Chapter 1

Data Mining

In this intoductory chapter we begin with the essence of data mining and a discussion of how data mining is treated by the various disciplines that contribute to this field. We cover "Bonferroni's Principle," which is really a warning about overusing the ability to mine data. This chapter is also the place where we summarize a few useful ideas that are not data mining but are useful in understanding some important data-mining concepts. These include the TF.IDF measure of word importance, behavior of hash functions and indexes, and identities involving e, the base of natural logarithms. Finally, we give an outline of the topics covered in the balance of the book.

1.1 What is Data Mining?

The most commonly accepted definition of "data mining" is the discovery of "models" for data. A "model," however, can be one of several things. We mention below the most important directions in modeling.

1.1.1 Statistical Modeling

Statisticians were the first to use the term "data mining." Originally, "data mining" or "data dredging" was a derogatory term referring to attempts to extract information that was not supported by the data. Section 1.2 illustrates the sort of errors one can make by trying to extract what really isn't in the data. Today, "data mining" has taken on a positive meaning. Now, statisticians view data mining as the construction of a *statistical model*, that is, an underlying distribution from which the visible data is drawn.

Example 1.1: Suppose our data is a set of numbers. This data is much simpler than data that would be data-mined, but it will serve as an example. A statistician might decide that the data comes from a Gaussian distribution and use a formula to compute the most likely parameters of this Gaussian. The mean

and standard deviation of this Gaussian distribution completely characterize the distribution and would become the model of the data. □

1.1.2 Machine Learning

There are some who regard data mining as synonymous with machine learning. There is no question that some data mining appropriately uses algorithms from machine learning. Machine-learning practitioners use the data as a training set, to train an algorithm of one of the many types used by machine-learning practitioners, such as Bayes nets, support-vector machines, decision trees, hidden Markov models, and many others.

There are situations where using data in this way makes sense. The typical case where machine learning is a good approach is when we have little idea of what we are looking for in the data. For example, it is rather unclear what it is about movies that makes certain movie-goers like or dislike it. Thus, in answering the "Netflix challenge" to devise an algorithm that predicts the ratings of movies by users, based on a sample of their responses, machine-learning algorithms have proved quite successful. We shall discuss a simple form of this type of algorithm in Section 9.4.

On the other hand, machine learning has not proved successful in situations where we can describe the goals of the mining more directly. An interesting case in point is the attempt by WhizBang! Labs[1] to use machine learning to locate people's resumes on the Web. It was not able to do better than algorithms designed by hand to look for some of the obvious words and phrases that appear in the typical resume. Since everyone who has looked at or written a resume has a pretty good idea of what resumes contain, there was no mystery about what makes a Web page a resume. Thus, there was no advantage to machine-learning over the direct design of an algorithm to discover resumes.

1.1.3 Computational Approaches to Modeling

More recently, computer scientists have looked at data mining as an algorithmic problem. In this case, the model of the data is simply the answer to a complex query about it. For instance, given the set of numbers of Example 1.1, we might compute their average and standard deviation. Note that these values might not be the parameters of the Gaussian that best fits the data, although they will almost certainly be very close if the size of the data is large.

There are many different approaches to modeling data. We have already mentioned the possibility of constructing a statistical process whereby the data could have been generated. Most other approaches to modeling can be described as either

1. Summarizing the data succinctly and approximately, or

[1]This startup attempted to use machine learning to mine large-scale data, and hired many of the top machine-learning people to do so. Unfortunately, it was not able to survive.

2. Extracting the most prominent features of the data and ignoring the rest.

We shall explore these two approaches in the following sections.

1.1.4 Summarization

One of the most interesting forms of summarization is the PageRank idea, which made Google successful and which we shall cover in Chapter 5. In this form of Web mining, the entire complex structure of the Web is summarized by a single number for each page. This number, the "PageRank" of the page, is (oversimplifying somewhat) the probability that a random walker on the graph would be at that page at any given time. The remarkable property this ranking has is that it reflects very well the "importance" of the page – the degree to which typical searchers would like that page returned as an answer to their search query.

Another important form of summary – clustering – will be covered in Chapter 7. Here, data is viewed as points in a multidimensional space. Points that are "close" in this space are assigned to the same cluster. The clusters themselves are summarized, perhaps by giving the centroid of the cluster and the average distance from the centroid of points in the cluster. These cluster summaries become the summary of the entire data set.

Example 1.2: A famous instance of clustering to solve a problem took place long ago in London, and it was done entirely without computers.[2] The physician John Snow, dealing with a Cholera outbreak plotted the cases on a map of the city. A small illustration suggesting the process is shown in Fig. 1.1.

