlier detection methods:

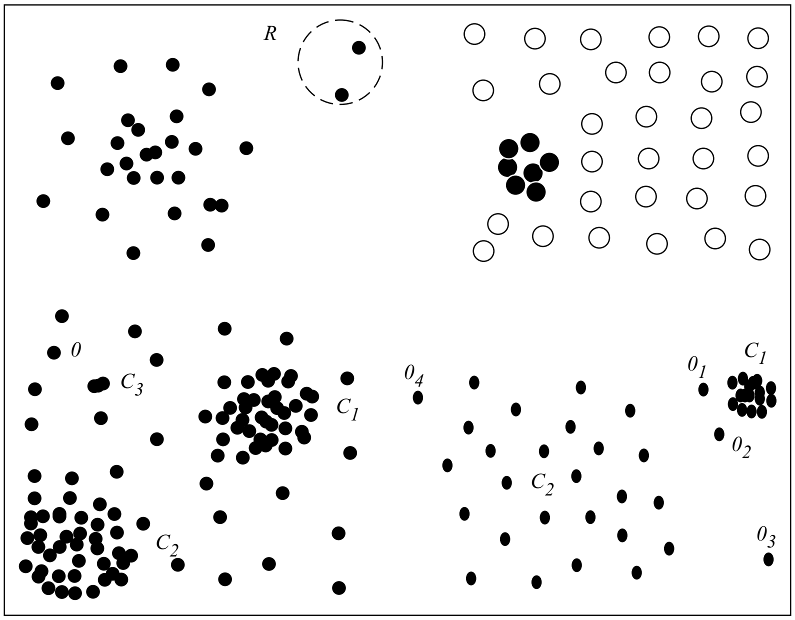

Outlier Detection Methods
Set based
Spatial set based
Statistical distribution of attribute value
Multi-dimensional metric spatial data set
Graph-based spatial data set
Distance-based
(FindOut) Wavelet based
Depth threshold
Spatial Graph Based Outlier Detection (SGBOD)
Distance to k-th Neighbor
(Optics - OF) Density in Neighborhood

The output of an outlier detection system can be categorized into two groups: one is the **labeled result** and the other is the **scored result** (or an ordered list).

Credit card fraud detection and statistical methods

One major solution to detect outliers is the model-based method or statistical method. The outlier is defined as the object not belonging to the model that is used to represent the original dataset. In other words, that model does not generate the outlier.

Among the accurate models to be adopted for the specific dataset, there are many choices available such as Gaussian and Poisson. If the wrong model is used to detect outliers, the normal data point may wrongly be recognized as an outlier. In addition to applying the single distribution model, the mixture of distribution models is practical too.



The log-likelihood function is adopted to find the estimation of parameters of a model:

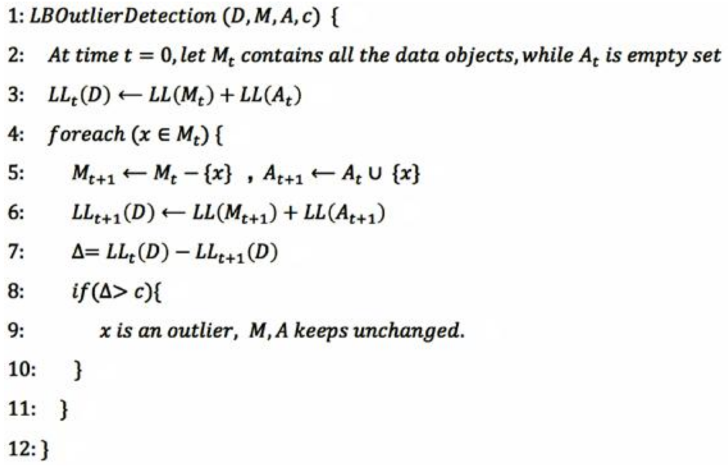
$$\text{In}\,\mathcal{L}(\mu,\sigma^2) = \sum_{i=1}^{n} \text{In}\ f(x_i \left| (\mu,\sigma^2)\right.) = -\frac{n}{2}\text{In}\ (2\pi) - \frac{n}{2}\text{In}\,\sigma^2 - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(\mathrm{x}_i - \mu)^2$$

$$\hat{\mu} = \overline{x} = \frac{1}{n}\sum_{i=1}^{n}\mathrm{x}_i$$

$$\hat{\sigma}^2 - \frac{1}{n}\sum_{i=1}^{n}\left(x_i - \overline{x}\right)^2$$

The likelihood-based outlier detection algorithm

The summarized pseudocode of the likelihood-based outlier detection algorithm is as follows:



```
$LBOutlierDetection\ (D, M, A, c)$ {
  $At\ time\ t = 0, let\ M_t\ contains\ all\ the\ data\ objects, while\ A_t\ is\ empty\ set$
  $LL_t(D) \longleftarrow LL(M_t) + LL(A_t)$
  $foreach\ (x \in M_t)$ {
      $M_{t+1} \longleftarrow M_t - \{x\}$ , $A_{t+1} \longleftarrow A_t \cup \{x\}$
      $LL_{t+1}(D) \longleftarrow LL(M_{t+1}) + LL(A_{t+1})$
      $\Delta = LL_t(D) - LL_{t+1}(D)$
      $if(\Delta > c)${
          $x\ is\ an\ outlier,\ M, A\ keeps\ unchanged.$
      }
  }
}
```

The R implementation

Look up the file of R codes, `ch_07_lboutlier_detection.R`, from the bundle of R codes for the previously mentioned algorithm. The codes can be tested with the following command:

```
> source("ch_07_lboutlier_detection.R")
```

Credit card fraud detection

Fraud denotes the criminal activities that happen in various commercial companies, such as credit card, banking, or insurance companies. For credit card fraud detection, two major applications are covered, fraudulent application of credit card and fraudulent usage of credit card. The fraud represents behavior anomalous to the average usage of credit cards to certain users, that is, transaction records of the users.

This kind of outlier statistically denotes credit card theft, which deviates from the normal nature of criminal activities. Some examples of outliers in this case are high rate of purchase, very high payments, and so on.

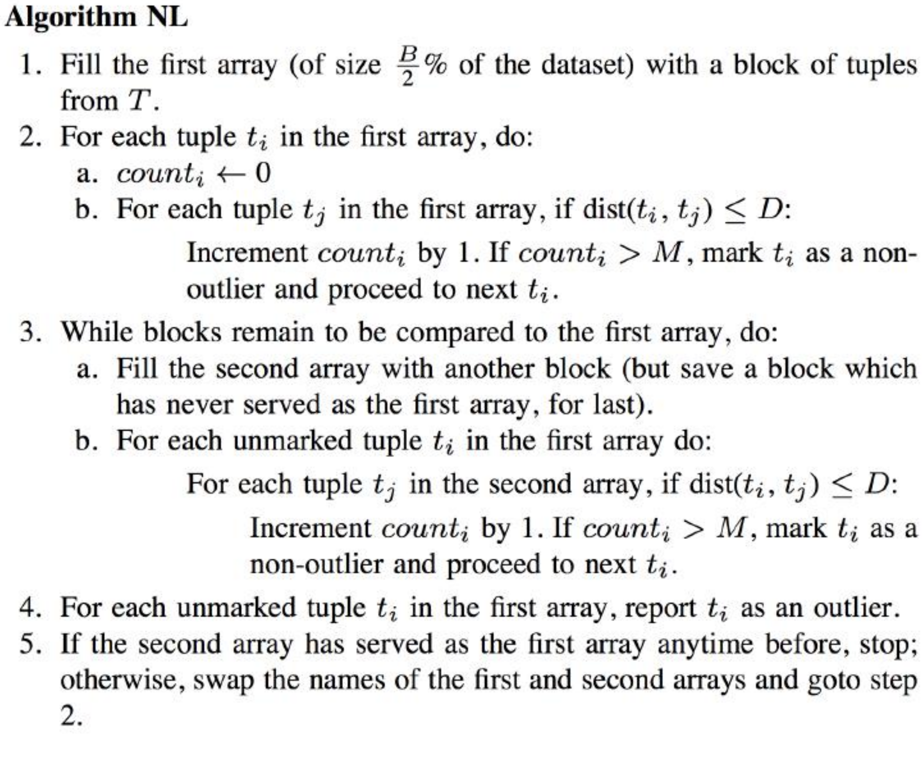
The location of payment, the user, and the context are possible attributes in the dataset. The clustering algorithms are the possible solutions.

Activity monitoring – the detection of fraud involving mobile phones and proximity-based methods

Two major approaches of proximity-based methods are distance-based and density-based outlier detection algorithms.

The NL algorithm

The summarized pseudocodes of the NL algorithm are as follows:

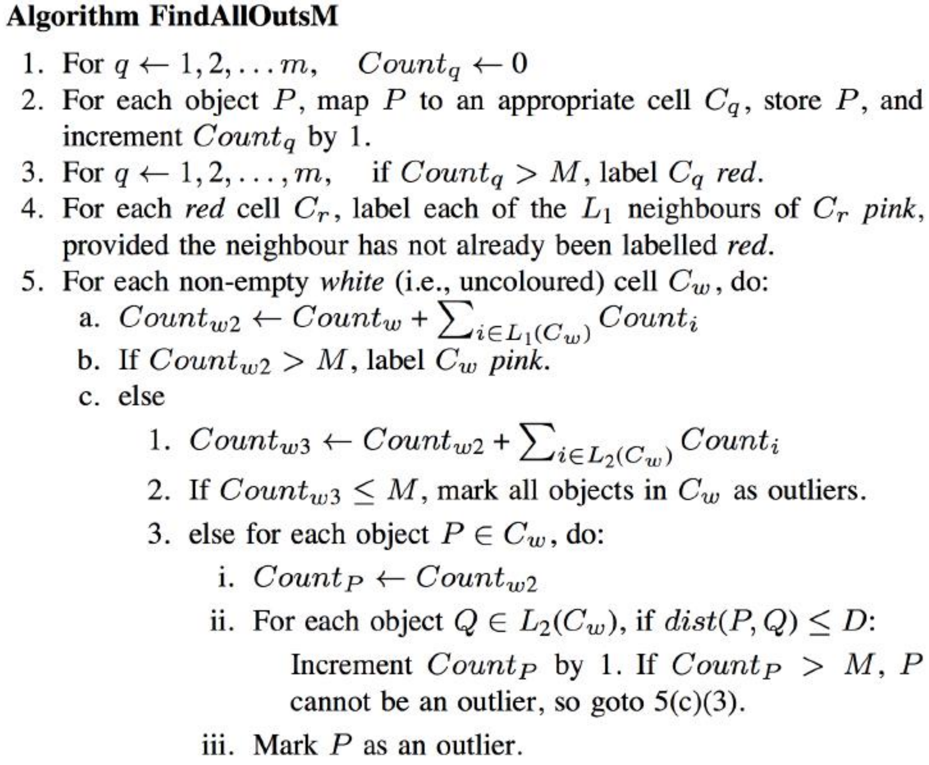


Algorithm NL

```
1. Fill the first array (of size $\frac{B}{2}$% of the dataset) with a block of tuples
   from $T$.
2. For each tuple $t_i$ in the first array, do:
   a. $count_i \leftarrow 0$
   b. For each tuple $t_j$ in the first array, if dist($t_i$, $t_j$) $\leq D$:
            Increment $count_i$ by 1. If $count_i > M$, mark $t_i$ as a non-
            outlier and proceed to next $t_i$.
3. While blocks remain to be compared to the first array, do:
   a. Fill the second array with another block (but save a block which
      has never served as the first array, for last).
   b. For each unmarked tuple $t_i$ in the first array do:
            For each tuple $t_j$ in the second array, if dist($t_i$, $t_j$) $\leq D$:
                Increment $count_i$ by 1. If $count_i > M$, mark $t_i$ as a
                non-outlier and proceed to next $t_i$.
4. For each unmarked tuple $t_i$ in the first array, report $t_i$ as an outlier.
5. If the second array has served as the first array anytime before, stop;
   otherwise, swap the names of the first and second arrays and goto step
   2.
```

The FindAllOutsM algorithm

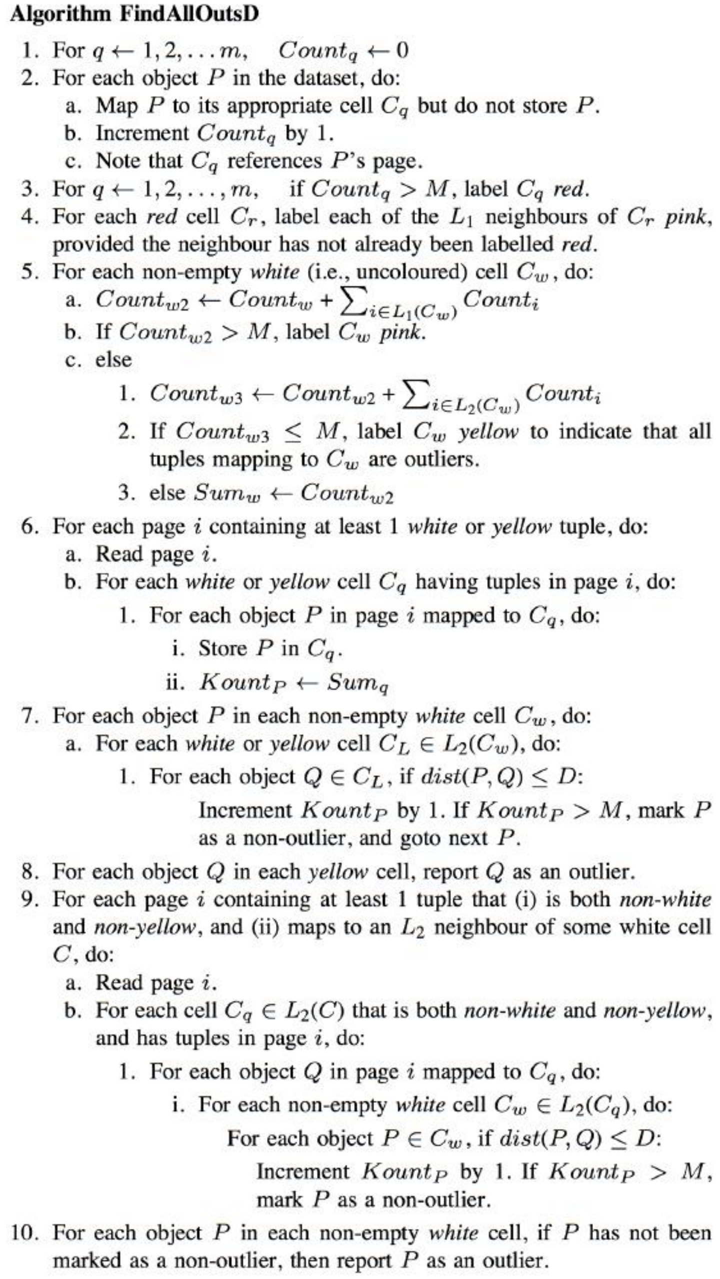
The following are the summarized pseudocode of the FindAllOutsM algorithm:



```
Algorithm FindAllOutsM
1. For $q \leftarrow 1, 2, \ldots m$,    $Count_q \leftarrow 0$
2. For each object $P$, map $P$ to an appropriate cell $C_q$, store $P$, and increment $Count_q$ by 1.
3. For $q \leftarrow 1, 2, \ldots, m$,    if $Count_q > M$, label $C_q$ red.
4. For each red cell $C_r$, label each of the $L_1$ neighbours of $C_r$ pink, provided the neighbour has not already been labelled red.
5. For each non-empty white (i.e., uncoloured) cell $C_w$, do:
   a. $Count_{w2} \leftarrow Count_w + \sum_{i \in L_1(C_w)} Count_i$
   b. If $Count_{w2} > M$, label $C_w$ pink.
   c. else
      1. $Count_{w3} \leftarrow Count_{w2} + \sum_{i \in L_2(C_w)} Count_i$
      2. If $Count_{w3} \leq M$, mark all objects in $C_w$ as outliers.
      3. else for each object $P \in C_w$, do:
         i. $Count_P \leftarrow Count_{w2}$
         ii. For each object $Q \in L_2(C_w)$, if $dist(P, Q) \leq D$:
               Increment $Count_P$ by 1. If $Count_P > M$, $P$ cannot be an outlier, so goto 5(c)(3).
         iii. Mark $P$ as an outlier.
```

The FindAllOutsD algorithm

The summarized pseudocodes of the FindAllOutsD algorithm are as follows:

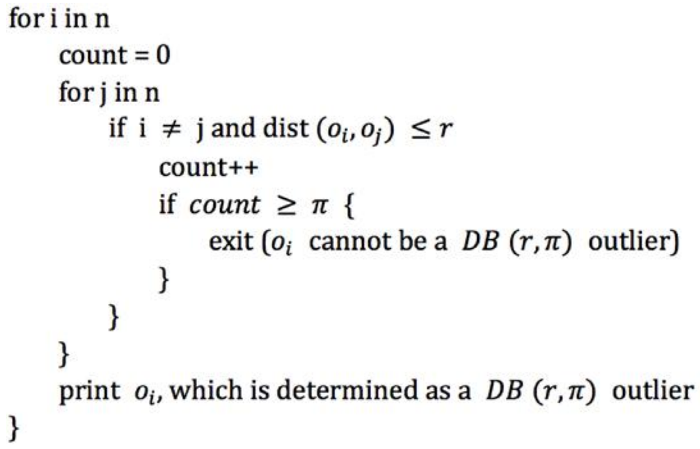


Algorithm FindAllOutsD

```
1. For $q \leftarrow 1, 2, \ldots m$,    $Count_q \leftarrow 0$
2. For each object $P$ in the dataset, do:
   a. Map $P$ to its appropriate cell $C_q$ but do not store $P$.
   b. Increment $Count_q$ by 1.
   c. Note that $C_q$ references $P$'s page.
3. For $q \leftarrow 1, 2, \ldots, m$,    if $Count_q > M$, label $C_q$ *red*.
4. For each *red* cell $C_r$, label each of the $L_1$ neighbours of $C_r$ *pink*,
   provided the neighbour has not already been labelled *red*.
5. For each non-empty *white* (i.e., uncoloured) cell $C_w$, do:
   a. $Count_{w2} \leftarrow Count_w + \sum_{i \in L_1(C_w)} Count_i$
   b. If $Count_{w2} > M$, label $C_w$ *pink*.
   c. else
         1. $Count_{w3} \leftarrow Count_{w2} + \sum_{i \in L_2(C_w)} Count_i$
         2. If $Count_{w3} \leq M$, label $C_w$ *yellow* to indicate that all
            tuples mapping to $C_w$ are outliers.
         3. else $Sum_w \leftarrow Count_{w2}$
6. For each page $i$ containing at least 1 *white* or *yellow* tuple, do:
   a. Read page $i$.
   b. For each *white* or *yellow* cell $C_q$ having tuples in page $i$, do:
         1. For each object $P$ in page $i$ mapped to $C_q$, do:
               i. Store $P$ in $C_q$.
               ii. $Kount_P \leftarrow Sum_q$
7. For each object $P$ in each non-empty *white* cell $C_w$, do:
   a. For each *white* or *yellow* cell $C_L \in L_2(C_w)$, do:
         1. For each object $Q \in C_L$, if $dist(P, Q) \leq D$:
               Increment $Kount_P$ by 1. If $Kount_P > M$, mark $P$
               as a non-outlier, and goto next $P$.
8. For each object $Q$ in each *yellow* cell, report $Q$ as an outlier.
9. For each page $i$ containing at least 1 tuple that (i) is both *non-white*
   and *non-yellow*, and (ii) maps to an $L_2$ neighbour of some white cell
   $C$, do:
   a. Read page $i$.
   b. For each cell $C_q \in L_2(C)$ that is both *non-white* and *non-yellow*,
      and has tuples in page $i$, do:
         1. For each object $Q$ in page $i$ mapped to $C_q$, do:
               i. For each non-empty *white* cell $C_w \in L_2(C_q)$, do:
                     For each object $P \in C_w$, if $dist(P, Q) \leq D$:
                           Increment $Kount_P$ by 1. If $Kount_P > M$,
                           mark $P$ as a non-outlier.
10. For each object $P$ in each non-empty *white* cell, if $P$ has not been
    marked as a non-outlier, then report $P$ as an outlier.
```

The distance-based algorithm

The summarized pseudocodes of the distance-based outlier detection algorithm are as follows, given a dataset *D*, size of the input dataset *n*, threshold *r* (`r > 0`), and $\pi \in (0,1]$:

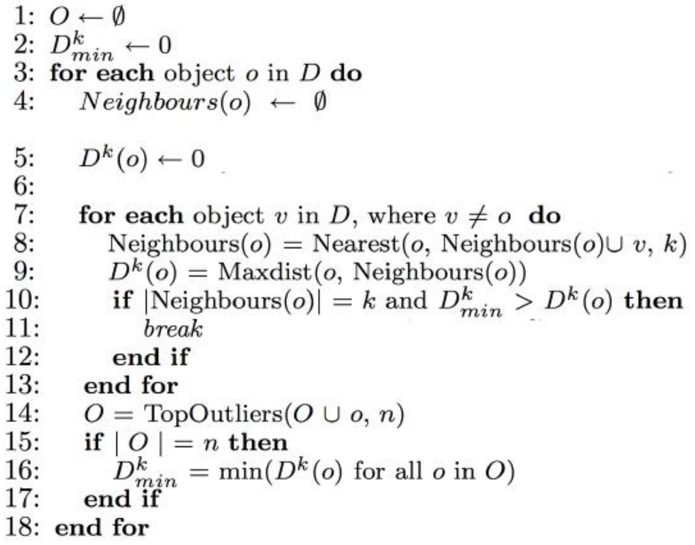


```
for i in n
    count = 0
    for j in n
        if i $\neq$ j and dist $(o_i, o_j)$ $\leq r$
            count++
            if $count$ $\geq$ $\pi$ {
                exit ($o_i$ cannot be a $DB$ $(r, \pi)$ outlier)
            }
        }
    }
    print $o_i$, which is determined as a $DB$ $(r, \pi)$ outlier
}
```

A $DB(r, \pi)$ outlier is defined as a data point, `o`, and subjected to this formula:

$$\left\| \{ o' \middle| dist(o, o') \leq r \} \right\| / \left\| D \right\| \leq \pi$$

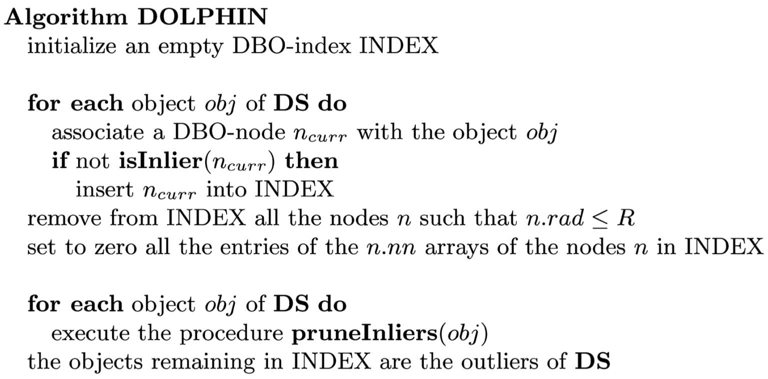
Let's now learn the pseudocodes for a variety of distance-based outlier detection algorithms, which are summarized in the following list. The input parameters are `k`, `n`, and `D`, which represent the neighbors' number, outlier number to be identified, and input dataset, respectively. A few supporter functions also are defined. `Nearest (o, S, k)` returns `k` nearest objects in `S` to `o`, `Maxdist (o, S)` returns the maximum distance between `o` and points from `S`, and `TopOutlier (S, n)` returns the top *n* outliers in `S` according to the distance to their *k*th nearest neighbor.



```
$O \leftarrow \emptyset$
$D^k_{min} \leftarrow 0$
for each object $o$ in $D$ do
    $Neighbours(o) \ \leftarrow \ \emptyset$

    $D^k(o) \leftarrow 0$

    for each object $v$ in $D$, where $v \neq o$ do
        Neighbours($o$) = Nearest($o$, Neighbours($o$)$\cup$ $v$, $k$)
        $D^k(o)$ = Maxdist($o$, Neighbours($o$))
        if |Neighbours($o$)| = $k$ and $D^k_{min} > D^k(o)$ then
            $break$
        end if
    end for
    $O$ = TopOutliers($O \cup o$, $n$)
    if | $O$ | = $n$ then
        $D^k_{min}$ = min($D^k(o)$ for all $o$ in $O$)
    end if
end for
```

The Dolphin algorithm

The Dolphin algorithm is a distance-based outlier detection algorithm. The summarized pseudocodes of this algorithm are listed as follows:



```
Algorithm DOLPHIN
  initialize an empty DBO-index INDEX

  for each object $obj$ of DS do
    associate a DBO-node $n_{curr}$ with the object $obj$
    if not isInlier($n_{curr}$) then
      insert $n_{curr}$ into INDEX
  remove from INDEX all the nodes $n$ such that $n.rad \leq R$
  set to zero all the entries of the $n.nn$ arrays of the nodes $n$ in INDEX

  for each object $obj$ of DS do
    execute the procedure pruneInliers($obj$)
  the objects remaining in INDEX are the outliers of DS
```

```
Function isInlier($n_{curr}$)

  perform a range query search in INDEX with center $n_{curr}.obj$ and radius $R$
  for each node $n_{index}$ returned by the range query do
    $dst = \text{dist}(n_{curr}.obj, n_{index}.obj)$
    if $dst \le R - n_{index}.rad$ then
      stop the search and report $n_{curr}.obj$ as an inlier [PR1] (return TRUE)
    if $dst \le R$ then
      $oldrad = n_{index}.rad$
      update the array $n_{index}.nn$ with the distance $dst$
      if $oldrad > R$ and $n_{index}.rad \le R$ then
        remove, with probability $1 - p_{inliers}$, the node $n_{index}$ from INDEX [PR2]
      update the array $n_{curr}.nn$ with the distance $dst$
      if $n_{curr}.rad \le R$ then
        stop the search and report $n_{curr}.obj$ as an inlier [PR3] (return TRUE)
  report $n_{curr}.obj$ as not an inlier (return FALSE)
```

```
Procedure pruneInliers($obj$)

  perform a range query search in INDEX with center $obj$ and radius $R$
  for each node $n_{index}$ returned by the range query do
    if $\text{dist}(obj, n_{index}.obj) \le R$ then
      update the array $n_{index}.nn$ with $obj$
      if $n_{index}.rad \le R$ then
        delete $n_{index}$ from INDEX
```

The R implementation

Look up the file of R codes, `ch_07_proximity_based.R`, from the bundle of R codes for the preceding algorithms. The codes can be tested with the following command:

```
> source("ch_07_proximity_based.R")
```

Activity monitoring and the detection of mobile fraud

The purpose of outlier detection is to find the patterns in source datasets that do not conform to the standard behavior. The dataset here consists of the calling records, and the patterns exist in the calling records.

There are many special algorithms developed for each specific domain. Misuse of a mobile is termed as mobile fraud. The subject under research is the calling activity or call records. The related attributes include, but are not limited to, call duration, calling city, call day, and various services' ratios.

Intrusion detection and density-based methods

Here is a formal definition of outliers formalized based on concepts such as, LOF, LRD, and so on. Generally speaking, an outlier is a data point biased from others so much that it seems as if it has not been generated from the same distribution functions as others have been.

Given a dataset, `D`, a DB (x, y)-outlier, `p`, is defined like this:

$$\left|\{q \in D | d\left(p,q\right) \leq y\}\right| \leq x$$

The k-distance of the `p` data point denotes the distance between `p` and the data point, `o`, which is member of `D`:

$$dist_k\left(p\right) denotes\, k - distance\, of\ data\ point\ p$$

$$\left|\{o' \in D \setminus \left\{p\right\} | d\left(p,o'\right) \leq d\left(p,o\right)\}\right| \geq k$$

$$\left|\{o' \in D \setminus \left\{p\right\} | d\left(p,o'\right) < d\left(p,o\right)\}\right| \leq k - 1$$

The k-distance neighborhood of the `p` object is defined as follows, `q` being the k-Nearest Neighbor of `p`:

$$N\left(p\right) = \{q \in D \setminus \left\{p\right\} | d\left(p,q\right) \leq dist_k\left(p\right)\}$$

The following formula gives the reachability distance of an object, `p`, with respect to an object, `o`:

$$reachdist_k\left(p,o\right) = \max\left\{dist_k\left(o\right), d\left(p,o\right)\right\}$$

The **Local Reachability Density** (**LRD**) of a data object, `o`, is defined like this:

$$lrd_{MinPts}(p) = 1 / \left(\frac{\sum_{o \in N_{MinPts}(\mathrm{p})} reach\text{-}dist_{MinPts}(p,o)}{\left| N_{MinPts}(p) \right|} \right)$$

The **Local Outlier Factor** (**LOF**) is defined as follows, and it measures the degree of the outlierness:

$$LOF_{MinPts}(p) = \frac{\sum_{o \in N_{MinPts}(\mathrm{p})} \frac{lrd_{MinPts}(o)}{lrd_{MinPts}(p)}}{\left| N_{MinPts}(p) \right|}$$

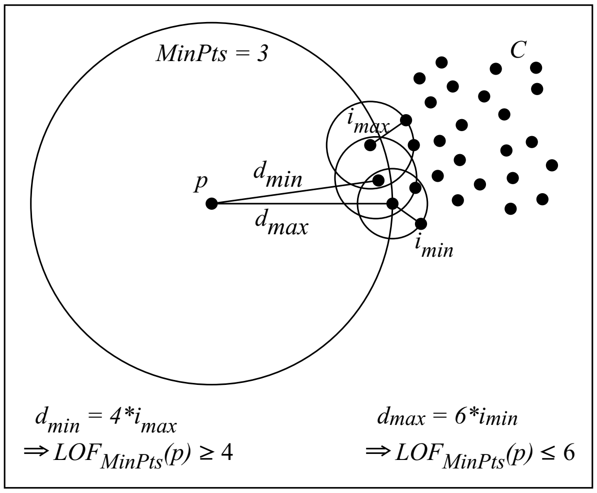
A property of `LOF (p)` is defined as shown in the following equation:

$$\frac{direct_{min}(p)}{indirect_{max}(p)} \leq LOF(p) \leq \frac{direct_{max}(p)}{indirect_{min}(p)}$$

$$direct_{min}(p) = \min\{reachdist(p,q) \mid q \in N_{MinPts}(p)\}$$

$$indirect_{min}(p) = \min\{reachdist(q,o) \mid q \in N_{MinPts}(p), o \in N_{MinPts}(q)\}$$

These equations are illustrated as follows:



The OPTICS-OF algorithm

The input parameters for the bagging algorithm are:

- $D(A_1, \ldots, A_m)$, the dataset
- α, the parameter
- β, another parameter

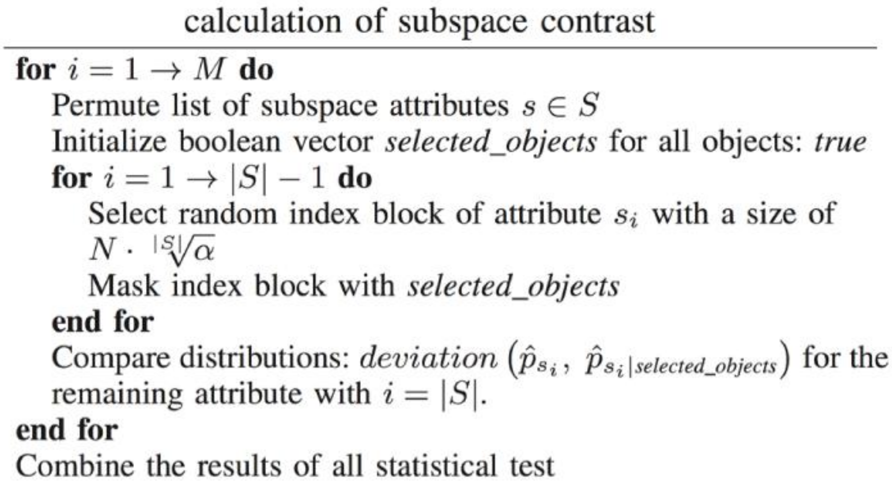
The output of the algorithm is the value of CBLOF, for all records.

The summarized pseudocodes of the OPTICS-OF algorithm are as follows:

```
$OPTICS - OF\ (D, K, S)$ {
  clustering the dataset $D(A_1, ..., A_m)$ by squeezer algorithm;
the prodused clusters are $C = \{C_1, ..., C_K\}$, and $|C_1| \geq ... \geq |C_k|$
   get LC and SC with the two parameters;
   for(each record t in the dataset D){
      if $(\mathrm{t} \in C_i$ and $C_i \in SC)$ {
         $\mathrm{CBLOF} = |C_i| * \min(\mathrm{distance}(t, C_i)), C_j \in LC$;
      } else {
         $\mathrm{CBLOF} = |C_i| * \mathrm{distance}(\mathrm{t}, \mathrm{C}_i), C_i \in LC$; ;
      }
      return CBLOF
  }
}
```

The High Contrast Subspace algorithm

The summarized pseudocodes of the **High Contrast Subspace** (**HiCS**) algorithm are as follows, where the input parameters are `S`, `M`, and ∝. The output is a contrast, `|S|`.



calculation of subspace contrast

```
for $i = 1 \rightarrow M$ do
   Permute list of subspace attributes $s \in S$
   Initialize boolean vector selected_objects for all objects: true
   for $i = 1 \rightarrow |S| - 1$ do
      Select random index block of attribute $s_i$ with a size of
      $N \cdot \sqrt[|S|]{\alpha}$
      Mask index block with selected_objects
   end for
   Compare distributions: $deviation\left(\hat{p}_{s_i}, \hat{p}_{s_i|selected\_objects}\right)$ for the
   remaining attribute with $i = |S|$.
end for
Combine the results of all statistical test
```

The R implementation

Look up the file of R codes, `ch_07_ density _based.R`, from the bundle of R codes for the previously mentioned algorithms. The codes can be tested using the following command:

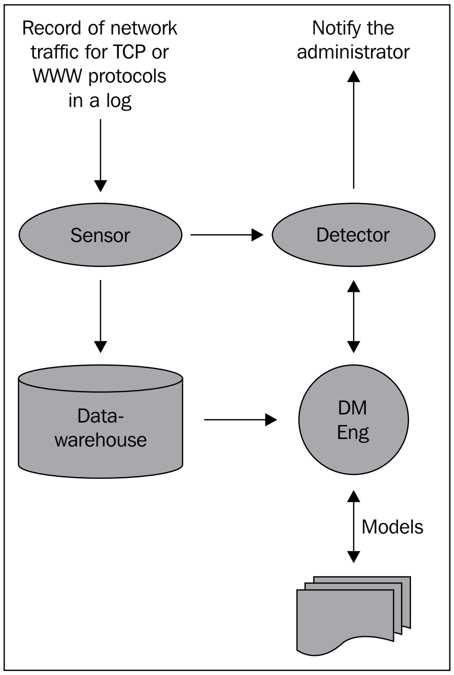
```
> source("ch_07_ density _based.R")
```

Intrusion detection

Any malicious activity against systems, networks, and servers can be treated as intrusion, and finding such activities is called **intrusion detection**.

The characteristics of situations where you can detect intrusion are high volume of data, missing labeled data in the dataset (which can be training data for some specific solution), time series data, and false alarm rate in the input dataset.

An intrusion detection system is of two types: host-based and network-based intrusion detection systems. A popular architecture for intrusion detection based on data mining is illustrated in the following diagram:



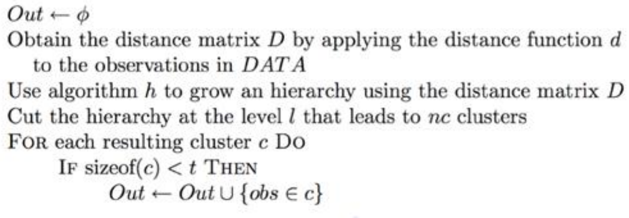
The core algorithms applied in an outlier detection system are usually semi-supervised or unsupervised according to the characteristics of intrusion detection.

Intrusion detection and clustering-based methods

The strategy of outlier detection technologies based on the clustering algorithm is focused on the relation between data objects and clusters.

Hierarchical clustering to detect outliers

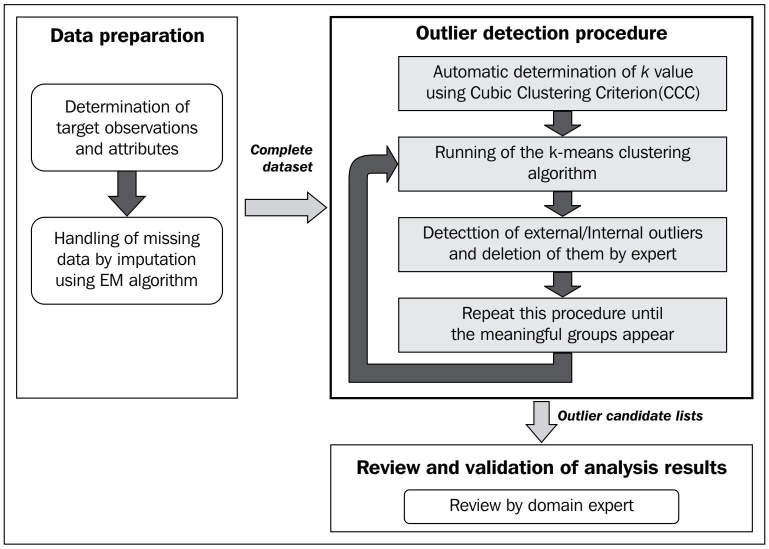
Outlier detection that uses the hierarchical clustering algorithm is based on the k-Nearest Neighbor graph. The input parameters include the input dataset, `DATA`, of size, `n`, and each data point with `k` variables, the distance measure function (`d`), one hierarchical algorithm (`h`), threshold (`t`), and cluster number (`nc`).



```
$Out \leftarrow \phi$
Obtain the distance matrix $D$ by applying the distance function $d$
   to the observations in $DATA$
Use algorithm $h$ to grow an hierarchy using the distance matrix $D$
Cut the hierarchy at the level $l$ that leads to $nc$ clusters
FOR each resulting cluster $c$ DO
      IF sizeof($c$) $< t$ THEN
            $Out \leftarrow Out \cup \{obs \in c\}$
```

The k-means-based algorithm

The process of the outlier detection based on the k-means algorithm is illustrated in the following diagram:



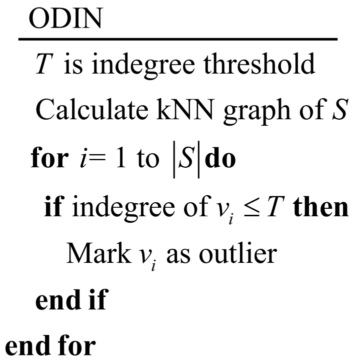
Data preparation
Determination of target observations and attributes
Handling of missing data by imputation using EM algorithm
Complete dataset
Outlier detection procedure
Automatic determination of *k* value using Cubic Clustering Criterion(CCC)
Running of the k-means clustering algorithm
Detecttion of external/Internal outliers and deletion of them by expert
Repeat this procedure until the meaningful groups appear
Outlier candidate lists
Review and validation of analysis results
Review by domain expert

The summarized pseudocodes of outlier detection using the k-means algorithm are as follows:

- **Phase 1** (**Data preparation**):
    1. The target observations and attributes should be aligned to improve the accuracy of the result and performance of the k-means clustering algorithm.
    2. If the original dataset has missing data, its handling activity must be carried out. The data of maximum likelihood that is anticipated by the EM algorithm is fed as input into the missing data.
- **Phase 2** (**The outlier detection procedure**):
    1. The *k* value should be determined in order to run the k-means clustering algorithm. The proper *k* value is decided by referring to the value of the **Cubic Clustering Criterion**.
    2. The k-means clustering algorithm runs with the decided *k* value. On completion, the expert checks the external and internal outliers in the clustering results. If the other groups' elimination of the outliers is more meaningful, then he/she stops this procedure. If the other groups need to be recalculated, he/she again runs the k-means clustering algorithms but without the detected outliers.
- **Phase 3** (**Review and validation**):
    1. The result of the previous phase is only a candidate for this phase. By considering the domain knowledge, we can find the true outliers.