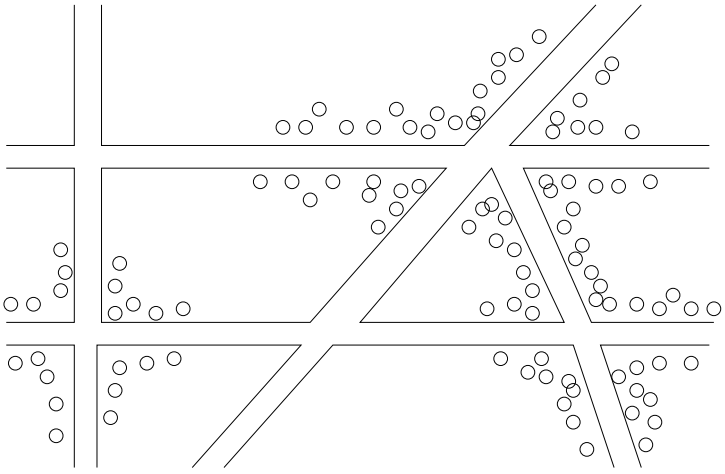

Figure 1.1: Plotting cholera cases on a map of London

[2]See `http://en.wikipedia.org/wiki/1854_Broad_Street_cholera_outbreak`.

The cases clustered around some of the intersections of roads. These intersections were the locations of wells that had become contaminated; people who lived nearest these wells got sick, while people who lived nearer to wells that had not been contaminated did not get sick. Without the ability to cluster the data, the cause of Cholera would not have been discovered. □

1.1.5 Feature Extraction

The typical feature-based model looks for the most extreme examples of a phenomenon and represents the data by these examples. If you are familiar with Bayes nets, a branch of machine learning and a topic we do not cover in this book, you know how a complex relationship between objects is represented by finding the strongest statistical dependencies among these objects and using only those in representing all statistical connections. Some of the important kinds of feature extraction from large-scale data that we shall study are:

1. *Frequent Itemsets.* This model makes sense for data that consists of "baskets" of small sets of items, as in the market-basket problem that we shall discuss in Chapter 6. We look for small sets of items that appear together in many baskets, and these "frequent itemsets" are the characterization of the data that we seek. The original application of this sort of mining was true market baskets: the sets of items, such as hamburger and ketchup, that people tend to buy together when checking out at the cash register of a store or super market.

2. *Similar Items.* Often, 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. An example is treating customers at an online store like Amazon as the set of items they have bought. In order for Amazon to recommend something else they might like, Amazon can look for "similar" customers and recommend something many of these customers have bought. This process is called "collaborative filtering." If customers were single-minded, that is, they bought only one kind of thing, then clustering customers might work. However, since customers tend to have interests in many different things, it is more useful to find, for each customer, a small number of other customers who are similar in their tastes, and represent the data by these connections. We discuss similarity in Chapter 3.

1.2 Statistical Limits on Data Mining

A common sort of data-mining problem involves discovering unusual events hidden within massive amounts of data. This section is a discussion of the problem, including "Bonferroni's Principle," a warning against overzealous use of data mining.

1.2.1 Total Information Awareness

In 2002, the Bush administration put forward a plan to mine all the data it could find, including credit-card receipts, hotel records, travel data, and many other kinds of information in order to track terrorist activity. This idea naturally caused great concern among privacy advocates, and the project, called TIA, or *Total Information Awareness*, was eventually killed by Congress, although it is unclear whether the project in fact exists under another name. It is not the purpose of this book to discuss the difficult issue of the privacy-security tradeoff. However, the prospect of TIA or a system like it does raise technical questions about its feasibility and the realism of its assumptions.

The concern raised by many is that if you look at so much data, and you try to find within it activities that look like terrorist behavior, are you not going to find many innocent activities – or even illicit activities that are not terrorism – that will result in visits from the police and maybe worse than just a visit? The answer is that it all depends on how narrowly you define the activities that you look for. Statisticians have seen this problem in many guises and have a theory, which we introduce in the next section.

1.2.2 Bonferroni's Principle

Suppose you have a certain amount of data, and you look for events of a certain type within that data. You can expect events of this type to occur, even if the data is completely random, and the number of occurrences of these events will grow as the size of the data grows. These occurrences are "bogus," in the sense that they have no cause other than that random data will always have some number of unusual features that look significant but aren't. A theorem of statistics, known as the *Bonferroni correction* gives a statistically sound way to avoid most of these bogus positive responses to a search through the data. Without going into the statistical details, we offer an informal version, *Bonferroni's principle*, that helps us avoid treating random occurrences as if they were real. Calculate the expected number of occurrences of the events you are looking for, on the assumption that data is random. If this number is significantly larger than the number of real instances you hope to find, then you must expect almost anything you find to be bogus, i.e., a statistical artifact rather than evidence of what you are looking for. This observation is the informal statement of Bonferroni's principle.

In a situation like searching for terrorists, where we expect that there are few terrorists operating at any one time, Bonferroni's principle says that we may only detect terrorists by looking for events that are so rare that they are unlikely to occur in random data. We shall give an extended example in the next section.