The ODIN algorithm

Outlier detection using the indegree number with the ODIN algorithm is based on the k-Nearest Neighbor graph.



```
ODIN
$T$ is indegree threshold
Calculate kNN graph of $S$
for $i = 1$ to $|S|$ do
    if indegree of $v_i \leq T$ then
        Mark $v_i$ as outlier
    end if
end for
```

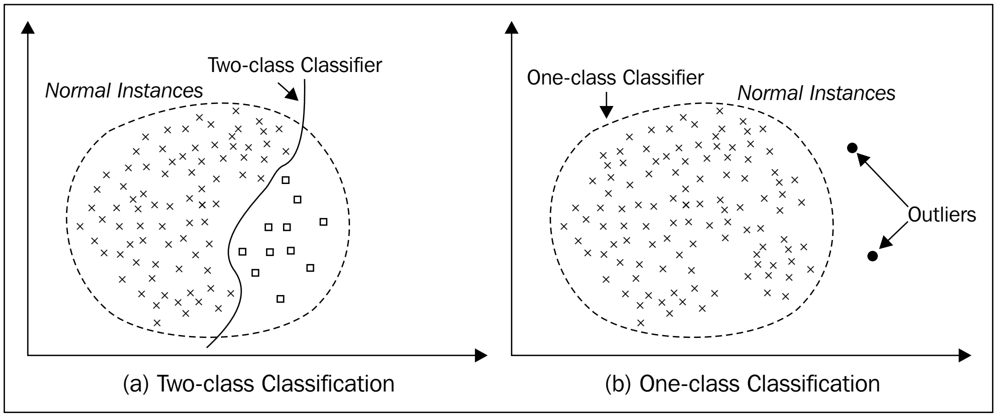
The R implementation

Look up the file of R codes, `ch_07_ clustering _based.R`, from the bundle of R codes for the previously mentioned algorithms. The codes can be tested with the following command:

```
> source("ch_07_ clustering _based.R")
```

Monitoring the performance of the web server and classification-based methods

Classification algorithms can be used to detect outliers. The ordinary strategy is to train a one-class model only for the normal data point in the training dataset. Once you set up the model, any data point that is not accepted by the model is marked as an outlier.



(a) Two-class Classification (b) One-class Classification

The OCSVM algorithm

The **OCSVM** (**One Class SVM**) algorithm projects input data into a high-dimensional feature space. Along with this process, it iteratively finds the maximum-margin hyperplane. The **hyperplane** defined in a **Gaussian reproducing kernel Hilbert** space best separates the training data from the origin. When $v \in [0,1]$, the major portion of outliers or the solution of OCSVM can be represented by the solution of the following equation (subject to $\sum_{k=1}^{n} w_k = 1$ and $0 \le w_1, ..., w_n \le \frac{1}{vn}$):

$$\min_{\{w_k\}_{k=1}^{n}} \frac{1}{2} \sum_{k,k'=1}^{n} w_k w_{k'} K_\sigma(\boldsymbol{x}_k, \boldsymbol{x}_{k'})$$

Initialization of the algorithm. We start by setting a random fraction ν of all α_i to $1/(\nu\ell)$. If $\nu\ell$ is not an integer, then one of the examples is set to a value in $(0, 1/(\nu\ell))$ to ensure that $\sum_i \alpha_i = 1$. Moreover, we set the initial ρ to $\max\{O_i : i \in [\ell], \alpha_i > 0\}$.

Optimization algorithm. We then select a first variable for the elementry optimization step in one of the two following ways. Here, we use the shorthand SV_{nb} for the indices of variables which are not at bound, that is, $SV_{nb} := \{i : i \in [\ell], 0 < \alpha_i < 1/(\nu\ell)\}$. At the end, these correspond to points that will sit exactly on the hyperplane, and that will therefore have a strong influence on its precise position.

(i) We scan over the entire data set[1] until we find a variable violating a KKT condition(Bertsekas,1995, e.g.), that is, a point such that $(O_i - \rho).\alpha_i > 0$ or $(\rho - O_i).(1/(\nu\ell)) - \alpha_i) > 0$. Once we have found one,say α_i, we pick α_j according to

$$j = \arg\max_{n\in\, sv_{nb}} |O_i - O_n|.$$

(ii) Same as (i), but the scan is only performed over sv_{nb}

The one-class nearest neighbor algorithm

This algorithm is based on the k-Nearest Neighbor algorithm. A couple of formulas are added.

The local density is denoted as follows:

$$P_n(x) = k_n / N / V_n$$

The distance between the test object, x, and its nearest neighbor in the training set, $NN^{tr}(x)$, is defined like this:

$$d_1 = \left\| x - NN^{tr}(x) \right\|$$

The distance between this nearest neighbor ($NN^{tr}(x)$) and its nearest neighbor in the training set ($NN^{tr}\left(NN^{tr}(x)\right)$) is defined as follows:

$$d_2 = \left\| NN^{tr}(x) - NN^{tr}\left(NN^{tr}(x)\right) \right\|$$

One data object is marked as an outlier once $d_1 >> d_2$, or, in another format, is marked as $\rho_{NN}(x) = d_1 / d_2$.

The R implementation

Look up the file of R codes, `ch_07_ classification _based.R`, from the bundle of R codes for previously mentioned algorithms. The codes can be tested with the following command:

```
> source("ch_07_ classification _based.R")
```

Monitoring the performance of the web server

Web server performance measurements are really important to the business and for operating system management. These measurements can be in the form of CPU usage, network bandwidth, storage, and so on.

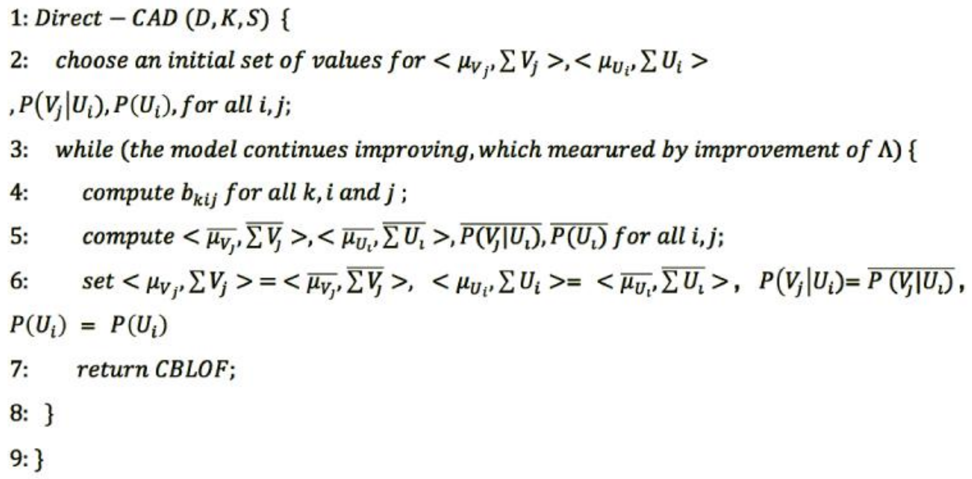
The dataset comes from various sources such as benchmark data, logs, and so on. The types of outliers that appear during the monitoring of the web server are point outliers, contextual outliers, and collective outliers.

Detecting novelty in text, topic detection, and mining contextual outliers

If each of the data instances in the training dataset is related to a specific context attribute, then the dramatic deviation of the data point from the context will be termed as an outlier. There are many applications of this assumption.

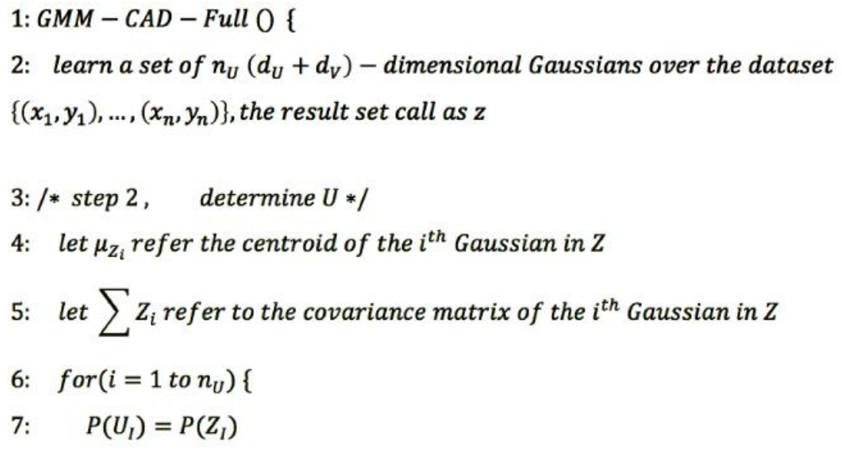
The conditional anomaly detection (CAD) algorithm

The summarized pseudocodes of the CAD algorithm are as follows:



```
$Direct - CAD\ (D, K, S)$ {
  $choose\ an\ initial\ set\ of\ values\ for < \mu_{V_j}, \sum V_j >, < \mu_{U_i}, \sum U_i >$
$, P(V_j|U_i), P(U_i), for\ all\ i, j;$
  $while\ (the\ model\ continues\ improving, which\ mearured\ by\ improvement\ of\ \Lambda)$ {
    $compute\ b_{kij}\ for\ all\ k, i\ and\ j\ ;$
    $compute < \overline{\mu_{V_j}}, \overline{\sum V_j} >, < \overline{\mu_{U_i}}, \overline{\sum U_i} >, \overline{P(V_j|U_i)}, \overline{P(U_i)}\ for\ all\ i, j;$
    $set < \mu_{V_j}, \sum V_j > = < \overline{\mu_{V_j}}, \overline{\sum V_j} >,\ < \mu_{U_i}, \sum U_i > = \ < \overline{\mu_{U_i}}, \overline{\sum U_i} >,\ P(V_j|U_i) = \overline{P\ (V_j|U_i)},$
$P(U_i)\ =\ P(U_i)$
    $return\ CBLOF;$
  }
}
```

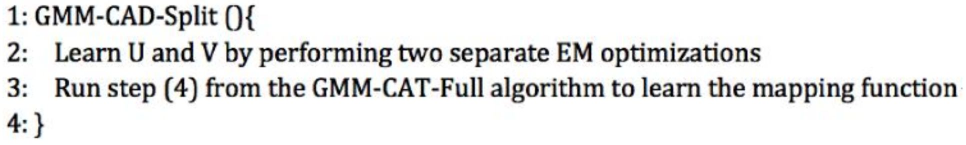
The following are the summarized pseudocodes of the GMM-CAD-Full algorithm:



```
$GMM - CAD - Full\ ()$ {
  $learn\ a\ set\ of\ n_U\ (d_U + d_V) - dimensional\ Gaussians\ over\ the\ dataset$
$\{(x_1, y_1), \dots, (x_n, y_n)\}, the\ result\ set\ call\ as\ z$

/* $step\ 2,\qquad determine\ U$ */
  $let\ \mu_{Z_i}\ refer\ the\ centroid\ of\ the\ i^{th}\ Gaussian\ in\ Z$
  $let\ \sum Z_i\ refer\ to\ the\ covariance\ matrix\ of\ the\ i^{th}\ Gaussian\ in\ Z$
  $for(i = 1\ to\ n_U)$ {
      $P(U_I) = P(Z_I)$
```

```
    $for(j = 1\ to\ d_U)\ \{$
        $\mu_{V_i}[j] = \mu_{Z_i}[j]$
        $for(k = 1\ to\ d_U)\ \{$
            $\sum V_i[j][k] = \sum Z_i[j][k]$
        $\}$
    $\}$
$\}$

/* step 3 ,        determine V */
$for(i = 1\ to\ n_v)\ \{$
    $for(j = 1\ to\ d_v)\ \{$
        $\mu_{V_i}[j] = \mu_{Z_i}[j + d_U]$
        $for(k = 1\ to\ d_v)\ \{$
            $\sum V_i[j][k] = \sum Z_i[j + d_U][k + d_U]$
        $\}$
    $\}$
$\}$

/* step 4 */
run a second EM algorithm to learn the mapping function.
  while (the model continues improving, which mearured by improvement of $\Lambda$) {
      compute $b_{kij}$ for all k, i and j  as in the previous sub section;
      compute $\overline{P(V_j|U_\imath)}$ as described in the previous subsection;
      set  $P(V_j|U_i)= \overline{P\ (V_j|U_\imath)}$
  }
```

The summarized pseudocodes of the GMM-CAD-Split algorithm are as follows:



```
GMM-CAD-Split (){
    Learn U and V by performing two separate EM optimizations
    Run step (4) from the GMM-CAT-Full algorithm to learn the mapping function
}
```

The R implementation

Look up the file of R codes, `ch_07_ contextual _based.R`, from the bundle of R codes for the previously mentioned algorithms. The codes can be tested with the following command:

```
> source("ch_07_ contextual _based.R")
```

Detecting novelty in text and topic detection

One application of outlier detection is in finding novel topics in a collection of documents or articles from newspapers. Major detection includes opinion detection. This is basically an outlier among a lot of opinions.

With the increase in social media, there are many events that happen every day. The earlier collection was that of only special events or ideas for related researchers or companies.

The characteristics related to increase in the collection are the various sources of data, documents in different formats, high-dimensional attributes, and sparse source data.

Collective outliers on spatial data

Given a dataset, if a collection of related data instances is anomalous with respect to the entire dataset, it is defined as a collective outlier.

The route outlier detection (ROD) algorithm

The summarized pseudocodes of the ROD algorithm are as follows. The input parameters include the multidimensional attribute space, attribute dataset (D), distance measure function (F), the depth of neighbor (ND), spatial graph (G = (V, E)), and confidence interval (CI):

```
for (i=1);i ≤ $|RN|$ ;i++){
    $O_i$ = Get_One_Object(i,D);
    NNS=Find_Neighbor_Nodes_set($O_i$, $ND$, $G$)
    $Accum\_Dist$=0;
    for(j=1; j ≤ |NSS|; j++){
        $O_k$ = Get_One_Object(j . NNS);
        Accum_Dist + = $F(O_i, O_k, S)$
    }
    AvgDist = Accum_Dist/ |NSS|;
    $T_{value} = \frac{AvgDist - \mu_s}{\sigma_s}$
    if (Check_Normal_Table($T_{value}$, $CI$) == True){
        Add_Element(Outlier_Set, i);
    }
}
return Outlier_Set
```

The R implementation

Look up the file of R codes, `ch_07_ rod.R`, from the bundle of R codes for the previously mentioned algorithm. The codes can be tested with the following command:

```
> source("ch_07_ rod.R")
```

Characteristics of collective outliers

Collective outliers denote a collection of data that is an abnormal contrast to the input dataset. As a major characteristic, only the collection of data appearing together will be collective outliers, but specific data itself in that collection does not appear together with other data in that collection of data, which is definitely not an outlier. Another characteristic of a collective outlier is that it can be a contextual outlier.

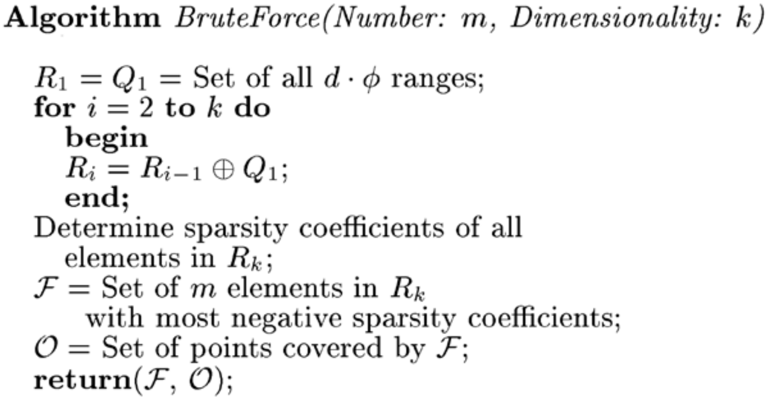
Collective outliers may be a sequence of data, spatial data, and so on.

Outlier detection in high-dimensional data

Outlier detection in high-dimensional data has some characteristics that make it different from other outlier detection problems.

The brute-force algorithm

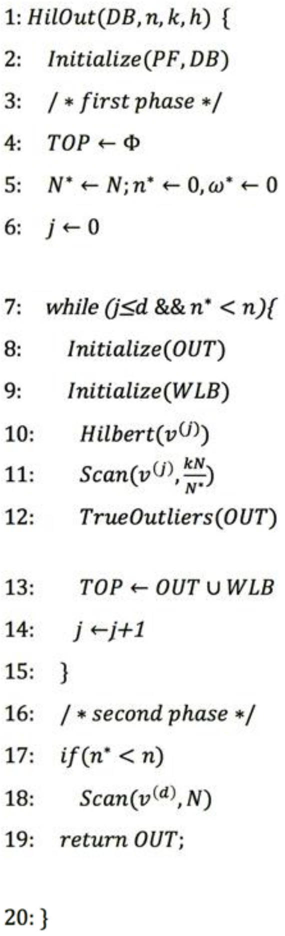
The summarized pseudocodes of the brute-force algorithm are as follows:



```
Algorithm BruteForce(Number: m, Dimensionality: k)

  $R_1 = Q_1 =$ Set of all $d \cdot \phi$ ranges;
  for $i = 2$ to $k$ do
    begin
    $R_i = R_{i-1} \oplus Q_1$;
    end;
  Determine sparsity coefficients of all
    elements in $R_k$;
  $\mathcal{F} =$ Set of $m$ elements in $R_k$
      with most negative sparsity coefficients;
  $\mathcal{O} =$ Set of points covered by $\mathcal{F}$;
  return($\mathcal{F}$, $\mathcal{O}$);
```

The HilOut algorithm

The following are the summarized pseudocodes of the HilOut algorithm:



```
$HilOut(DB, n, k, h)$ {
  $Initialize(PF, DB)$
  /* first phase */
  $TOP \leftarrow \Phi$
  $N^* \leftarrow N; n^* \leftarrow 0, \omega^* \leftarrow 0$
  $j \leftarrow 0$

  while ($j \leq d$ && $n^* < n$){
     $Initialize(OUT)$
     $Initialize(WLB)$
       $Hilbert(v^{(j)})$
       $Scan(v^{(j)}, \frac{kN}{N^*})$
       $TrueOutliers(OUT)$

       $TOP \leftarrow OUT \cup WLB$
      $j \leftarrow j+1$
   }
   /* second phase */
   $if (n^* < n)$
       $Scan(v^{(d)}, N)$
   return $OUT$;

}
```

The R implementation

Look up the R file, `hil_out.R`, from the bundle of R codes for the HilOut algorithm.

Look up the file of R codes, `ch_07_ hilout.R`, from the bundle of R codes for the previously mentioned algorithms. The codes can be tested with the following command:

```
> source("ch_07_ hilout.R")
```

Time for action

Here are some practice questions for you so as to check your understanding of the concepts:

- What is an outlier?
- List as many types of outliers as possible and your categorization measures.
- List as many areas of application of outlier detection as possible.

Summary

In this chapter, we looked at:

- Statistical methods based on probabilistic distribution functions. The normal data points are those that are generated by the models. Otherwise, they are defined as outliers.
- Proximity-based methods.
- Density-based methods.
- Clustering-based methods.
- Classification-based methods.
- Mining contextual outliers.
- Collective outliers.
- Outlier detection in high-dimensional data.

The next chapter will cover the major topics related to outlier detection algorithms and examples for them, which are based on the previous chapters. All of this will be covered with a major difference in our viewpoint.

8

Mining Stream, Time-series, and Sequence Data

In this chapter, you will learn how to write mining codes for stream data, time-series data, and sequence data.

The characteristics of stream, time-series, and sequence data are unique, that is, large and endless. It is too large to get an exact result; this means an approximate result will be achieved. The classic data-mining algorithm should be extended, or a new algorithm needs to be designed for this type of the dataset.

In relation to the mining of stream, time-series, and sequence data, there are some topics we can't avoid. They are association, frequent pattern, classification and clustering algorithms, and so on. In the following sections, we will go through these major topics.