1.2.3 An Example of Bonferroni's Principle

Suppose there are believed to be some "evil-doers" out there, and we want to detect them. Suppose further that we have reason to believe that periodically, evil-doers gather at a hotel to plot their evil. Let us make the following assumptions about the size of the problem:

1. There are one billion people who might be evil-doers.
2. Everyone goes to a hotel one day in 100.
3. A hotel holds 100 people. Hence, there are 100,000 hotels – enough to hold the 1% of a billion people who visit a hotel on any given day.
4. We shall examine hotel records for 1000 days.

To find evil-doers in this data, we shall look for people who, on two different days, were both at the same hotel. Suppose, however, that there really are no evil-doers. That is, everyone behaves at random, deciding with probability 0.01 to visit a hotel on any given day, and if so, choosing one of the 10^5 hotels at random. Would we find any pairs of people who appear to be evil-doers?

We can do a simple approximate calculation as follows. The probability of any two people both deciding to visit a hotel on any given day is .0001. The chance that they will visit the same hotel is this probability divided by 10^5, the number of hotels. Thus, the chance that they will visit the same hotel on one given day is 10^{-9}. The chance that they will visit the same hotel on two different given days is the square of this number, 10^{-18}. Note that the hotels can be different on the two days.

Now, we must consider how many events will indicate evil-doing. An "event" in this sense is a pair of people and a pair of days, such that the two people were at the same hotel on each of the two days. To simplify the arithmetic, note that for large n, $\binom{n}{2}$ is about $n^2/2$. We shall use this approximation in what follows. Thus, the number of pairs of people is $\binom{10^9}{2} = 5 \times 10^{17}$. The number of pairs of days is $\binom{1000}{2} = 5 \times 10^5$. The expected number of events that look like evil-doing is the product of the number of pairs of people, the number of pairs of days, and the probability that any one pair of people and pair of days is an instance of the behavior we are looking for. That number is

$$5 \times 10^{17} \times 5 \times 10^5 \times 10^{-18} = 250,000$$

That is, there will be a quarter of a million pairs of people who look like evil-doers, even though they are not.

Now, suppose there really are 10 pairs of evil-doers out there. The police will need to investigate a quarter of a million other pairs in order to find the real evil-doers. In addition to the intrusion on the lives of half a million innocent people, the work involved is sufficiently great that this approach to finding evil-doers is probably not feasible.

1.2.4 Exercises for Section 1.2

Exercise 1.2.1: Using the information from Section 1.2.3, what would be the number of suspected pairs if the following changes were made to the data (and all other numbers remained as they were in that section)?

(a) The number of days of observation was raised to 2000.

(b) The number of people observed was raised to 2 billion (and there were therefore 200,000 hotels).

(c) We only reported a pair as suspect if they were at the same hotel at the same time on three different days.

! **Exercise 1.2.2:** Suppose we have information about the supermarket purchases of 100 million people. Each person goes to the supermarket 100 times in a year and buys 10 of the 1000 items that the supermarket sells. We believe that a pair of terrorists will buy exactly the same set of 10 items (perhaps the ingredients for a bomb?) at some time during the year. If we search for pairs of people who have bought the same set of items, would we expect that any such people found were truly terrorists?[3]

1.3 Things Useful to Know

In this section, we offer brief introductions to subjects that you may or may not have seen in your study of other courses. Each will be useful in the study of data mining. They include:

1. The TF.IDF measure of word importance.
2. Hash functions and their use.
3. Secondary storage (disk) and its effect on running time of algorithms.
4. The base e of natural logarithms and identities involving that constant.
5. Power laws.

1.3.1 Importance of Words in Documents

In several applications of data mining, we shall be faced with the problem of categorizing documents (sequences of words) by their topic. Typically, topics are identified by finding the special words that characterize documents about that topic. For instance, articles about baseball would tend to have many occurrences of words like "ball," "bat," "pitch,", "run," and so on. Once we

[3]That is, assume our hypothesis that terrorists will surely buy a set of 10 items in common at some time during the year. We don't want to address the matter of whether or not terrorists would necessarily do so.

have classified documents to determine they are about baseball, it is not hard to notice that words such as these appear unusually frequently. However, until we have made the classification, it is not possible to identify these words as characteristic.

Thus, classification often starts by looking at documents, and finding the significant words in those documents. Our first guess might be that the words appearing most frequently in a document are the most significant. However, that intuition is exactly opposite of the truth. The most frequent words will most surely be the common words such as "the" or "and," which help build ideas but do not carry any significance themselves. In fact, the several hundred most common words in English (called *stop words*) are often removed from documents before any attempt to classify them.

In fact, the indicators of the topic are relatively rare words. However, not all rare words are equally useful as indicators. There are certain words, for example "notwithstanding" or "albeit," that appear rarely in a collection of documents, yet do not tell us anything useful. On the other hand, a word like "chukker" is probably equally rare, but tips us off that the document is about the sport of polo. The difference between rare words that tell us something and those that do not has to do with the concentration of the useful words in just a few documents. That is, the presence of a word like "albeit" in a document does not make it terribly more likely that it will appear multiple times. However, if an article mentions "chukker" once, it is likely to tell us what happened in the "first chukker," then the "second chukker," and so on. That is, the word is likely to be repeated if it appears at all.