In this chapter, we will cover the following topics;

- The credit card transaction flow and STREAM algorithm
- Predicting future prices and time-series analysis
- Stock market data and time-series clustering and classification
- Web click streams and mining symbolic sequences
- Mining sequence patterns in transactional databases

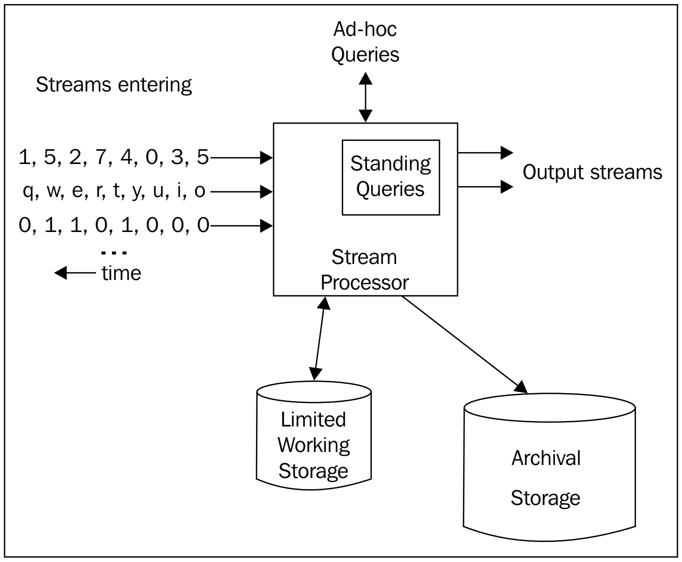
The credit card transaction flow and STREAM algorithm

As we mentioned in the previous chapters, one kind of data source always requires a variety of predefined algorithms or a brand new algorithm to deal with. Streaming data behaves a bit different from a traditional dataset.

The streaming dataset comes from various sources in modern life, such as credit record transaction stream, web feeds, phone-call records, sensor data from a satellite or radar, network traffic data, a security event's stream, and a long running list of various data streams.

The targets to stream data processing are, and not limited to, summarization of the stream to specific extents.

With the characteristics of streaming data, the typical architecture to stream a management system is illustrated in the following diagram:



The STREAM algorithm is a classical algorithm used to cluster stream data. In the next section, the details are present and explained by R code.

The STREAM algorithm

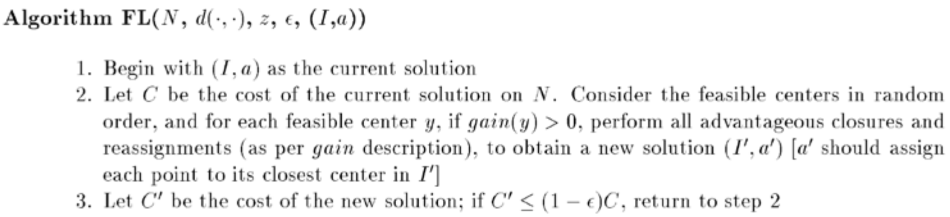
The summarized pseudocode of the STREAM algorithm are as follows:

```
STREAM
For each chunk $X_i$ in the stream
    1. If a sample of size $\geq \frac{1}{\epsilon} \log \frac{k}{\delta}$ contains fewer than k distinct points then:
        • $X_i \longleftarrow$ weighted representation
    2. Cluster $X_i$ using LOCALSEARCH
    3. $X' \longleftarrow ik$ centers obtained from chunks 1 through $i$ iterations of the stream, where each
       center $c$ obtained by clustering $X_i$ is weighted by the number of points in $X_i$ assigned to $c$.
    4. Output the $k$ centers obtained by clustering $X_i$ using LOCALSEARCH
```

In the preceding algorithm, **LOCALSEARCH** is a revised k-median algorithm.

```
LOCALSEARCH ($N$, $d(\cdot , \cdot)$, $k$, $\epsilon$, $\epsilon'$,$\cdot\epsilon''$)
    1. Read in $n$ data points.
    2. Set $z_{min}$=0
    3. Set $z_{max}=\sum_{x \epsilon N} d(x, x_0)$, where $x_0$ is an arbitrary point in $N$
    4. Set z to be $\frac{z_{max} + z_{min}}{2}$
    5. Obtain an initial solution $(I , a)$ using Algorithm InitialSolution $(N , z)$.
    6. Select $\Theta$ $(\frac{1}{p} \log k)$ random points to serve as feasible centers
    7. While more or fewer that k centers and $z_{min} < (1-\epsilon'')\ z_{max}$
        • Let $(F, g)$ be the current solution
        • Run $FL$ $(N, d, \epsilon, (F, g))$ to obtain a new solution $(F' , g')$
        • If $|F'|$ is "about" $k$, run $FL$ $(N, d, \epsilon, (F', g'))$ to obtain a new solution; reset $(F', g')$ to
          be this new solution
        • If $|F'| < k$ then set $z_{min}$ = z and z = $\frac{z_{max} + z_{min}}{2}$ ; else if $|F'| > k$ then set $z_{max}$ = z and
          z = $\frac{z_{max} + z_{min}}{2}$
    8. To Simulate a continuous space, move each cluster center to the center - of - mass for its cluster
    9. Return our solution $(F' , g')$
```

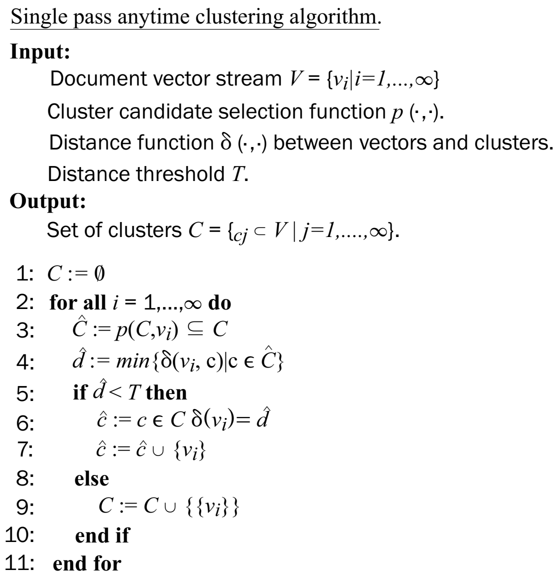
```
InitialSolution (data set $N$, facility cost z)
1. Reorder data points randomly
2. Create a cluster center at the first point
3. For every point after the first,
    • Let $d$ be the distance form the current data point to the nearest existing cluster center
    • With probability $d / z$ create a new cluster center at the current data point; otherwise
      add the current point to the best current cluster
```



```
Algorithm FL($N$, $d(\cdot,\cdot)$, $z$, $\epsilon$, $(I,a)$)

  1. Begin with $(I, a)$ as the current solution
  2. Let $C$ be the cost of the current solution on $N$. Consider the feasible centers in random
     order, and for each feasible center $y$, if $gain(y) > 0$, perform all advantageous closures and
     reassignments (as per $gain$ description), to obtain a new solution $(I', a')$ [$a'$ should assign
     each point to its closest center in $I'$]
  3. Let $C'$ be the cost of the new solution; if $C' \leq (1 - \epsilon)C$, return to step 2
```

The single-pass-any-time clustering algorithm

Here is a clustering algorithm designed to deal with a high-frequency news stream:



```
Single pass anytime clustering algorithm.
Input:
    Document vector stream $V = \{v_i | i=1,...,\infty\}$
    Cluster candidate selection function $p(\cdot,\cdot)$.
    Distance function $\delta(\cdot,\cdot)$ between vectors and clusters.
    Distance threshold $T$.
Output:
    Set of clusters $C = \{c_j \subset V \mid j=1,....,\infty\}$.

$C := \emptyset$
for all $i = 1,...,\infty$ do
    $\hat{C} := p(C,v_i) \subseteq C$
    $\hat{d} := min\{\delta(v_i, c) | c \in \hat{C}\}$
    if $\hat{d} < T$ then
        $\hat{c} := c \in C \; \delta(v_i) = \hat{d}$
        $\hat{c} := \hat{c} \cup \{v_i\}$
    else
        $C := C \cup \{\{v_i\}\}$
    end if
end for
```

The R implementation

Please take a look at the R codes file `ch_08_stream.R` from the bundle of R codes for previously mentioned algorithms. The codes can be tested with the following command:

```
> source("ch_08_stream.R")
```

The credit card transaction flow

Day by day, the growing e-commerce market is driving the growth of the usage of credit card, which, in turn, is bringing in a large number of transaction streams. Fraudulent usage of credit cards happens every day; we want the algorithm to detect this kind of transaction in a very short time compared to the big volume of transactions. Besides this, the requirement to find out the valuable customers by an analysis of the transaction streams is becoming more and more important. However, it is harder to get valid information in a very short time in response to the advertisement needs, such as recommend the necessary finance services or goods to these customers.

The credit card transaction flow is also the process to generate stream data, and the related stream-mining algorithm can be applied with high accuracy.

One application for the credit card transaction flow mining is the behavior analysis of the consumers. With each transaction record that is stored, various costs or purchases of a certain person can be tracked. With the mining of such transaction records, we can provide an analysis to the credit card holder to help them keep financial balance or other financial target. We can also provide this analysis to the card issuer that published the credit card to create a new business, such as funding or loaning, or to the retailers (the merchants) such as Walmart to help in the arrangement of appropriate goods.

Another application for the credit card transaction flow mining is fraud detection. It is the most obvious one among tons of applications. Using this solution, the card issuer or the bank can reduce the rate of successful fraud.

The dataset of the credit card transaction includes the owner of cards, place of consumption, date, cost, and so on.

Predicting future prices and time-series analysis

Auto Regressive Integrated Moving Average (**ARIMA**) is a classic algorithm to analyze time–series data. As the initial step, an ARIMA model will be chosen to model the time-series data. Assuming that p_t is a time-series variable such as a price, the formula is defined as follows, and it includes the main features of the variable:

$$\phi\left(B\right)pt=\theta\left(B\right)\varepsilon_t$$

$$\begin{aligned}\phi(B)\,pt = &\left(1-\phi_1 B^1 - \phi_2 B^2\right)\left(1-\phi_{24}B^{24} - \phi_{48}B^{48}\right)\\ &\times\left(1-\phi_{168}B^{168}\right)\left(1-B\right)\left(1-B^{24}\right)\end{aligned}$$

$$\begin{aligned}&\left(1-\phi_1 B^1 - \phi_2 B^2 - \phi_3 B^3 - \phi_4 B^4 - \phi_5 B^5\right)\\ &\times\left(1-\phi_{23}B^{23} - \phi_{24}B^{24} - \phi_{47}B^{47} - \phi_{48}B^{48}\right.\\ &\left.-\phi_{72}B^{72} - \phi_{96}B^{96} - \phi_{120}B^{120} - \phi_{144}B^{144}\right)\\ &\times\left(1-\phi_{168}B^{168} - \phi_{336}B^{336} - \phi_{504}B^{504}\right)\log pt\\ &= c + \left(1-\theta_1 B^1 - \theta_2 B^2\right)\left(1-\theta_{24}B^{24}\right)\\ &\quad\times\left(1-\theta_{168}B^{168} - \theta_{336}B^{336} - \theta_{504}B^{504}\right)\varepsilon_t\end{aligned}$$

$$\begin{aligned}&\left(1-\phi_1 B^1 - \phi_2 B^2\right)\\ &\times\left(1-\phi_{23}B^{23} - \phi_{24}B^{24} - \phi_{47}B^{47} - \phi_{48}B^{48}\right.\\ &\quad\left.-\phi_{72}B^{72} - \phi_{96}B^{96} - \phi_{120}B^{120} - \phi_{144}B^{144}\right)\\ &\left(1-\phi_{167}B^{167} - \phi_{168}B^{168} - \phi_{169}B^{169} - \phi_{192}B^{192}\right)\\ &\times\left(1-B\right)\left(1-B^{24}\right)\left(1-B^{168}\right)\log pt\\ &\quad= c + \left(1-\theta_1 B^1 - \theta_2 B^2\right)\\ &\qquad\times\left(1-\theta_{24}B^{24} - \theta_{48}B^{48} - \theta_{72}B^{72} - \theta_{96}B^{96}\right)\\ &\qquad\times\left(1-\theta_{144}B^{144}\right)\\ &\qquad\times\left(1-\theta_{168}B^{168} - \theta_{336}B^{336} - \theta_{504}B^{504}\right)\varepsilon_t\end{aligned}$$

The ARIMA algorithm

The summarized steps for the ARIMA algorithm is as follows:

1. A class of models is formulated assuming certain hypotheses.
2. A model is identified for the observed data.

3. The model parameters are estimated.
4. If the hypotheses of the model are validated, proceed with this step; otherwise, go to step 1 to refine the model. The model is now ready for forecasting.

Predicting future prices

Predicting future prices is one subproblem of predicting the future; this is just a sign of the difficulty of this issue. Another similar issue under this domain is estimating the future market demands.

The stock market is a good example of predicting the future price; here, the prices changes along with time. Predicting future prices helps estimate the equity returns in the future and also helps in deciding the timing for financial investors, such as when to buy/sell.

The price graph shows an oscillation. Many factors can affect this value, even the psycology of humans.

The key problem in this prediction is the huge data volume. As a direct result, the algorithms to be used for this topic need to be efficient. The other key problem is that the price might change dramatically in a short time. Also, the complexity of the market is definitely a big problem.

The data instances for the future price prediction include quantitative attributes such as technical factors, and they also include macroeconomics, microeconomics, political events, and investor expectations. Moreover, they include the domain expert's knowledge.

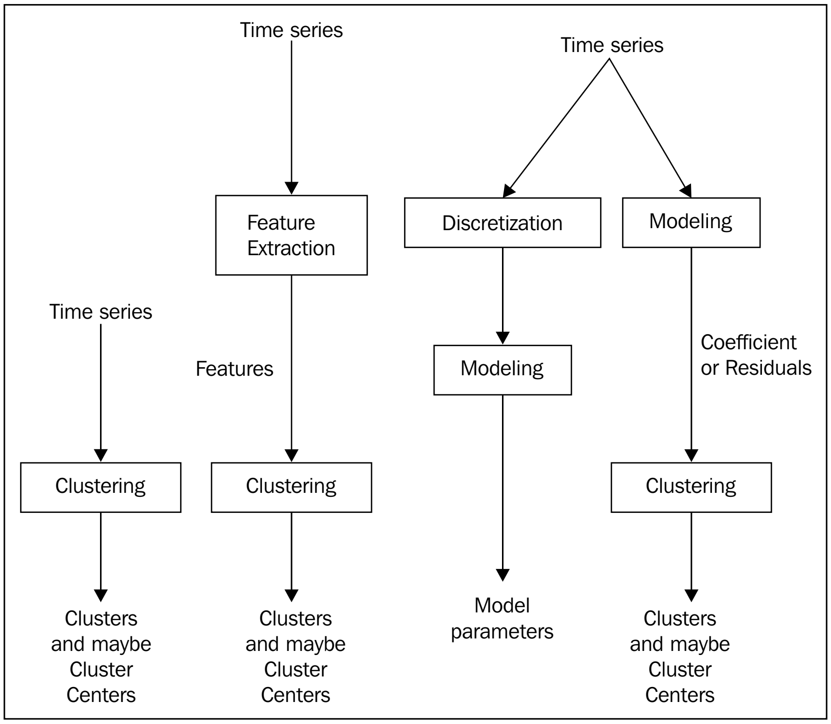
All the solutions based on the aggregate disperse information from the preselected factors.

Price is obviously affected by time, and is a time-series variable too. To forecast the future value, the time-series-analysis algorithm will apply. ARIMA can be used to forecast the prices of the next day.

One classical solution is to determine the trends of the market at a very early stage. Another solution is fuzzy solutions, which is also a good choice because of the huge volume of dataset and wide range of factors affecting the prices.

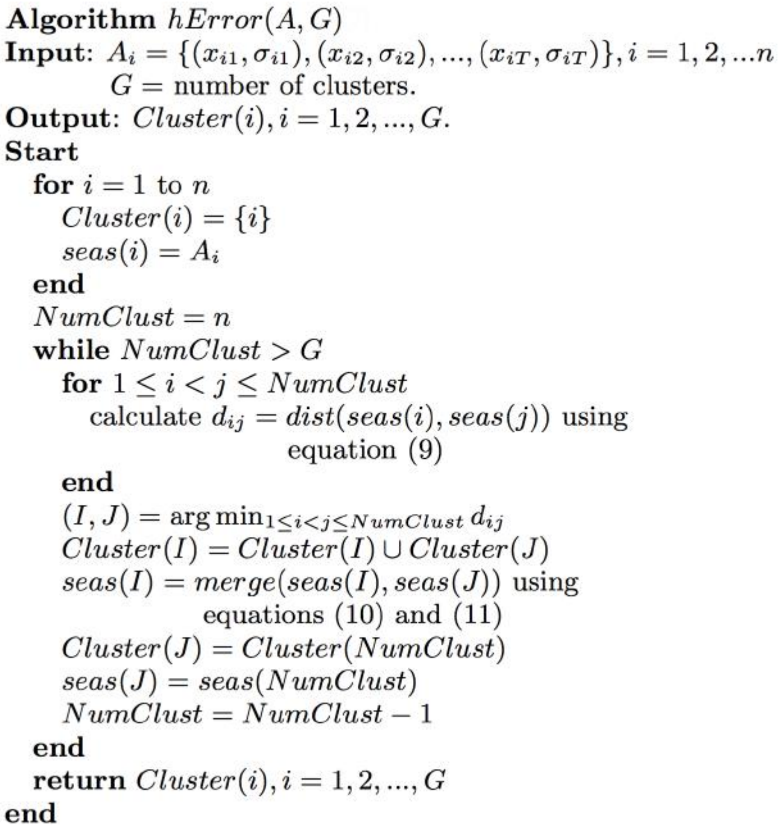
Stock market data and time-series clustering and classification

Time-series clustering has been proven to provide effective information for further research. In contrast to the classic clustering, the time-series dataset comprises data changed with time.



The hError algorithm

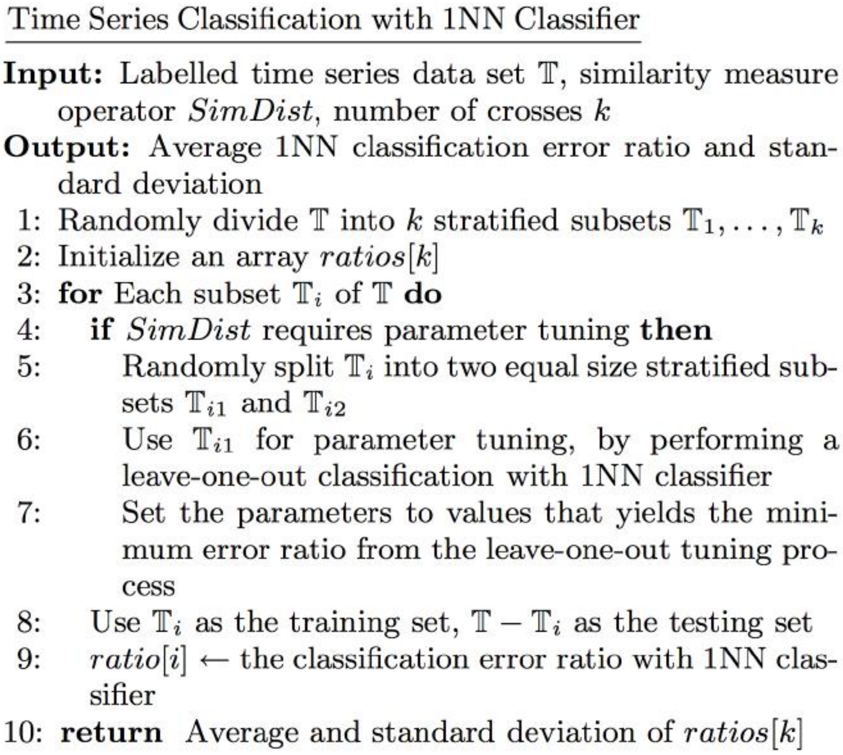
This algorithm is denoted as **"clustering seasonality patterns in the presence of errors**"; the summarized algorithm is listed in the following figure. The major characteristic of this algorithm is to introduce a specific distance function and a dissimilarity function.



```
Algorithm $hError(A, G)$
Input: $A_i = \{(x_{i1}, \sigma_{i1}), (x_{i2}, \sigma_{i2}), ..., (x_{iT}, \sigma_{iT})\}, i = 1, 2, ...n$
       $G$ = number of clusters.
Output: $Cluster(i), i = 1, 2, ..., G$.
Start
  for $i = 1$ to $n$
    $Cluster(i) = \{i\}$
    $seas(i) = A_i$
  end
  $NumClust = n$
  while $NumClust > G$
    for $1 \leq i < j \leq NumClust$
      calculate $d_{ij} = dist(seas(i), seas(j))$ using
                 equation (9)
    end
    $(I, J) = \arg\min_{1 \leq i < j \leq NumClust} d_{ij}$
    $Cluster(I) = Cluster(I) \cup Cluster(J)$
    $seas(I) = merge(seas(I), seas(J))$ using
           equations (10) and (11)
    $Cluster(J) = Cluster(NumClust)$
    $seas(J) = seas(NumClust)$
    $NumClust = NumClust - 1$
  end
  return $Cluster(i), i = 1, 2, ..., G$
end
```

Time-series classification with the 1NN classifier

The summarized pseudocode for the 1NN classifier algorithm is as follows:



```
Time Series Classification with 1NN Classifier

Input: Labelled time series data set $\mathbb{T}$, similarity measure operator $SimDist$, number of crosses $k$
Output: Average 1NN classification error ratio and standard deviation
Randomly divide $\mathbb{T}$ into $k$ stratified subsets $\mathbb{T}_1, \ldots, \mathbb{T}_k$
Initialize an array $ratios[k]$
for Each subset $\mathbb{T}_i$ of $\mathbb{T}$ do
    if $SimDist$ requires parameter tuning then
        Randomly split $\mathbb{T}_i$ into two equal size stratified subsets $\mathbb{T}_{i1}$ and $\mathbb{T}_{i2}$
        Use $\mathbb{T}_{i1}$ for parameter tuning, by performing a leave-one-out classification with 1NN classifier
        Set the parameters to values that yields the minimum error ratio from the leave-one-out tuning process
    Use $\mathbb{T}_i$ as the training set, $\mathbb{T} - \mathbb{T}_i$ as the testing set
    $ratio[i] \leftarrow$ the classification error ratio with 1NN classifier
return Average and standard deviation of $ratios[k]$
```

The R implementation

Please take a look at the R codes file `ch_08_herror.R` from the bundle of R codes for previously mentioned algorithms. The codes can be tested with the following command:

```
> source("ch_08_herror.R")
```

Stock market data

Stock market is a complex dynamic system, and many factors can affect the market. For example, breaking financial news is also a key factor for this topic.