The formal measure of how concentrated into relatively few documents are the occurrences of a given word is called TF.IDF (*Term Frequency times Inverse Document Frequency*). It is normally computed as follows. Suppose we have a collection of N documents. Define f_{ij} to be the *frequency* (number of occurrences) of term (word) i in document j. Then, define the *term frequency* TF_{ij} to be:

$$TF_{ij} = \frac{f_{ij}}{\max_k f_{kj}}$$

That is, the term frequency of term i in document j is f_{ij} normalized by dividing it by the maximum number of occurrences of any term (perhaps excluding stop words) in the same document. Thus, the most frequent term in document j gets a TF of 1, and other terms get fractions as their term frequency for this document.

The IDF for a term is defined as follows. Suppose term i appears in n_i of the N documents in the collection. Then $IDF_i = \log_2(N/n_i)$. The TF.IDF score for term i in document j is then defined to be $TF_{ij} \times IDF_i$. The terms with the highest TF.IDF score are often the terms that best characterize the topic of the document.

Example 1.3: Suppose our repository consists of $2^{20} = 1{,}048{,}576$ documents. Suppose word w appears in $2^{10} = 1024$ of these documents. Then $IDF_w =$

$\log_2(2^{20}/2^{10}) = \log 2(2^{10}) = 10$. Consider a document j in which w appears 20 times, and that is the maximum number of times in which any word appears (perhaps after eliminating stop words). Then $TF_{wj} = 1$, and the TF.IDF score for w in document j is 10.

Suppose that in document k, word w appears once, while the maximum number of occurrences of any word in this document is 20. Then $TF_{wk} = 1/20$, and the TF.IDF score for w in document k is $1/2$. □

1.3.2 Hash Functions

The reader has probably heard of hash tables, and perhaps used them in Java classes or similar packages. The hash functions that make hash tables feasible are also essential components in a number of data-mining algorithms, where the hash table takes an unfamiliar form. We shall review the basics here.

First, a hash function h takes a *hash-key* value as an argument and produces a *bucket number* as a result. The bucket number is an integer, normally in the range 0 to $B - 1$, where B is the number of buckets. Hash-keys can be of any type. There is an intuitive property of hash functions that they "randomize" hash-keys. To be precise, if hash-keys are drawn randomly from a reasonable population of possible hash-keys, then h will send approximately equal numbers of hash-keys to each of the B buckets. It would be impossible to do so if, for example, the population of possible hash-keys were smaller than B. Such a population would not be "reasonable." However, there can be more subtle reasons why a hash function fails to achieve an approximately uniform distribution into buckets.

Example 1.4: Suppose hash-keys are positive integers. A common and simple hash function is to pick $h(x) = x \mod B$, that is, the remainder when x is divided by B. That choice works fine if our population of hash-keys is all positive integers. $1/B$th of the integers will be assigned to each of the buckets. However, suppose our population is the even integers, and $B = 10$. Then only buckets 0, 2, 4, 6, and 8 can be the value of $h(x)$, and the hash function is distinctly nonrandom in its behavior. On the other hand, if we picked $B = 11$, then we would find that 1/11th of the even integers get sent to each of the 11 buckets, so the hash function would work very well. □

The generalization of Example 1.4 is that when hash-keys are integers, chosing B so it has any common factor with all (or even most of) the possible hash-keys will result in nonrandom distribution into buckets. Thus, it is normally preferred that we choose B to be a prime. That choice reduces the chance of nonrandom behavior, although we still have to consider the possibility that all hash-keys have B as a factor. Of course there are many other types of hash functions not based on modular arithmetic. We shall not try to summarize the options here, but some sources of information will be mentioned in the bibliographic notes.

What if hash-keys are not integers? In a sense, all data types have values that are composed of bits, and sequences of bits can always be interpreted as integers. However, there are some simple rules that enable us to convert common types to integers. For example, if hash-keys are strings, convert each character to its ASCII or Unicode equivalent, which can be interpreted as a small integer. Sum the integers before dividing by B. As long as B is smaller than the typical sum of character codes for the population of strings, the distribution into buckets will be relatively uniform. If B is larger, then we can partition the characters of a string into groups of several characters each. Treat the concatenation of the codes for the characters of a group as a single integer. Sum the integers associated with all the groups of a string, and divide by B as before. For instance, if B is around a billion, or 2^{30}, then grouping characters four at a time will give us 32-bit integers. The sum of several of these will distribute fairly evenly into a billion buckets.

For more complex data types, we can extend the idea used for converting strings to integers, recursively.

- For a type that is a record, each of whose components has its own type, recursively convert the value of each component to an integer, using the algorithm appropriate for the type of that component. Sum the integers for the components, and convert the integer sum to buckets by dividing by B.