The characteristics of stock market data are large volume (near infinite), real time, high dimensions, and high complexity. The various data streams from stock markets go along with many events and correlations.

Stock market sentiment analysis is one topic related to this domain. The stock market is too complex with many factors that can affect the market. People's opinion or sentiment is one of the major factors.

The real-time information needs from the stock market require the fast, efficient online-mining algorithms. The time-series data accounts for various data including the stock market data that updates along with time.

To predict the stock market, the past data is really important. Using the past return on certain stock, the future price of that stock can be predicted based on the price-data stream.

Web click streams and mining symbolic sequences

Web click streams data is large and continuously emerging, with hidden trends buried and to be discovered for various usages, such as recommendations. **TECNO-STREAMS** (**Tracking Evolving Clusters in NOisy Streams**) is a one-pass algorithm.

The TECNO-STREAMS algorithm

The whole algorithm is modeled on the following equations: the robust weight or activation function **(1)**, influence zone **(2)**, pure simulation **(3)**, optimal scale update **(4)**, incremental update of pure simulation and optimal update **(5)** and **(6)**, the simulation and scale values **(7)**, and finally, the D-W-B-cell update equations **(8)**.

$$w_{ij} = w_i\left(d_{ij}^2\right) = e - \left(\frac{d_{ij}^2}{2\sigma_{i,j}^2} + \frac{\left(J - j\right)}{\tau} \right)$$

$$IZ_i = \left\{ x_j \in x_a \mid w_{ij} \geq w_{\min} \right\}$$

$$S_{ai,J} = \frac{\sum_{j}^{J} = 1 w_{ij}}{\sigma_{i,J}^{2}}$$

$$\sigma_{i,J}^{2} = \frac{\sum_{j}^{J} = 1 w_{ij} d_{ij}^{2}}{2\sum_{j=1}^{J} w_{ij}}$$

$$S_{ai,J} = \frac{e^{-\frac{1}{\tau}} W_{i,J-1} + w_{iJ}}{\sigma_{i,J}^{2}}$$

$$\sigma_{i,J}^{2} = \frac{e^{-\frac{1}{\tau}} \sigma_{i,J}^{2} W_{i,J-1} + w_{iJ} d_{i,J}^{2}}{2\left(e^{-\frac{1}{\tau}} W_{i,J-1} + w_{iJ}\right)}$$

$$S_{i} = S_{ai,J} + \alpha\left(t\right)\frac{\sum_{l=1}^{N_{B}^{i}} w_{il}}{\sigma_{i,J}^{2}} - \beta\left(t\right)\frac{\sum_{l=1}^{N_{B}^{i}} w_{il}}{\sigma_{i,J}^{2}}$$

$$\sigma_{i,J}^{2} = \frac{1}{2}\frac{D_{i,J}^{2} + \alpha\left(t\right)\sum_{l=1}^{N_{B}^{i}} w_{il} d_{l}^{2} - \beta\left(t\right)\sum_{l=1}^{N_{B}^{i}} w_{il} d_{l}^{2}}{Wi,J + \alpha\left(t\right)\sum_{l=1}^{N_{B}^{i}} w_{il} - \beta\left(t\right)\sum_{l=1}^{N_{B}^{i}} w_{il}}$$

The similarity measures applied in the learning phase are defined here:

$$S_{\cos ij} = \frac{\sum_{k=1}^{n} x_{ik} \times p_{jk}}{\sqrt{\sum_{k=1}^{n} x_{ik} \sum_{k=1}^{n} x_{ik} p_{jk}}}$$

$$S_{\cos ij} = \sqrt{\mathrm{Prec}_{ij}^{L} Covg_{ij}^{L}}$$

$$\mathrm{Prec}_{ij}^{L} = \frac{\sum_{k=1}^{n} x_{ik} \times p_{jk}}{\sum_{k=1}^{n} p_{jk}}$$

$$Covg_{ij}^{L} = \frac{\sum_{k=1}^{n} x_{ik} \times p_{jk}}{\sum_{k=1}^{n} x_{jk}}$$

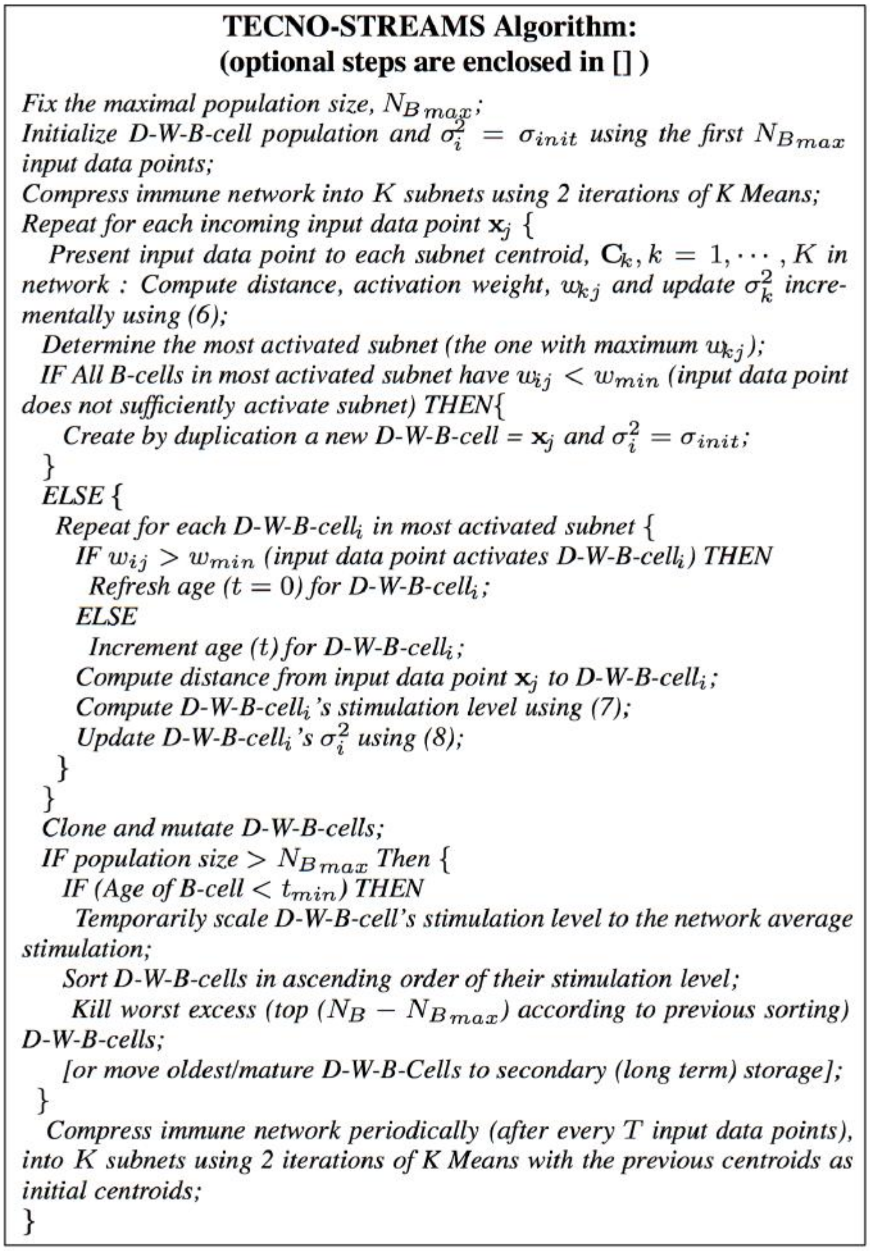
$$S_{\min ij} = \min\left\{\mathrm{Prec}_{ij}^{L}, \mathrm{Covg}_{ij}^{L}\right\}$$

The similarity measures applied in the validation phase are defined here:

$$\mathrm{Prec}_{ij}^{v} = \frac{\sum_{k=1}^{n} pL_{ik} \times pGT_{jk}}{\sum_{k=1}^{n} pL_{jk}}$$

$$Covg_{ij}^{v} = \frac{\sum_{k=1}^{n} pL_{ik} \times pGT_{jk}}{\sum_{k=1}^{n} pGT_{jk}}$$

The summarized pseudocode for the TECNO-STREAMS algorithm is as follows:



TECNO-STREAMS Algorithm:
(optional steps are enclosed in [])

```
Fix the maximal population size, $N_{B max}$;
Initialize D-W-B-cell population and $\sigma_i^2 = \sigma_{init}$ using the first $N_{B max}$
input data points;
Compress immune network into $K$ subnets using 2 iterations of K Means;
Repeat for each incoming input data point $\mathbf{x}_j$ {
  Present input data point to each subnet centroid, $\mathbf{C}_k, k = 1, \cdots, K$ in
network : Compute distance, activation weight, $w_{kj}$ and update $\sigma_k^2$ incre-
mentally using (6);
  Determine the most activated subnet (the one with maximum $w_{kj}$);
  IF All B-cells in most activated subnet have $w_{ij} < w_{min}$ (input data point
does not sufficiently activate subnet) THEN{
    Create by duplication a new D-W-B-cell = $\mathbf{x}_j$ and $\sigma_i^2 = \sigma_{init}$;
  }
  ELSE {
    Repeat for each D-W-B-cell$_i$ in most activated subnet {
      IF $w_{ij} > w_{min}$ (input data point activates D-W-B-cell$_i$) THEN
        Refresh age ($t = 0$) for D-W-B-cell$_i$;
      ELSE
        Increment age ($t$) for D-W-B-cell$_i$;
      Compute distance from input data point $\mathbf{x}_j$ to D-W-B-cell$_i$;
      Compute D-W-B-cell$_i$'s stimulation level using (7);
      Update D-W-B-cell$_i$'s $\sigma_i^2$ using (8);
    }
  }
  Clone and mutate D-W-B-cells;
  IF population size > $N_{B max}$ Then {
    IF (Age of B-cell < $t_{min}$) THEN
      Temporarily scale D-W-B-cell's stimulation level to the network average
stimulation;
    Sort D-W-B-cells in ascending order of their stimulation level;
      Kill worst excess (top ($N_B - N_{B max}$) according to previous sorting)
D-W-B-cells;
    [or move oldest/mature D-W-B-Cells to secondary (long term) storage];
  }
  Compress immune network periodically (after every $T$ input data points),
into $K$ subnets using 2 iterations of K Means with the previous centroids as
initial centroids;
}
```

The R implementation

Please take a look at the R codes file `ch_08_tecno_stream.R` from the bundle of R codes for previous algorithm. The codes can be tested with the following command:

```
> source("ch_08_tecno_stream.R")
```

Web click streams

The web click streams denote the user's behavior when visiting the site, especially for e-commerce sites and **CRM** (**Customer Relation Management**). The analysis of web click streams will improve the user experience of the customer and optimize the structure of the site to meet the customers' expectation and, finally, increase the income of the site.

In other aspects, web click streams mining can be used to detect DoS attacks, track the attackers, and prevent these on the Web in advance.

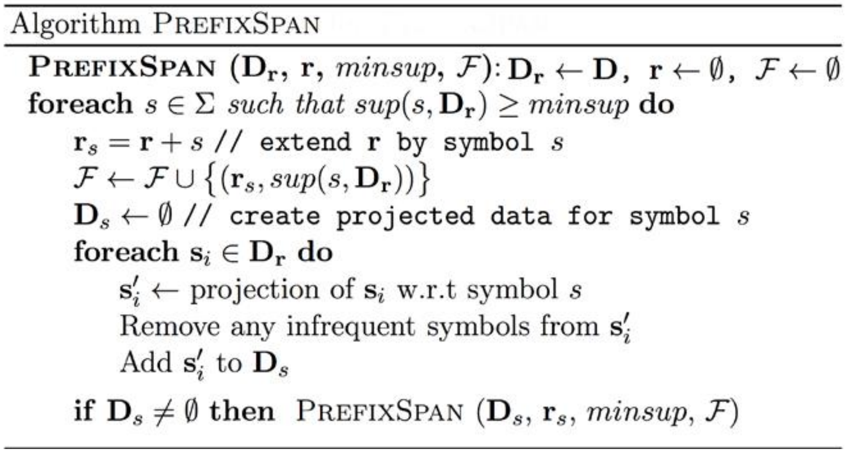
The dataset for web click stream is obviously the click records that get generated when the user visits various sites. The major characteristics of this dataset are that it is huge, and the size goes on increasing.

Mining sequence patterns in transactional databases

Mining sequence patterns can be thought of as association discovery over the temporal data or sequence dataset. Similarly, the classic pattern-mining algorithm should be extended or modified as per the sequence dataset's scenario.

The PrefixSpan algorithm

PrefixSpan is a frequent sequence-mining algorithm. The summarized pseudocode for the PrefixSpan algorithm is as follows:



```
Algorithm PrefixSpan
PrefixSpan ($\mathbf{D_r}$, $\mathbf{r}$, $minsup$, $\mathcal{F}$): $\mathbf{D_r} \leftarrow \mathbf{D}$, $\mathbf{r} \leftarrow \emptyset$, $\mathcal{F} \leftarrow \emptyset$
foreach $s \in \Sigma$ such that $sup(s, \mathbf{D_r}) \geq minsup$ do
    $\mathbf{r}_s = \mathbf{r} + s$ // extend r by symbol $s$
    $\mathcal{F} \leftarrow \mathcal{F} \cup \{(\mathbf{r}_s, sup(s, \mathbf{D_r}))\}$
    $\mathbf{D}_s \leftarrow \emptyset$ // create projected data for symbol $s$
    foreach $\mathbf{s}_i \in \mathbf{D_r}$ do
        $\mathbf{s}'_i \leftarrow$ projection of $\mathbf{s}_i$ w.r.t symbol $s$
        Remove any infrequent symbols from $\mathbf{s}'_i$
        Add $\mathbf{s}'_i$ to $\mathbf{D}_s$
    if $\mathbf{D}_s \neq \emptyset$ then PrefixSpan ($\mathbf{D}_s$, $\mathbf{r}_s$, $minsup$, $\mathcal{F}$)
```

The R implementation

Please take a look at the R codes file `ch_08_prefix_span.R` from the bundle of R codes for the previous algorithm. The codes can be tested with the following command:

```
> source("ch_08_prefix_span.R")
```

Time for action

Here are some questions for you to know whether you have understood the concepts:

- What is a stream?
- What is a time series?
- What is a sequence?

Summary

In this chapter, we looked at stream mining, time-series analysis, mining symbolic sequences, and mining sequence patterns.

The next chapter will cover the major topics related to graph mining, algorithms, and some examples related to them.

9
Graph Mining and Network Analysis

In this chapter, you will learn the algorithms written in R for graph mining and network analysis.

In this chapter, we will cover the following topics:

- Graph mining
- Mining frequent subgraph patterns
- Social network mining
- Social influence mining

Graph mining

Grouping, messaging, dating, and many other means are the major forms of social communication or the classic social behavior in the social network. All these concepts are modeled with graphs; that is, nodes, edges, and other attributes. Graph mining is developed to mine this information, which is similar to other types of information, such as biological information, and so on.

Graph

Graph G contains nodes V and edges E and is represented with an equation, *G = (V, E)*. As per graph mining, there are some concepts that need to be clarified. There are two types of graphs: directed graphs, which have ordered pairs of vertices in the edge set, E, and undirected graphs.

Graph mining algorithms

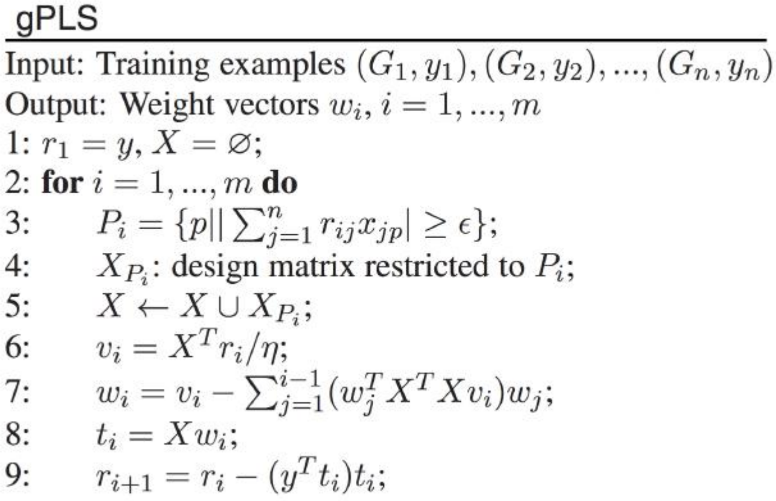
Although the data instances under research are very different from the other data types that we saw earlier in this book, graph-mining algorithms still include frequent pattern (subgraph) mining, classification, and clustering.

In the next section, we will look at frequent subgraph patterns mining algorithm, links mining, and clustering.

Mining frequent subgraph patterns

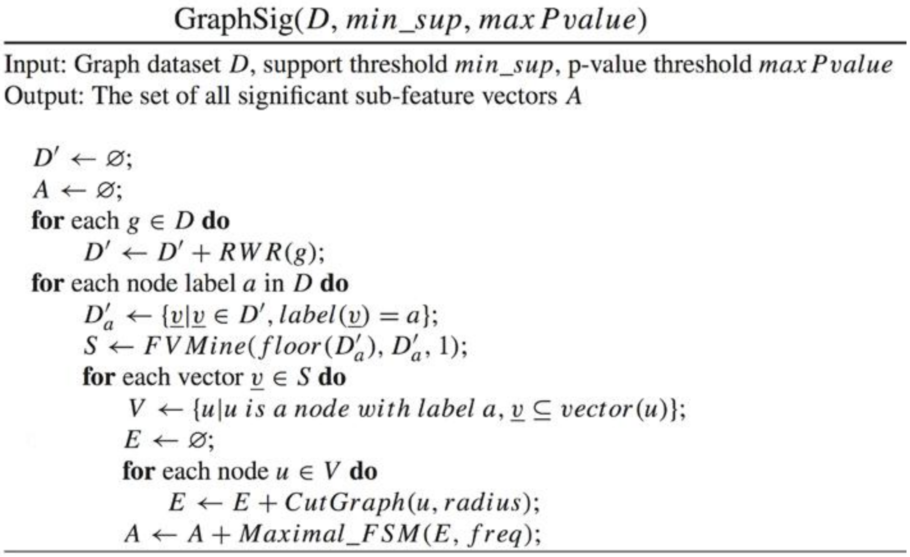
The subgraph pattern or graph pattern is an important application of data mining; this is used for bioinformatics, social network analysis, and so on. Frequent subgraph patterns are patterns that occur frequently in a set of graphs or in a large graph.