- For a type that is an array, set, or bag of elements of some one type, convert the values of the elements' type to integers, sum the integers, and divide by B.

1.3.3 Indexes

An *index* is a data structure that makes it efficient to retrieve objects given the value of one or more elements of those objects. The most common situation is one where the objects are records, and the index is on one of the fields of that record. Given a value v for that field, the index lets us retrieve all the records with value v in that field. For example, we could have a file of (name, address, phone) triples, and an index on the phone field. Given a phone number, the index allows us to find quickly the record or records with that phone number.

There are many ways to implement indexes, and we shall not attempt to survey the matter here. The bibliographic notes give suggestions for further reading. However, a hash table is one simple way to build an index. The field or fields on which the index is based form the hash-key for a hash function. Records have the hash function applied to value of the hash-key, and the record itself is placed in the bucket whose number is determined by the hash function. The bucket could be a list of records in main-memory, or a disk block, for example.

Then, given a hash-key value, we can hash it, find the bucket, and need to search only that bucket to find the records with that value for the hash-key. If we choose the number of buckets B to be comparable to the number of records in the file, then there will be relatively few records in any bucket, and the search of a bucket takes little time.

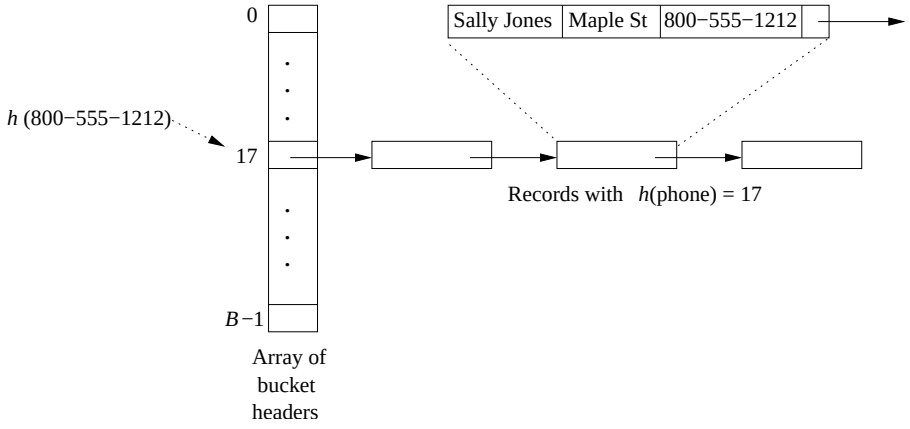

Figure 1.2: A hash table used as an index; phone numbers are hashed to buckets, and the entire record is placed in the bucket whose number is the hash value of the phone

Example 1.5: Figure 1.2 suggests what a main-memory index of records with name, address, and phone fields might look like. Here, the index is on the phone field, and buckets are linked lists. We show the phone 800-555-1212 hashed to bucket number 17. There is an array of *bucket headers*, whose ith element is the head of a linked list for the bucket numbered i. We show expanded one of the elements of the linked list. It contains a record with name, address, and phone fields. This record is in fact one with the phone number 800-555-1212. Other records in that bucket may or may not have this phone number. We only know that whatever phone number they have is a phone that hashes to 17. □

1.3.4 Secondary Storage

It is important, when dealing with large-scale data, that we have a good understanding of the difference in time taken to perform computations when the data is initially on disk, as opposed to the time needed if the data is initially in main memory. The physical characteristics of disks is another subject on which we could say much, but shall say only a little and leave the interested reader to follow the bibliographic notes.

Disks are organized into *blocks*, which are the minimum units that the operating system uses to move data between main memory and disk. For example,

the Windows operating system uses blocks of 64K bytes (i.e., $2^{16} = 65{,}536$ bytes to be exact). It takes approximately ten milliseconds to *access* (move the disk head to the track of the block and wait for the block to rotate under the head) and read a disk block. That delay is at least five orders of magnitude (a factor of 10^5) slower than the time taken to read a word from main memory, so if all we want to do is access a few bytes, there is an overwhelming benefit to having data in main memory. In fact, if we want to do something simple to every byte of a disk block, e.g., treat the block as a bucket of a hash table and search for a particular value of the hash-key among all the records in that bucket, then the time taken to move the block from disk to main memory will be far larger than the time taken to do the computation.

By organizing our data so that related data is on a single *cylinder* (the collection of blocks reachable at a fixed radius from the center of the disk, and therefore accessible without moving the disk head), we can read all the blocks on the cylinder into main memory in considerably less than 10 milliseconds per block. You can assume that a disk cannot transfer data to main memory at more than a hundred million bytes per second, no matter how that data is organized. That is not a problem when your dataset is a megabyte. But a dataset of a hundred gigabytes or a terabyte presents problems just accessing it, let alone doing anything useful with it.

1.3.5 The Base of Natural Logarithms

The constant $e = 2.7182818\cdots$ has a number of useful special properties. In particular, e is the limit of $(1 + \frac{1}{x})^x$ as x goes to infinity. The values of this expression for $x = 1, 2, 3, 4$ are approximately $2, 2.25, 2.37, 2.44$, so you should find it easy to believe that the limit of this series is around 2.72.

Some algebra lets us obtain approximations to many seemingly complex expressions. Consider $(1+a)^b$, where a is small. We can rewrite the expression as $(1+a)^{(1/a)(ab)}$. Then substitute $a = 1/x$ and $1/a = x$, so we have $(1+\frac{1}{x})^{x(ab)}$, which is

$$\Big((1 + \frac{1}{x})^x\Big)^{ab}$$

Since a is assumed small, x is large, so the subexpression $(1 + \frac{1}{x})^x$ will be close to the limiting value of e. We can thus approximate $(1 + a)^b$ as e^{ab}.

Similar identities hold when a is negative. That is, the limit as x goes to infinity of $(1 - \frac{1}{x})^x$ is $1/e$. It follows that the approximation $(1 + a)^b = e^{ab}$ holds even when a is a small negative number. Put another way, $(1 - a)^b$ is approximately e^{-ab} when a is small and b is large.

Some other useful approximations follow from the Taylor expansion of e^x. That is, $e^x = \sum_{i=0}^{\infty} x^i/i!$, or $e^x = 1 + x + x^2/2 + x^3/6 + x^4/24 + \cdots$. When x is large, the above series converges slowly, although it does converge because $n!$ grows faster than x^n for any constant x. However, when x is small, either positive or negative, the series converges rapidly, and only a few terms are necessary to get a good approximation.

Example 1.6: Let $x = 1/2$. Then

$$e^{1/2} = 1 + \frac{1}{2} + \frac{1}{8} + \frac{1}{48} + \frac{1}{384} + \cdots$$

or approximately $e^{1/2} = 1.64844$.

Let $x = -1$. Then

$$e^{-1} = 1 - 1 + \frac{1}{2} - \frac{1}{6} + \frac{1}{24} - \frac{1}{120} + \frac{1}{720} - \frac{1}{5040} + \cdots$$

or approximately $e^{-1} = 0.36786$. □

1.3.6 Power Laws

There are many phenomena that relate two variables by a *power law*, that is, a linear relationship between the logarithms of the variables. Figure 1.3 suggests such a relationship. If x is the horizontal axis and y is the vertical axis, then the relationship is $\log_{10} y = 6 - 2\log_{10} x$.

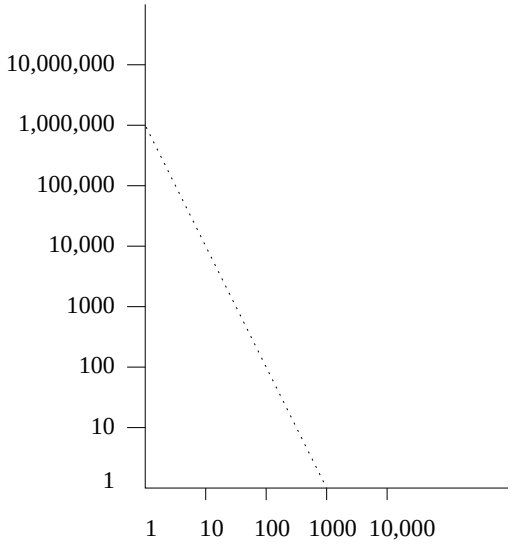

Figure 1.3: A power law with a slope of -2

Example 1.7: We might examine book sales at Amazon.com, and let x represent the rank of books by sales. Then y is the number of sales of the xth best-selling book over some period. The implication of the graph of Fig. 1.3 would be that the best-selling book sold 1,000,000 copies, the 10th best-selling book sold 10,000 copies, the 100th best-selling book sold 100 copies, and so on for all ranks between these numbers and beyond. The implication that above

The Matthew Effect

Often, the existence of power laws with values of the exponent higher than 1 are explained by the *Matthew effect*. In the biblical *Book of Matthew*, there is a verse about "the rich get richer." Many phenomena exhibit this behavior, where getting a high value of some property causes that very property to increase. For example, if a Web page has many links in, then people are more likely to find the page and may choose to link to it from one of their pages as well. As another example, if a book is selling well on Amazon, then it is likely to be advertised when customers go to the Amazon site. Some of these people will choose to buy the book as well, thus increasing the sales of this book.

rank 1000 the sales are a fraction of a book is too extreme, and we would in fact expect the line to flatten out for ranks much higher than 1000. □

The general form of a power law relating x and y is $\log y = b + a \log x$. If we raise the base of the logarithm (which doesn't actually matter), say e, to the values on both sides of this equation, we get $y = e^b e^{a \log x} = e^b x^a$. Since e^b is just "some constant," let us replace it by constant c. Thus, a power law can be written as $y = cx^a$ for some constants a and c.