The gPLS algorithm



```
gPLS
Input: Training examples $(G_1, y_1), (G_2, y_2), ..., (G_n, y_n)$
Output: Weight vectors $w_i, i = 1, ..., m$
$r_1 = y, X = \varnothing$;
for $i = 1, ..., m$ do
    $P_i = \{p | |\sum_{j=1}^{n} r_{ij} x_{jp}| \geq \epsilon\}$;
    $X_{P_i}$: design matrix restricted to $P_i$;
    $X \leftarrow X \cup X_{P_i}$;
    $v_i = X^T r_i / \eta$;
    $w_i = v_i - \sum_{j=1}^{i-1} (w_j^T X^T X v_i) w_j$;
    $t_i = X w_i$;
    $r_{i+1} = r_i - (y^T t_i) t_i$;
```

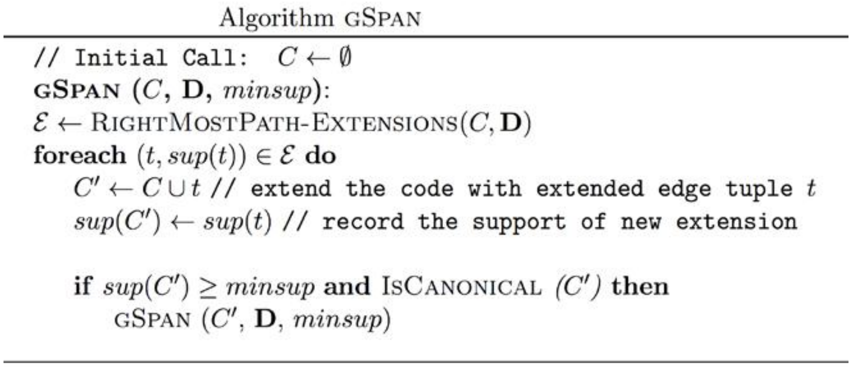
The GraphSig algorithm



```
GraphSig($D$, $min\_sup$, $maxPvalue$)
Input: Graph dataset $D$, support threshold $min\_sup$, p-value threshold $maxPvalue$
Output: The set of all significant sub-feature vectors $A$

$D' \leftarrow \varnothing$;
$A \leftarrow \varnothing$;
for each $g \in D$ do
    $D' \leftarrow D' + RWR(g)$;
for each node label $a$ in $D$ do
    $D'_a \leftarrow \{\underline{v} | \underline{v} \in D', label(\underline{v}) = a\}$;
    $S \leftarrow FVMine(floor(D'_a), D'_a, 1)$;
    for each vector $\underline{v} \in S$ do
        $V \leftarrow \{u | u\ is\ a\ node\ with\ label\ a, \underline{v} \subseteq vector(u)\}$;
        $E \leftarrow \varnothing$;
        for each node $u \in V$ do
            $E \leftarrow E + CutGraph(u, radius)$;
        $A \leftarrow A + Maximal\_FSM(E, freq)$;
```

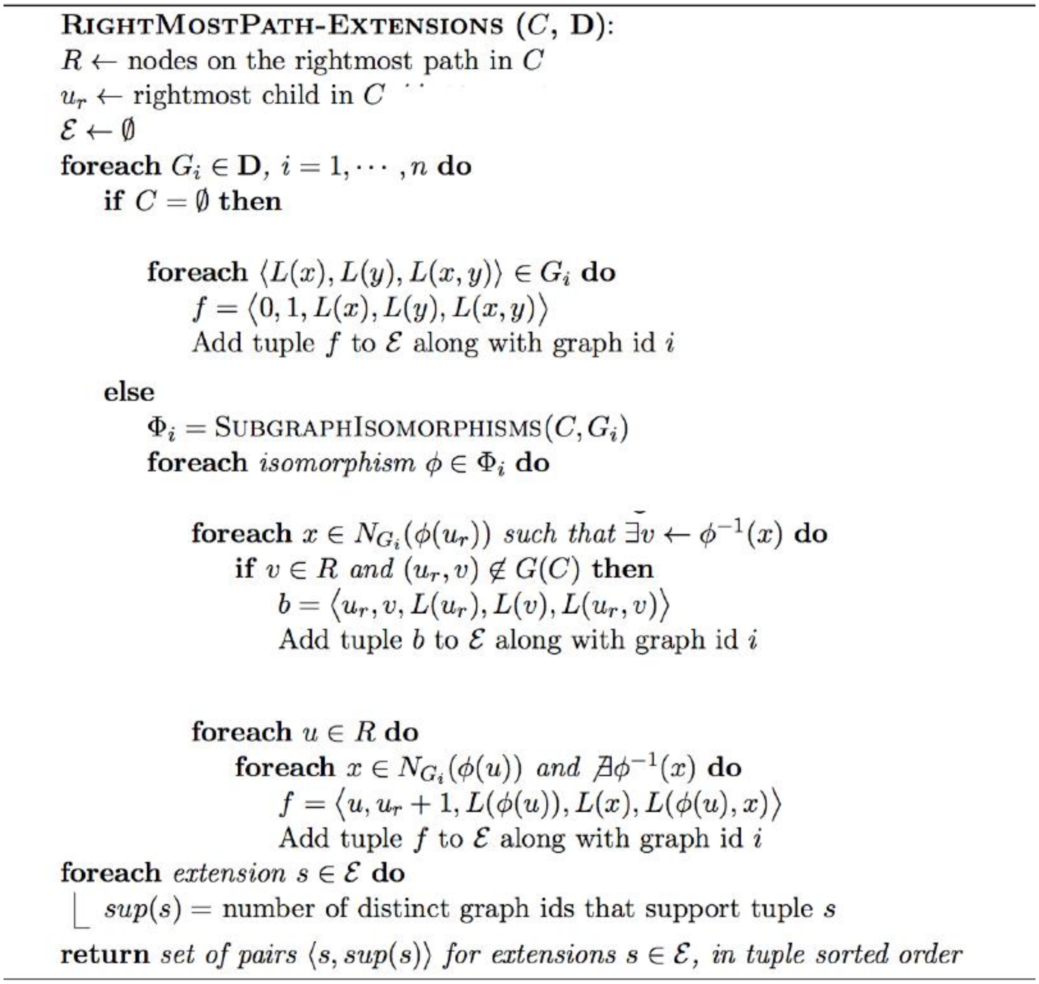
The gSpan algorithm

The summarized pseudocode for the gSpan algorithm is as follows:



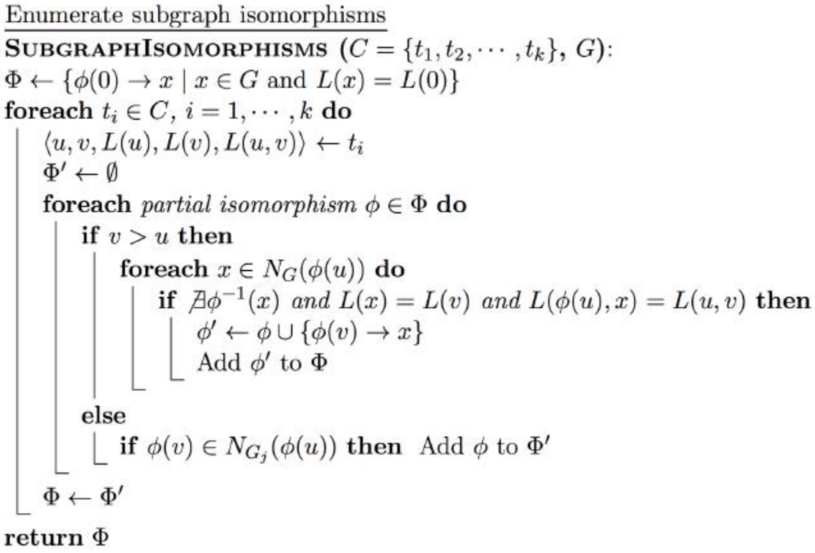
```
Algorithm gSpan
// Initial Call:  $C \leftarrow \emptyset$
gSpan ($C$, $\mathbf{D}$, $minsup$):
$\mathcal{E} \leftarrow$ RightMostPath-Extensions($C$, $\mathbf{D}$)
foreach $(t, sup(t)) \in \mathcal{E}$ do
    $C' \leftarrow C \cup t$ // extend the code with extended edge tuple $t$
    $sup(C') \leftarrow sup(t)$ // record the support of new extension

    if $sup(C') \geq minsup$ and IsCanonical $(C')$ then
        gSpan ($C'$, $\mathbf{D}$, $minsup$)
```

Rightmost path extensions and their supports



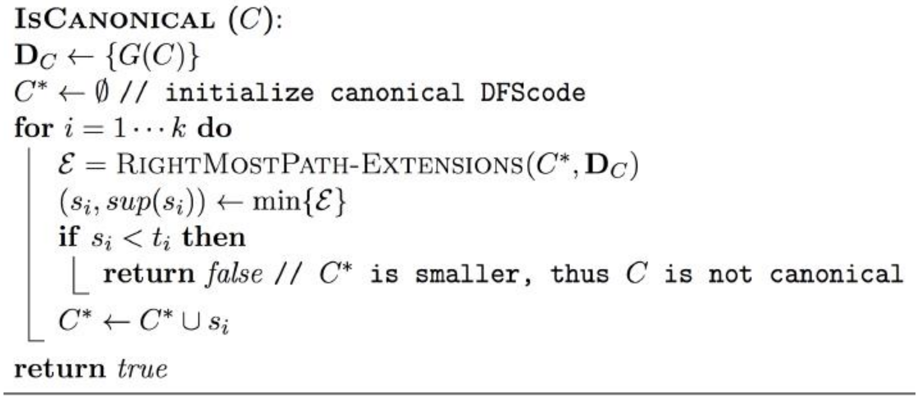
```
RightMostPath-Extensions ($C$, $\mathbf{D}$):
$R \leftarrow$ nodes on the rightmost path in $C$
$u_r \leftarrow$ rightmost child in $C$
$\mathcal{E} \leftarrow \emptyset$
foreach $G_i \in \mathbf{D}$, $i = 1, \cdots, n$ do
    if $C = \emptyset$ then

        foreach $\langle L(x), L(y), L(x,y) \rangle \in G_i$ do
            $f = \langle 0, 1, L(x), L(y), L(x,y) \rangle$
            Add tuple $f$ to $\mathcal{E}$ along with graph id $i$

    else
        $\Phi_i = $ SubgraphIsomorphisms$(C, G_i)$
        foreach isomorphism $\phi \in \Phi_i$ do

            foreach $x \in N_{G_i}(\phi(u_r))$ such that $\exists v \leftarrow \phi^{-1}(x)$ do
                if $v \in R$ and $(u_r, v) \notin G(C)$ then
                    $b = \langle u_r, v, L(u_r), L(v), L(u_r, v) \rangle$
                    Add tuple $b$ to $\mathcal{E}$ along with graph id $i$


            foreach $u \in R$ do
                foreach $x \in N_{G_i}(\phi(u))$ and $\nexists \phi^{-1}(x)$ do
                    $f = \langle u, u_r + 1, L(\phi(u)), L(x), L(\phi(u), x) \rangle$
                    Add tuple $f$ to $\mathcal{E}$ along with graph id $i$
foreach extension $s \in \mathcal{E}$ do
    $sup(s) = $ number of distinct graph ids that support tuple $s$
return set of pairs $\langle s, sup(s) \rangle$ for extensions $s \in \mathcal{E}$, in tuple sorted order
```

The subgraph isomorphism enumeration algorithm



```
Enumerate subgraph isomorphisms
SubgraphIsomorphisms ($C = \{t_1, t_2, \cdots, t_k\}$, $G$):
$\Phi \leftarrow \{\phi(0) \rightarrow x \mid x \in G$ and $L(x) = L(0)\}$
foreach $t_i \in C$, $i = 1, \cdots, k$ do
    $\langle u, v, L(u), L(v), L(u,v) \rangle \leftarrow t_i$
    $\Phi' \leftarrow \emptyset$
    foreach partial isomorphism $\phi \in \Phi$ do
        if $v > u$ then
            foreach $x \in N_G(\phi(u))$ do
                if $\nexists \phi^{-1}(x)$ and $L(x) = L(v)$ and $L(\phi(u), x) = L(u, v)$ then
                    $\phi' \leftarrow \phi \cup \{\phi(v) \rightarrow x\}$
                    Add $\phi'$ to $\Phi$
        else
            if $\phi(v) \in N_{G_j}(\phi(u))$ then  Add $\phi$ to $\Phi'$
    $\Phi \leftarrow \Phi'$
return $\Phi$
```

The canonical checking algorithm



```
IsCanonical ($C$):
$\mathbf{D}_C \leftarrow \{G(C)\}$
$C^* \leftarrow \emptyset$ // initialize canonical DFScode
for $i = 1 \cdots k$ do
    $\mathcal{E}$ = RightMostPath-Extensions($C^*, \mathbf{D}_C$)
    $(s_i, sup(s_i)) \leftarrow \min\{\mathcal{E}\}$
    if $s_i < t_i$ then
        return false // $C^*$ is smaller, thus $C$ is not canonical
    $C^* \leftarrow C^* \cup s_i$
return true
```

The R implementation

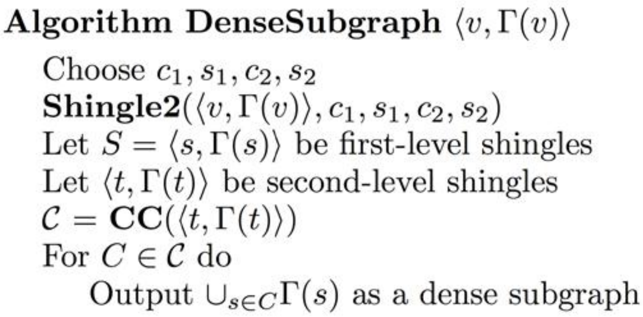
Please take a look at the R codes file `ch_09_gspan.R` from the bundle of R codes for previously mentioned algorithms. The codes can be tested with the following command:

```
> source("ch_09_gspan.R")
```

Social network mining

Social network is based on human interactions, from the most classical definition. The data instances collected in the social network have graph-like and temporal characteristics. There are two major strategies for data mining tasks for social networks: one is linkage-based or structure-based, and the other is content-based. The data instances collected in the social network also have two kinds of data instances: static and dynamic or times-series data, such as the tweets on Twitter. Due to the characteristics of the data instance of graphs, there are vast versatile algorithms developed to solve the challenges.

Community detection and the shingling algorithm



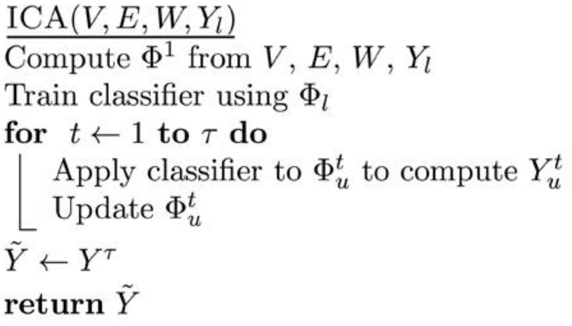
```
**Algorithm DenseSubgraph** $\langle v, \Gamma(v) \rangle$
    Choose $c_1, s_1, c_2, s_2$
    **Shingle2**$(\langle v, \Gamma(v) \rangle, c_1, s_1, c_2, s_2)$
    Let $S = \langle s, \Gamma(s) \rangle$ be first-level shingles
    Let $\langle t, \Gamma(t) \rangle$ be second-level shingles
    $\mathcal{C} = \mathbf{CC}(\langle t, \Gamma(t) \rangle)$
    For $C \in \mathcal{C}$ do
        Output $\cup_{s \in C} \Gamma(s)$ as a dense subgraph
```

```
**Algorithm CC** $(t_1, \Gamma(t_1), \ldots, t_m, \Gamma(t_m))$
    Let $V_S = \cup_{i=1}^{m} \Gamma(t_i)$
    Let $\mathcal{C} = \{\{s\} \mid s \in V_S\}$
    For $i = 1$ to $m$ do
        Merge the sets $\{$ Find$(s) \mid s \in \Gamma(t_i)\}$ in $\mathcal{C}$
    For $C \in \mathcal{C}$
        Output cluster $\{s \mid s \in C\}$
```

```
**Algorithm Shingle2**$(v_1, \ldots, v_n, s_1, c_1, s_2, c_2)$
    For $i = 1$ to $n$ do
        $S_1(v_i) =$ **Shingle**$(\Gamma(v_i), c_1, s_1)$
    Let $S = \cup_{i=1}^{n} S_1(v_i)$
    For $s \in S$ do
        Let $\Gamma(s) = \{v \mid S_1(v) \ni s\}$
        $S_2(s) =$ **Shingle**$(\Gamma(s), c_2, s_2)$
    Let $T = \cup_{s \in S} S_2(s)$
    For $t \in T$ do
        Let $\Gamma(t) = \{s \in S \mid S_2(s) \ni t\}$
        Output $\langle t, \Gamma(t) \rangle$
```

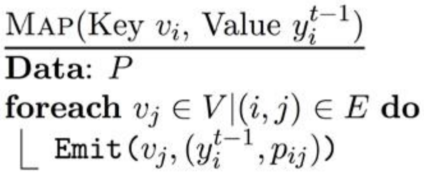
```
**Algorithm Shingle**$(a_1, \ldots, a_n, s, c)$
    Let $H$ be a hash function from strings to integers
    Let $p$ be a large random prime (say, 32 bits)
    Let $a_1, b_1, \ldots, a_c, b_c$ be random integers in $[1 \ldots p]$
    For $i = 1$ to $n$ do $x_i = H(\text{"}a_i\text{"})$
    For $j = 1$ to $c$ do
        For $i = 1$ to $n$ do $y_i = (a_j * x_i + b_j) \bmod p$
        Let $y'_1, \ldots, y'_s$ be $s$ minimal elements of $y$
        Let $z_j = H(\text{"}y'_1 \circ \cdots \circ y'_s\text{"})$
    Output $z_1, \ldots, z_c$
```

The node classification and iterative classification algorithms



```
$\underline{\mathrm{ICA}(V, E, W, Y_l)}$
Compute $\Phi^1$ from $V$, $E$, $W$, $Y_l$
Train classifier using $\Phi_l$
for  $t \leftarrow 1$ to $\tau$ do
    Apply classifier to $\Phi_u^t$ to compute $Y_u^t$
    Update $\Phi_u^t$
$\tilde{Y} \leftarrow Y^\tau$
return $\tilde{Y}$
```

The second-order algorithm to reduce the number of iterations is as follows:



```
$\underline{\textsc{Map}(\text{Key } v_i, \text{ Value } y_i^{t-1})}$
Data: $P$
foreach $v_j \in V | (i, j) \in E$ do
    Emit$(v_j, (y_i^{t-1}, p_{ij}))$
```

The R implementation

Please take a look at the R codes file `ch_09_shingling.R` from the bundle of R codes for previously mentioned algorithms. The codes can be tested with the following command:

```
> source("ch_09_shingling.R")
```

Time for action

Here are some practice questions for you to check whether you have understood the concepts:

- What is a graph?
- What graph opportunities are used?
- What is the PageRank algorithm, and what is its application in web search?

Summary

In this chapter, we looked at:

- Graph mining. We also saw that the characteristics of graph data can be divided into frequent pattern mining, classification, and clustering
- Mining frequent subgraph patterns is done to find the frequent patterns in a set of graphs or a single massive graph
- Social network analysis includes a wide range of web applications with broad definitions, such as Facebook, LinkedIn, Google+, StackOverflow, and so on

In the next chapter, we will focus on the major topics related to web mining and algorithms and look at some examples based on them.

10

Mining Text and Web Data

In this chapter, you will learn the algorithm written in R for text mining and web data mining.

For text mining, the semistructured and nonstructured documents are the main dataset. There are a few of major categories of text mining, such as clustering, document retrieval and representation, and anomaly detection. The application of text mining includes, but is not limited to, topic tracking, and text summarization and categorization.

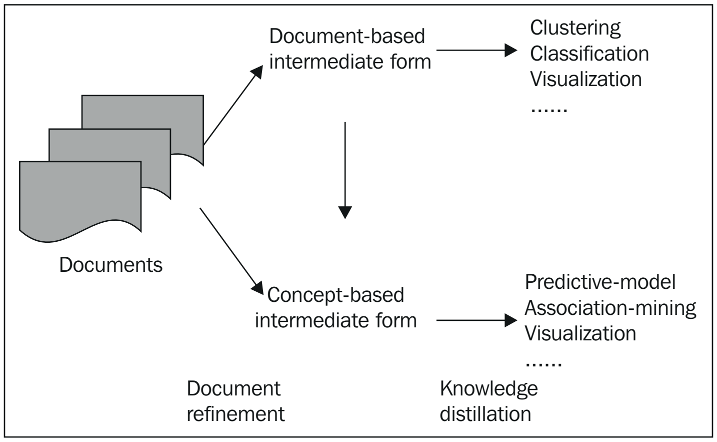
Web content, structure, and usage mining is one application of web mining. Web mining is also used for user behavior modeling, personalized views and content annotation, and so on. In another aspect, web mining integrates the result information from the traditional data-mining technologies and the information from WWW.