Example 1.8: In Fig. 1.3 we see that when $x = 1$, $y = 10^6$, and when $x = 1000$, $y = 1$. Making the first substitution, we see $10^6 = c$. The second substitution gives us $1 = c(1000)^a$. Since we now know $c = 10^6$, the second equation gives us $1 = 10^6(1000)^a$, from which we see $a = -2$. That is, the law expressed by Fig. 1.3 is $y = 10^6 x^{-2}$, or $y = 10^6/x^2$. □

We shall meet in this book many ways that power laws govern phenomena. Here are some examples:

1. *Node Degrees in the Web Graph*: Order all pages by the number of in-links to that page. Let x be the position of a page in this ordering, and let y be the number of in-links to the xth page. Then y as a function of x looks very much like Fig. 1.3. The exponent a is slightly larger than the -2 shown there; it has been found closer to 2.1.

2. *Sales of Products*: Order products, say books at Amazon.com, by their sales over the past year. Let y be the number of sales of the xth most popular book. Again, the function $y(x)$ will look something like Fig. 1.3. we shall discuss the consequences of this distribution of sales in Section 9.1.2, where we take up the matter of the "long tail."

3. *Sizes of Web Sites*: Count the number of pages at Web sites, and order sites by the number of their pages. Let y be the number of pages at the xth site. Again, the function $y(x)$ follows a power law.

4. *Zipf's Law*: This power law originally referred to the frequency of words in a collection of documents. If you order words by frequency, and let y be the number of times the xth word in the order appears, then you get a power law, although with a much shallower slope than that of Fig. 1.3. Zipf's observation was that $y = cx^{-1/2}$. Interestingly, a number of other kinds of data follow this particular power law. For example, if we order states in the US by population and let y be the population of the xth most populous state, then x and y obey Zipf's law approximately.

1.3.7 Exercises for Section 1.3

Exercise 1.3.1: Suppose there is a repository of ten million documents. What (to the nearest integer) is the IDF for a word that appears in (a) 40 documents (b) 10,000 documents?

Exercise 1.3.2: Suppose there is a repository of ten million documents, and word w appears in 320 of them. In a particular document d, the maximum number of occurrences of a word is 15. Approximately what is the TF.IDF score for w if that word appears (a) once (b) five times?

! **Exercise 1.3.3:** Suppose hash-keys are drawn from the population of all non-negative integers that are multiples of some constant c, and hash function $h(x)$ is $x \mod 15$. For what values of c will h be a suitable hash function, i.e., a large random choice of hash-keys will be divided roughly equally into buckets?

Exercise 1.3.4: In terms of e, give approximations to

$$\text{(a) } (1.01)^{500} \text{ (b) } (1.05)^{1000} \text{ (c) } (0.9)^{40}$$

Exercise 1.3.5: Use the Taylor expansion of e^x to compute, to three decimal places: (a) $e^{1/10}$ (b) $e^{-1/10}$ (c) e^2.

1.4 Outline of the Book

This section gives brief summaries of the remaining chapters of the book.

Chapter 2 is not about data mining per se. Rather, it introduces us to the MapReduce methodology for exploiting parallelism in computing clouds (racks of interconnected processors). There is reason to believe that cloud computing, and MapReduce in particular, will become the normal way to compute when analysis of very large amounts of data is involved. A pervasive issue in later chapters will be the exploitation of the MapReduce methodology to implement the algorithms we cover.

Chapter 3 is about finding similar items. Our starting point is that items can be represented by sets of elements, and similar sets are those that have a large fraction of their elements in common. The key techniques of minhashing and locality-sensitive hashing are explained. These techniques have numerous

applications and often give surprisingly efficient solutions to problems that appear impossible for massive data sets.

In Chapter 4, we consider data in the form of a stream. The difference between a stream and a database is that the data in a stream is lost if you do not do something about it immediately. Important examples of streams are the streams of search queries at a search engine or clicks at a popular Web site. In this chapter, we see several of the surprising applications of hashing that make management of stream data feasible.

Chapter 5 is devoted to a single application: the computation of PageRank. This computation is the idea that made Google stand out from other search engines, and it is still an essential part of how search engines know what pages the user is likely to want to see. Extensions of PageRank are also essential in the fight against spam (euphemistically called "search engine optimization"), and we shall examine the latest extensions of the idea for the purpose of combating spam.

Then, Chapter 6 introduces the market-basket model of data, and its canonical problems of association rules and finding frequent itemsets. In the market-basket model, data consists of a large collection of baskets, each of which contains a small set of items. We give a sequence of algorithms capable of finding all frequent pairs of items, that is pairs of items that appear together in many baskets. Another sequence of algorithms are useful for finding most of the frequent itemsets larger than pairs, with high efficiency.

Chapter 7 examines the problem of clustering. We assume a set of items with a distance measure defining how close or far one item is from another. The goal is to examine a large amount of data and partition it into subsets (clusters), each cluster consisting of items that are all close to one another, yet far from items in the other clusters.

Chapter 8 is devoted to on-line advertising and the computational problems it engenders. We introduce the notion of an on-line algorithm – one where a good response must be given immediately, rather than waiting until we have seen the entire dataset. The idea of competitive ratio is another important concept covered in this chapter; it is the ratio of the guaranteed performance of an on-line algorithm compared with the performance of the optimal algorithm that is allowed to see all the data before making any decisions. These ideas are used to give good algorithms that match bids by advertisers for the right to display their ad in response to a query against the search queries arriving at a search engine.

Chapter 9 is devoted to recommendation systems. Many Web applications involve advising users on what they might like. The Netflix challenge is one example, where it is desired to predict what movies a user would like, or Amazon's problem of pitching a product to a customer based on information about what they might be interested in buying. There are two basic approaches to recommendation. We can characterize items by features, e.g., the stars of a movie, and recommend items with the same features as those the user is known to like. Or, we can look at other users with preferences similar to that of the

user in question, and see what they liked (a technique known as collaborative filtering).

In Chapter 10, we study social networks and algorithms for their analysis. The canonical example of a social network is the graph of Facebook friends, where the nodes are people, and edges connect two people if they are friends. Directed graphs, such as followers on Twitter, can also be viewed as social networks. A common example of a problem to be addressed is identifying "communities," that is, small sets of nodes with an unusually large number of edges among them. Other questions about social networks are general questions about graphs, such as computing the transitive closure or diameter of a graph, but are made more difficult by the size of typical networks.

Chapter 11 looks at dimensionality reduction. We are given a very large matrix, typically sparse. Think of the matrix as representing a relationship between two kinds of entities, e.g., ratings of movies by viewers. Intuitively, there are a small number of concepts, many fewer concepts than there are movies or viewers, that explain why certain viewers like certain movies. We offer several algorithms that simplify matrices by decomposing them into a product of matrices that are much smaller in one of the two dimensions. One matrix relates entities of one kind to the small number of concepts and another relates the concepts to the other kind of entity. If done correctly, the product of the smaller matrices will be very close to the original matrix.

Finally, Chapter 12 discusses algorithms for machine learning from very large datasets. Techniques covered include perceptrons, support-vector machines, finding models by gradient descent, nearest-neighbor models, and decision trees.

1.5 Summary of Chapter 1

- *Data Mining*: This term refers to the process of extracting useful models of data. Sometimes, a model can be a summary of the data, or it can be the set of most extreme features of the data.
- *Bonferroni's Principle*: If we are willing to view as an interesting feature of data something of which many instances can be expected to exist in random data, then we cannot rely on such features being significant. This observation limits our ability to mine data for features that are not sufficiently rare in practice.
- *TF.IDF*: The measure called TF.IDF lets us identify words in a collection of documents that are useful for determining the topic of each document. A word has high TF.IDF score in a document if it appears in relatively few documents, but appears in this one, and when it appears in a document it tends to appear many times.
- *Hash Functions*: A hash function maps hash-keys of some data type to integer bucket numbers. A good hash function distributes the possible

hash-key values approximately evenly among buckets. Any data type can be the domain of a hash function.

- *Indexes*: An index is a data structure that allows us to store and retrieve data records efficiently, given the value in one or more of the fields of the record. Hashing is one way to build an index.
- *Storage on Disk*: When data must be stored on disk (secondary memory), it takes very much more time to access a desired data item than if the same data were stored in main memory. When data is large, it is important that algorithms strive to keep needed data in main memory.
- *Power Laws*: Many phenomena obey a law that can be expressed as $y = cx^a$ for some power a, often around -2. Such phenomena include the sales of the xth most popular book, or the number of in-links to the xth most popular page.

1.6 References for Chapter 1

[7] is a clear introduction to the basics of data mining. [2] covers data mining principally from the point of view of machine learning and statistics.

For construction of hash functions and hash tables, see [4]. Details of the TF.IDF measure and other matters regarding document processing can be found in [5]. See [3] for more on managing indexes, hash tables, and data on disk.

Power laws pertaining to the Web were explored by [1]. The Matthew effect was first observed in [6].

1. A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Weiner, "Graph structure in the web," *Computer Networks* **33**:1–6, pp. 309–320, 2000.
2. M.M. Gaber, *Scientific Data Mining and Knowledge Discovery — Principles and Foundations*, Springer, New York, 2010.
3. H. Garcia-Molina, J.D. Ullman, and J. Widom, *Database Systems: The Complete Book* Second Edition, Prentice-Hall, Upper Saddle River, NJ, 2009.
4. D.E. Knuth, *The Art of Computer Programming* Vol. 3 (*Sorting and Searching*), Second Edition, Addison-Wesley, Upper Saddle River, NJ, 1998.
5. C.P. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*, Cambridge Univ. Press, 2008.
6. R.K. Merton, "The Matthew effect in science," *Science* **159**:3810, pp. 56–63, Jan. 5, 1968.

7. P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*, Addison-Wesley, Upper Saddle River, NJ, 2005.