In this chapter, we will cover the following topics:

- Text mining and the TM package
- Text summarization
- The question answering system
- Genre categorization of web pages
- Categorization of newspaper articles and newswires into topics
- Web usage mining with web logs

Text mining and TM packages

Along with the appearance of text mining, due to the characteristics of text or documents, the traditional data-mining algorithms need some minor adjustments or extensions. The classical text-mining process is as follows:



The popular text-clustering algorithms include the distance-based clustering algorithm, the hierarchical clustering algorithm, the partition-based clustering algorithm, and so on.

The popular text-classification algorithms include decision trees, pattern-based classification, SVM classification, Bayesian classification, and so on.

As a popular preprocessing step, here are the details of the word-extraction algorithm.

Text summarization

The target of text summarization is to generate a concise and coherent conclusion or summary of the major information of the input. Three major steps are performed in most of the summarization systems. They are as follows:

1. Build a temporary structure that contains the main portion of the key point of the input text.
2. Next, the sentences from the input are scored with the output of the first step.
3. Lastly, a final summary that represents the input documents is set up by several sentences.

One popular strategy is to remove the unimportant information, clauses, or sentences and, at the same time, build classifiers to make sure that the key information is not thrown away, which is, in another viewpoint, the relative importance of topics functioned here during the summarization process. The final result is represented in a coherent way.

The summarization is a dynamic nonstop process. First, we need to build summaries on the set of old documents' dataset, which means multidocuments' summarization. The second step is to summarize the new documents when we get the result of the first step.

Due to the difficulty in building a new sentence for the summaries, one solution is extractive summarization, that is, to extract the most relevant sentences from the document dataset or documents. With the growth of the size of the documents set, the time evolution of the topics is also an important issue for which statistical topic modeling such as time-series-based algorithms are proposed.

There are two popular solutions for intermediate representation: topic representation and indicator representation. Sentence scoring, the score of each sentence, is determined by a couple of important factors such as the combination of characters. Summary sentence selection is determined by the most important *N* sentences.

There are many good characteristics that describe the final summary: it is indicative or informative, extract or abstract, generic or query-oriented, consists of background or just the news, is monolingual or cross lingual, and consists of a single document or multiple documents.

The benefit of text summarization is to improve the efficiency of document processing during which the summarization of a certain document can help the reader decide whether to analyze the document for specific purposes. One example is to summarize the multilingual, large (nearing unlimited), dynamic dataset of documents from various sources, including the Web. Examples include summarization of medical articles, e-mail, the Web, speech, and so on.

Topic representation

Topic representation such as topic signature plays an important role in the document-summarization system. Various topic representations are provided, such as topic signature, enhanced topic signature, thematic signature, and so on.

The topic signature is defined as a set of related terms; topic is the target concept, and signature is a list of terms that is related to the topic with a specific weight. Each term can be the stemmed content words, bigram or trigram:

$$TS = \{topic, signature\} = \{topic, < (t_1, w_1), \ldots, (t_n, w_n) >\}$$

The topic term selection process is as follows. The input document sets are divided into two sets: relevant and nonrelevant texts for the topic. Two hypotheses are defined:

$$Hypothesis\ 1, (H_1), P(R|T_i) = p = P(R|\widetilde{t_i})$$

$$Hypothesis\ 2, (H_2), P(R|T_i) = p_1 \neq p_2 = P(R|\widetilde{t_i})$$

The first hypothesis denotes that the relevance of a document is independent of the term, whereas the second hypothesis indicates a strong relevance with the presence of those terms, given that $p_1 \gg p_2$ and the 2 x 2 contingency table:

	R	$\tilde{R}$
t_1	O_{11}	O_{12}
t_1	O_{21}	O_{22}

O_{11} represents the frequency of the term, t_1, occurring in R, and O_{12}, is the O_{21} frequency of the term occurring in $\tilde{R}$. O_{21} is the frequency of the term, $\widetilde{t_i} \neq t_i$, occurring in R, and O_{22} is the frequency of the term, $\widetilde{t_i} \neq t_i$, occurring in $\tilde{R}$.

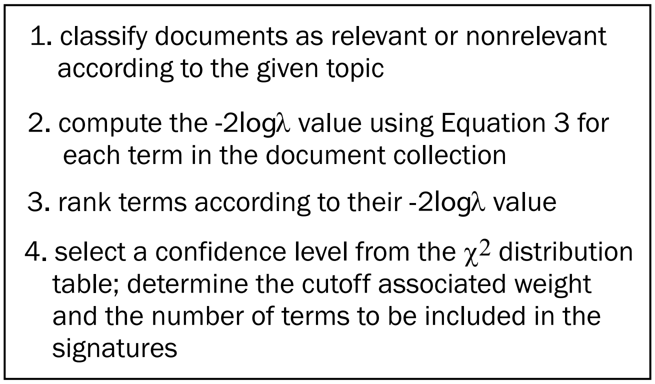
The likelihood of both hypotheses is calculated as follows; here, *b* denotes the binomial distribution:

$$L(H_1) = b(O_{11}; O_{11} + O_{12}, p) * b(O_{21}; O_{21} + O_{22}, p)$$

$$L(H_2) = b(O_{11}; O_{11} + O_{12}, p_1) * b(O_{21}; O_{21} + O_{22}, p_2)$$

$$-2 \log \lambda = \frac{L(H_1)}{L(H_2)}$$

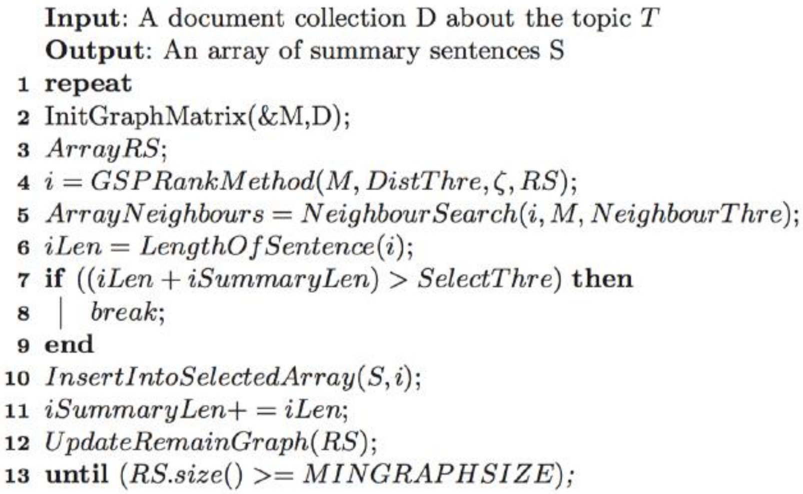
The algorithm to create the topic signature for a given topic is illustrated as follows:



1. classify documents as relevant or nonrelevant according to the given topic
2. compute the -2logλ value using Equation 3 for each term in the document collection
3. rank terms according to their -2logλ value
4. select a confidence level from the χ^2 distribution table; determine the cutoff associated weight and the number of terms to be included in the signatures

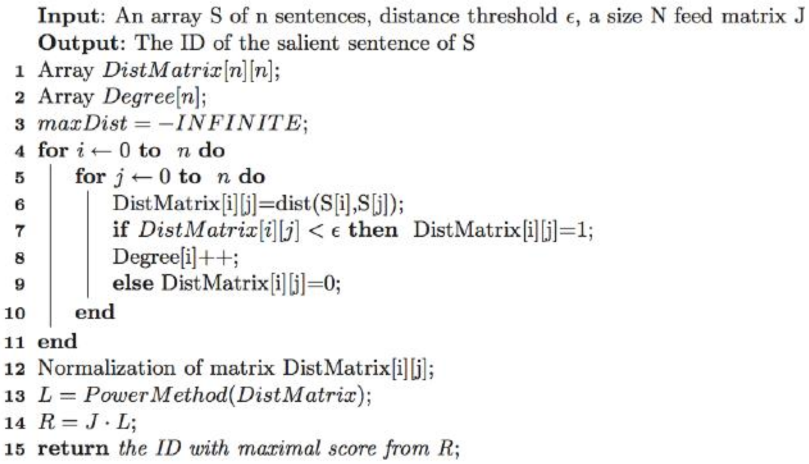
The multidocument summarization algorithm

Here is the **Graph-Based Sub-topic Partition Algorithm** (**GSPSummary**) algorithm for multidocument summarization:



```
Input: A document collection D about the topic $T$
Output: An array of summary sentences S
repeat
InitGraphMatrix(&M,D);
$ArrayRS$;
$i = GSPRankMethod(M, DistThre, \zeta, RS)$;
$ArrayNeighbours = NeighbourSearch(i, M, NeighbourThre)$;
$iLen = LengthOfSentence(i)$;
if $((iLen + iSummaryLen) > SelectThre)$ then
    $break$;
end
$InsertIntoSelectedArray(S, i)$;
$iSummaryLen += iLen$;
$UpdateRemainGraph(RS)$;
until $(RS.size() >= MINGRAPHSIZE)$;
```

The GSPRankMethod is defined as follows:



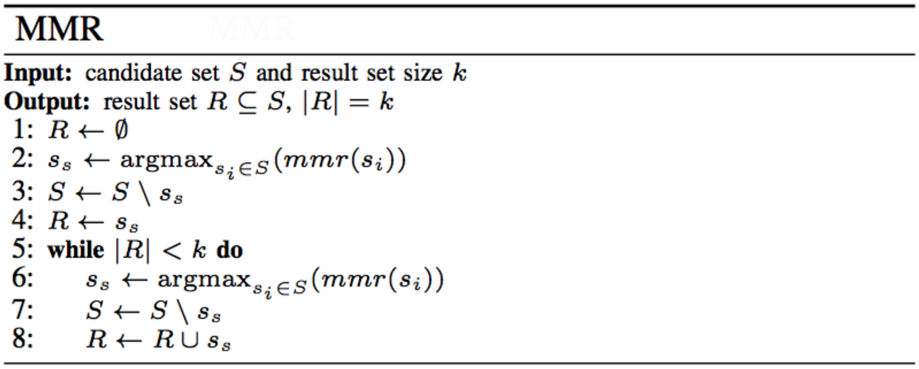
```
Input: An array S of n sentences, distance threshold $\epsilon$, a size N feed matrix J
Output: The ID of the salient sentence of S
Array $DistMatrix[n][n]$;
Array $Degree[n]$;
$maxDist = -INFINITE$;
for $i \leftarrow 0$ to $n$ do
    for $j \leftarrow 0$ to $n$ do
        DistMatrix[i][j]=dist(S[i],S[j]);
        if $DistMatrix[i][j] < \epsilon$ then DistMatrix[i][j]=1;
        Degree[i]++;
        else DistMatrix[i][j]=0;
    end
end
Normalization of matrix DistMatrix[i][j];
$L = PowerMethod(DistMatrix)$;
$R = J \cdot L$;
return the ID with maximal score from $R$;
```

The Maximal Marginal Relevance algorithm

Maximal Marginal Relevance (**MMR**), which is comparatively suited for query-based and multidocument summarization, selects the most important sentence in each iteration of sentence selection. Each selected sentence has minimal relevance to the selected sentence set.

$$MMR \overset{\text{def}}{=} Arg \max_{D_i \in R \setminus S} \left[\lambda (Sim_1(D_i, Q) - (1-\lambda) \max_{D_j \in S} Sim_2(D_i, D_j)) \right]$$

The summarized algorithm of MMR is as follows:



```
MMR
Input: candidate set $S$ and result set size $k$
Output: result set $R \subseteq S$, $|R| = k$
$R \leftarrow \emptyset$
$s_s \leftarrow \text{argmax}_{s_i \in S}(mmr(s_i))$
$S \leftarrow S \setminus s_s$
$R \leftarrow s_s$
while $|R| < k$ do
    $s_s \leftarrow \text{argmax}_{s_i \in S}(mmr(s_i))$
    $S \leftarrow S \setminus s_s$
    $R \leftarrow R \cup s_s$
```

The R implementation

Please take a look at the R codes file `ch_10_mmr.R` from the bundle of R codes for the preceding algorithms. The codes can be tested with the following command:

```
> source("ch_10_mmr.R")
```

The question answering system

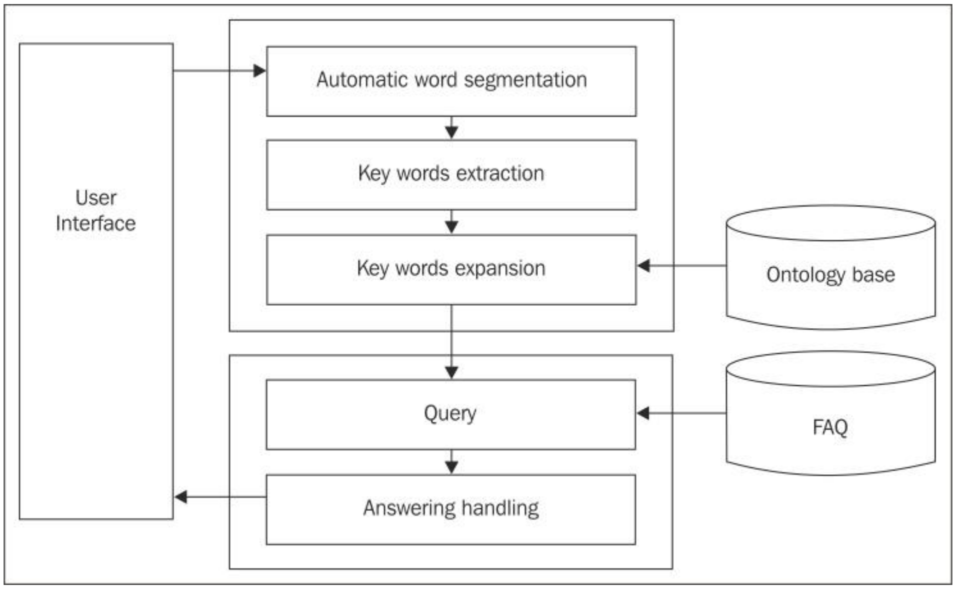
The **question answering** (**QA**) system is a one hot topic related to IR, IE, NLP, data mining, and so on. A QA system mines a large set of texts to find a short phrase or sentence that answers a user's question with a certain precision. If the source is the Web or, even further, the whole information world, the challenge of the QA system increases dramatically.

There are basically three kinds of QA systems: slot filling, for which the format of the query and answer are similar; limited domain, for which the domains are limited with dictionaries and ontologies; and open domain, for which there is no limitation of domains.

One of the conceptual architectures among various solutions is illustrated in the following image. The input of the QA system is natural language questions; the output of the system, that is, the answer to the input question, is provided in natural language too. The system is composed of three major parts: the user interface, the processing part for questions, and the generation part for answers.

Assuming the question to be a coherent sentence, the questions from the user interface are processed, and the final right query is generated after a question-term identification, such as word segmentation and key word extraction and expansion. The ontology base here serves to support query expansion.

The answer is selected or generated from the FAQ base by the query that is provided in the earlier steps.



One popular question answering system is Yahoo! Answers, where a huge number of questions are asked and answered every day.

Genre categorization of web pages

Genre categorization can be used for large article corpora and web pages. A genre can be defined in terms of purpose and physical form. It denotes any widely accepted categories of texts defined by common communicative purpose or other functional traits, and the categories are extensible. The web genre can also be defined based on the facets, complexity of the language, subjectivity, and number of graphics. Genre categorization has many applications such as improving search efficiency and satisfying the users' information need.

For the WWW or web pages, the genre can be defined as the usability of the miner and feasibility with respect to the efficiency.

There are some major challenges for this web page genre categorization. One is the instability of the Web itself. The second is the complex and unpredictable properties of web pages. The third is how to judge the genre for a specific web page. There are more challenges, but they are not listed here, or they will appear in future applications. For certain web pages, they might have multiple genres or no genre for existing recognized genres libraries.

Due to the fast pace of the evolution of the Web, new genres are continuously introduced to the current genre classes and the current genre classes are continuously updated and upgraded.

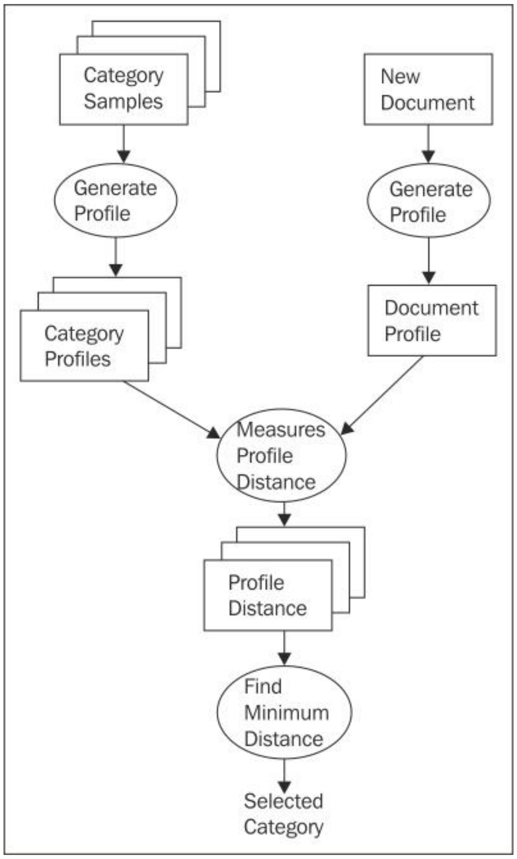
Possible solutions include, but are not limited to, Naïve Bayes, k-Nearest Neighbor, SVM, and tree nodes as classification methods.

Categorizing newspaper articles and newswires into topics

Articles and newswires denote the huge periodical source of events of knowledge at different periods of time. The classification of text is the preprocessing step to store all these documents into a specific corpus. The categorization of text is the base of text processing.

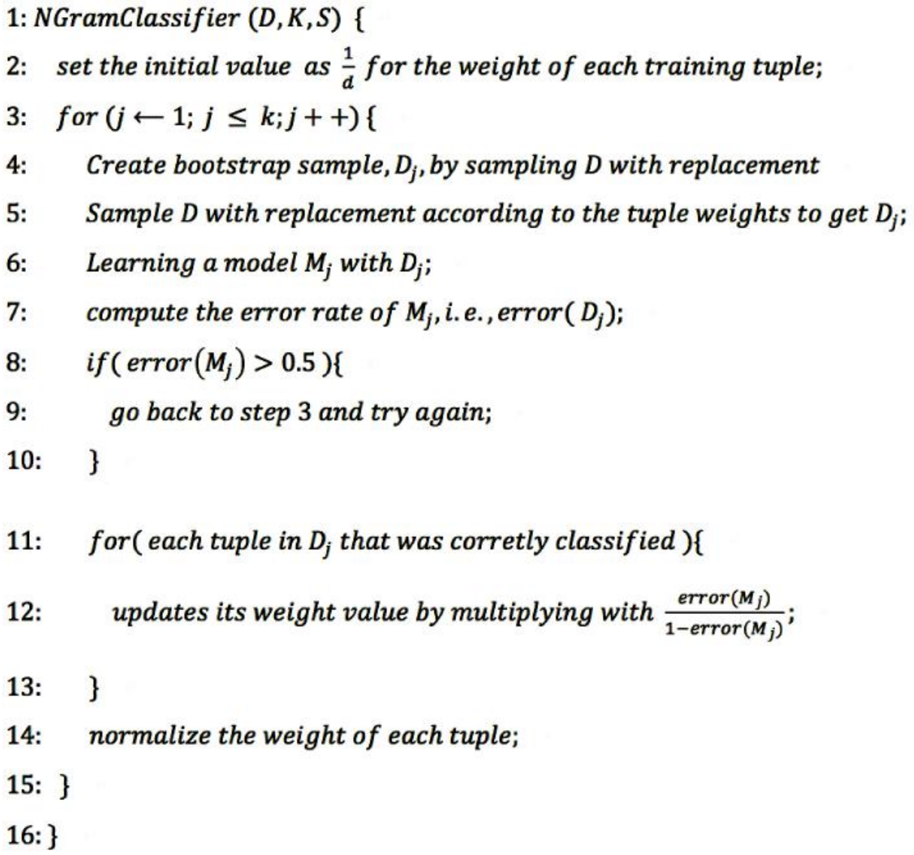
We will now introduce an N-gram-based text-classification algorithm. From a longer string, an N-character slice is called N-gram. The key point of this algorithm is the calculation of the profiles of the N-gram frequencies.

Before the introduction of the algorithm, here are the necessary illustrations of a couple of concepts adopted in the algorithm:



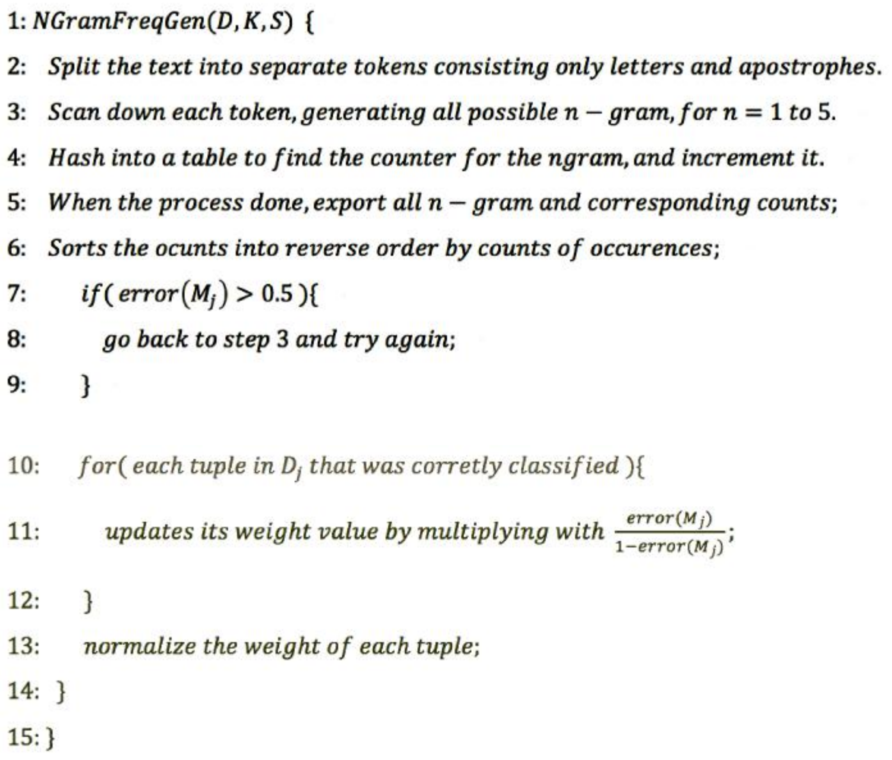
The N-gram-based text categorization

The summarized pseudocodes for the N-gram-based text-categorization algorithm are as follows:



```
1: $NGramClassifier\ (D, K, S)$ {
2:    set the initial value as $\frac{1}{d}$ for the weight of each training tuple;
3:    for ($j \leftarrow 1$; $j \leq k$; $j++$) {
4:       Create bootstrap sample, $D_j$, by sampling $D$ with replacement
5:       Sample $D$ with replacement according to the tuple weights to get $D_j$;
6:       Learning a model $M_j$ with $D_j$;
7:       compute the error rate of $M_j$, i.e., $error(D_j)$;
8:       if( $error(M_j) > 0.5$ ){
9:          go back to step 3 and try again;
10:      }

11:      for( each tuple in $D_j$ that was corretly classified ){
12:         updates its weight value by multiplying with $\frac{error(M_j)}{1-error(M_j)}$;
13:      }
14:      normalize the weight of each tuple;
15:   }
16: }
```

The N-gram frequency-generation algorithm is as follows:



```
1: NGramFreqGen(D, K, S) {
2:   Split the text into separate tokens consisting only letters and apostrophes.
3:   Scan down each token, generating all possible $n - gram$, for $n = 1$ to 5.
4:   Hash into a table to find the counter for the ngram, and increment it.
5:   When the process done, export all $n - gram$ and corresponding counts;
6:   Sorts the ocunts into reverse order by counts of occurences;
7:       if( $error(M_j) > 0.5$ ){
8:          go back to step 3 and try again;
9:       }

10:      for( each tuple in $D_j$ that was corretly classified ){
11:         updates its weight value by multiplying with $\frac{error(M_j)}{1-error(M_j)}$;
12:      }
13:      normalize the weight of each tuple;
14:  }
15: }
```

The R implementation

Please take a look at the R codes file `ch_10_ngram_classifier.R` from the bundle of R codes for the above algorithms. The codes can be tested with the following command:

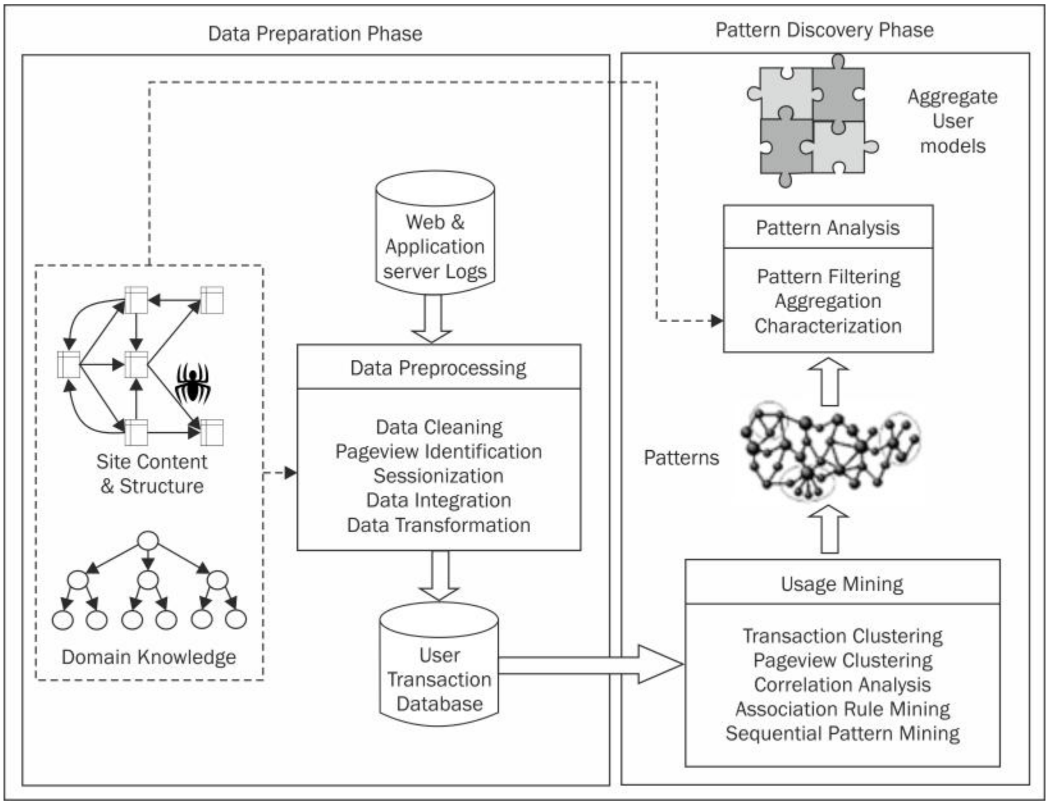
```
> source("ch_10_ngram_classifier.R")
```

Web usage mining with web logs

Web usage mining denotes the discovery and analytics of patterns in web logs (such as system access logs) and transactions. The output is the relation of user interaction and resources on the Web. The user behavior can be identified based on this output. The web logs record the track of the web user's interaction with web servers, web proxy servers, and browsers.

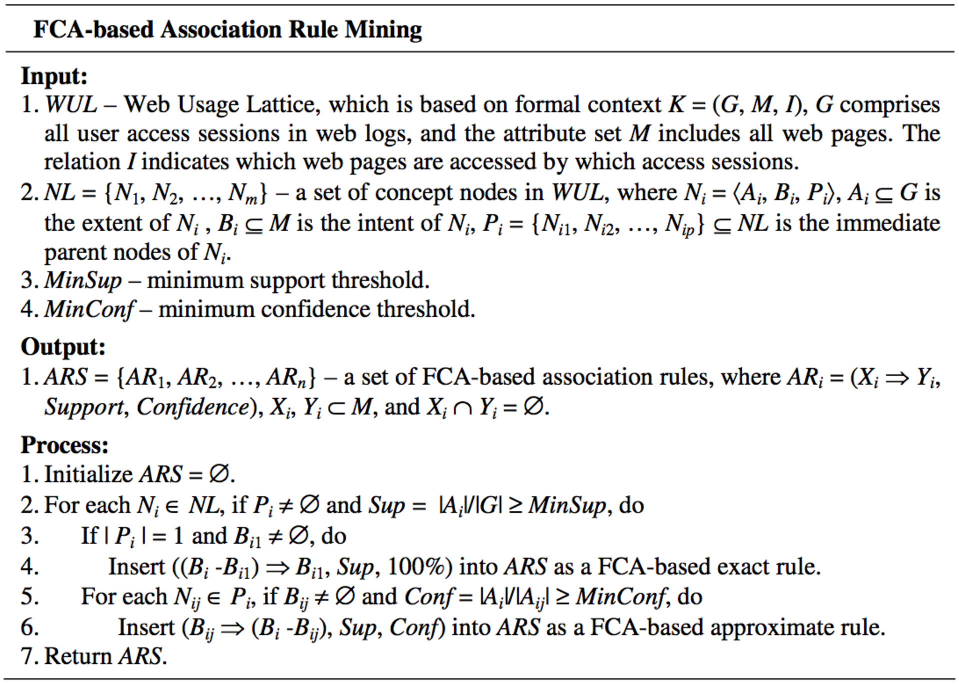
The popular web usage mining process is illustrated in the images, and it includes three major steps: data collection and preprocessing, pattern discovery, and pattern analysis.

The preprocessing contains data cleaning, session identification, and data conversion. The pattern discovery includes path analysis, association rules, sequential patterns, and cluster and classification rules.



The FCA-based association rule mining algorithm

The summarized pseudocodes for the FCA-based association rule mining algorithm are as follows:



FCA-based Association Rule Mining

Input:

1. *WUL* – Web Usage Lattice, which is based on formal context $K = (G, M, I)$, G comprises all user access sessions in web logs, and the attribute set M includes all web pages. The relation I indicates which web pages are accessed by which access sessions.
2. $NL = \{N_1, N_2, \ldots, N_m\}$ – a set of concept nodes in *WUL*, where $N_i = \langle A_i, B_i, P_i \rangle$, $A_i \subseteq G$ is the extent of N_i , $B_i \subseteq M$ is the intent of N_i, $P_i = \{N_{i1}, N_{i2}, \ldots, N_{ip}\} \subseteq NL$ is the immediate parent nodes of N_i.
3. *MinSup* – minimum support threshold.
4. *MinConf* – minimum confidence threshold.

Output:

1. $ARS = \{AR_1, AR_2, \ldots, AR_n\}$ – a set of FCA-based association rules, where $AR_i = (X_i \Rightarrow Y_i$, *Support*, *Confidence*), X_i, $Y_i \subset M$, and $X_i \cap Y_i = \varnothing$.

Process:

```
1. Initialize $ARS = \varnothing$.
2. For each $N_i \in NL$, if $P_i \neq \varnothing$ and $Sup = |A_i|/|G| \geq MinSup$, do
3.     If $|P_i| = 1$ and $B_{i1} \neq \varnothing$, do
4.         Insert $((B_i - B_{i1}) \Rightarrow B_{i1}, Sup, 100\%)$ into ARS as a FCA-based exact rule.
5.     For each $N_{ij} \in P_i$, if $B_{ij} \neq \varnothing$ and $Conf = |A_i|/|A_{ij}| \geq MinConf$, do
6.         Insert $(B_{ij} \Rightarrow (B_i - B_{ij}), Sup, Conf)$ into ARS as a FCA-based approximate rule.
7. Return ARS.
```

The R implementation

Please take a look at the R codes file `ch_10_fca.R` from the bundle of R codes for the above algorithms. The codes can be tested with the following command:

```
> source("ch_10_fca.R")
```

Time for action

Here are some questions for you to know whether you have understood the concepts:

1. What are the characteristics of text mining?
2. What is the difference between text mining and data mining?
3. What is the difference between web mining and data mining?

Summary

In this chapter, we looked at text mining, which is to find brand new or unknown information by extracting information from the documents dataset. We also looked at how text summarization presents the condensed result of the document set under research; in other words, takes a source, extracts contents, and presents the key contents in a condensed format that is sensitive to the final needs. Also, we looked at how genre classification discriminates between documents by form, style, and the targeted system or audience. We also covered the question answering system, topic detection, and web mining.

In this book, many useful data-mining algorithms are illustrated in the form of the R language, which has been there for years, even decades. The most popular algorithms are included with a detailed description. You can start working with the knowledge structure of the classical and living data-mining algorithms and solutions built with R.

Algorithms and Data Structures

Here is a list of algorithms related to association rules mining; it is only a small portion of the available algorithms, but it has proved to be effective:

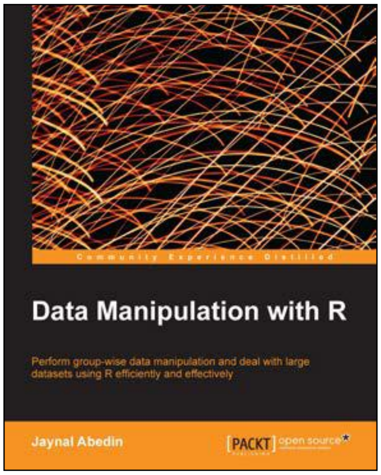
Approach	Dataset	Sequential pattern mining	Sequential rule mining	Frequent itemset mining	Association rule mining
Apriori	Transaction			Yes	
AprioriTid	Transaction			Yes	
DHP (**Direct Hashing and Pruning**)	Transaction			Yes	
FDM (**Fast Distributed Mining of association rules**)	Transaction			Yes	
GSP (**Generalized Sequential Patterns**)	Sequence	Yes			
DIC	Transaction			Yes	
Pincer Search (**the Pincer-search algorithm**)	Transaction			Yes	
CARMA (**Continuous Association Rule Mining Algorithm**)	Transaction			Yes	

Approach	**Dataset**	**Sequential pattern mining**	**Sequential rule mining**	**Frequent itemset mining**	**Association rule mining**
CHARM (**Closed Association Rule Mining**)	Transaction			Yes (closed)	
Depth-project	Transaction			Yes (maximal)	
Eclat	Transaction			Yes	
SPAD	Sequence	Yes			
SPAM	Sequence	Yes			
Diffset	Transaction			Yes	
FP-growth	Transaction			Yes	FP-growth
DSM-FI (**Data Stream Mining for Frequent Itemsets**)	Transaction			Yes	
PRICES	Transaction			Yes	
PrefixSpan	Sequence	Yes			
Sporadic Rules	Transaction				Yes
IGB	Transaction				Yes
GenMax	Transaction			Yes (maximal)	
FPMax (**Frequent Maximal Item Set**)	Transaction			Yes	
FHARM (**Fuzzy Healthy Association Rule Mining**)	Transaction			Yes	
H-Mine	Transaction			Yes	
FHSAR	Transaction				Yes
Reverse Apriori	Transaction			Yes (maximal)	
DTFIM	Transaction			Yes	
GIT tree	Transaction			Yes	
Scaling Apriori	Transaction			Yes	
CMRules	Sequence		Yes		
Minimum effort	Transaction			Yes (maximal)	

Approach	**Dataset**	**Sequential pattern mining**	**Sequential rule mining**	**Frequent itemset mining**	**Association rule mining**
TopSeqRules	Sequence		Yes		
FPG ARM	Transaction			Yes	
TNR	Transaction				Yes
ClaSP	Sequence	Yes (closed)			

Index

[PACKT] open source
community experience distilled
PUBLISHING

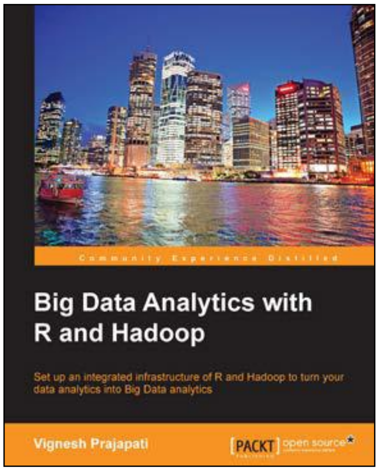

Community Experience Distilled
Data Manipulation with R
Perform group-wise data manipulation and deal with large datasets using R efficiently and effectively
Jaynal Abedin
[PACKT] open source

Data Manipulation with R

ISBN: 978-1-78328-109-1 Paperback: 102 pages

Perform group-wise data manipulation and deal with large datasets using R efficiently and effectively

1. Perform factor manipulation and string processing.
2. Learn group-wise data manipulation using plyr.
3. Handle large datasets, interact with database software, and manipulate data using sqldf.


Community Experience Distilled
Big Data Analytics with R and Hadoop
Set up an integrated infrastructure of R and Hadoop to turn your data analytics into Big Data analytics
Vignesh Prajapati
[PACKT] open source

Big Data Analytics with R and Hadoop

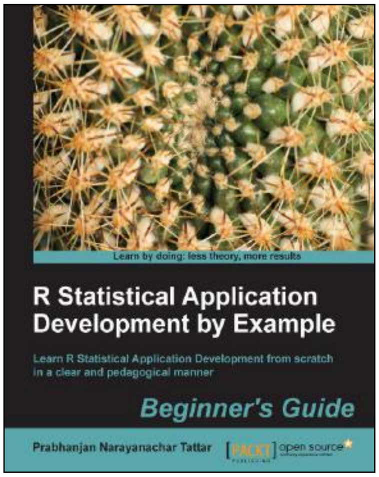
ISBN: 978-1-78216-328-2 Paperback: 328 pages

Set up an integrated infrastructure of R and Hadoop to turn your data analytics into Big Data analytics

1. Write Hadoop MapReduce within R.
2. Learn data analytics with R and the Hadoop platform.
3. Handle HDFS data within R.
4. Understand Hadoop streaming with R.
5. Encode and enrich datasets into R.

Please check **www.PacktPub.com** for information on our titles

[PACKT] open source
PUBLISHING
community experience distilled


Learn by doing: less theory, more results
R Statistical Application Development by Example
Learn R Statistical Application Development from scratch in a clear and pedagogical manner
Beginner's Guide
Prabhanjan Narayanachar Tattar
[PACKT] open source

R Statistical Application Development by Example Beginner's Guide

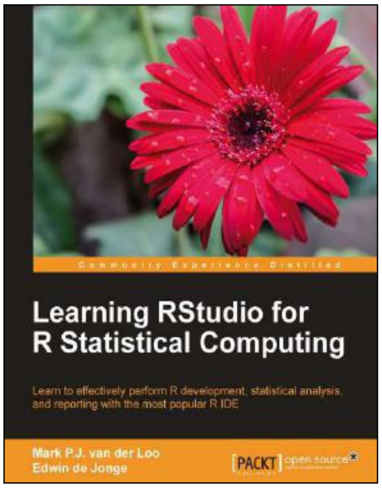
ISBN: 978-1-84951-944-1 Paperback: 344 pages

Learn R Statistical Application Development from scratch in a clear and pedagogical manner

1. A self-learning guide for the user who needs statistical tools for understanding uncertainty in computer science data.

2. Essential descriptive statistics, effective data visualization, and efficient model building.

3. Every method explained through real data sets enables clarity and confidence for unforeseen scenarios.


Community Experience Distilled
Learning RStudio for R Statistical Computing
Learn to effectively perform R development, statistical analysis, and reporting with the most popular R IDE
Mark P.J. van der Loo
Edwin de Jonge
[PACKT] open source

Learning RStudio for R Statistical Computing

ISBN: 978-1-78216-060-1 Paperback: 126 pages

Learn to effectively perform R development, statistical analysis, and reporting with the most popular R IDE

1. A complete practical tutorial for RStudio, designed keeping in mind the needs of analysts and R developers alike.

2. Step-by-step examples that apply the principles of reproducible research and good programming practices to R projects.

3. Learn to effectively generate reports, create graphics, and perform analysis, and even build R-packages with RStudio.

Please check **www.PacktPub.com** for information on our titles