1 Data Mining

In this introductory 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 *per se*, 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?

In the 1990s "data mining" was an exciting and popular new concept. Around 2010, people instead started to speak of "big data." Today, the popular term is "data science." However, during all this time, the concept remained the same: use the most powerful hardware, the most powerful programming systems, and the most efficient algorithms to solve problems in science, commerce, healthcare, government, the humanities, and many other fields of human endeavor.

1.1.1 Modeling

To many, data mining is the process of creating a model from data, often by the process of machine learning, which we mention in Section 1.1.3 and discuss more fully in Chapter 12. However, more generally, the objective of data mining is an algorithm. For instance, we discuss locality-sensitive hashing in Chapter 3 and a number of stream-mining algorithms in Chapter 4, none of which involve a model. Yet in many important applications, the hard part is creating the model, and once the model is available, the algorithm to use the model is straightforward.

EXAMPLE 1.1 Consider the problem of detecting emails that are phishing attacks. The most common approach is to build a model of phishing emails, perhaps by examining emails that people have recently reported as phishing attacks and looking for the words or phrases that appear unusually often in those emails, such as "Nigerian prince" or "verify account." The model could be weights on

words, with positive weights for words that appear frequently in phishing emails and negative weights for words that do not. Then the algorithm to detect phishing emails is simple. Apply the model to each email, that is, sum the weights of the words in that email, and say the email is phishing if and only if the sum is positive. Finding the best weights is a difficult problem, one we shall take up in Section 12.2. □

1.1.2 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 1.2 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. □

1.1.3 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 for machine-learning, such as Bayes nets, support-vector machines, decision trees, hidden Markov models, and a great variety of 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 the data says about the problem we are trying to solve. 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.

However, machine learning can be uncompetitive 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

[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.

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 be a resume. Thus, there was no advantage to machine-learning over the direct design of an algorithm to discover resumes.

Another problem with some machine-learning methods is that they often yield a model that, while it may be quite accurate, is not explainable. In some cases, explainability is not important. For example, if you ask Google why it has classified a gmail as spam, it usually says something like "it looks like other messages that people have identified as spam." That is, the email matches whatever model of spam Google has developed that day, undoubtedly using a technique from the arsenal of machine-learning algorithms. That explanation is probably satisfactory. We really don't care what Google does, as long as it makes the correct spam/not-spam decision.

On the other hand, consider an automobile-insurance company that creates a model of the risk associated with each driver and assigns different premiums to each, according to the model. If your premium goes up, you might well want an explanation of what the new model is doing and why it changed the estimate of your risk. Unfortunately, in many machine-learning methods, especially "deep learning," where the model involves layer upon layer of small elements, each of which makes a decision based on inputs from the previous layer, it may not be possible to give a coherent explanation of what the model is doing.

1.1.4 Computational Approaches to Modeling

In contrast to the statisical approach, computer scientists tend to look at data mining as an algorithmic problem. In this case, a model of the data is simply the answer to a complex query about that data. For instance, given the set of numbers of Example 1.2, 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, and the source of the data is truly Gaussian.

There are many different approaches to modeling data. We have already mentioned the possibility of constructing a random 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

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

We shall explore these two approaches in the following sections.

1.1.5 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. Remarkably, this ranking 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 then become the summary of the entire data set.

EXAMPLE 1.3 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.

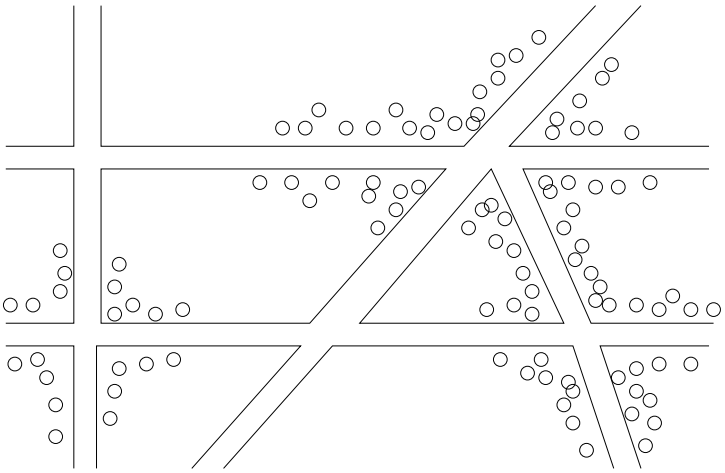

Figure 1.1 Plotting cholera cases on a map of London

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. □

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

1.1.6 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 on-line 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

Following the terrorist attack of Sept. 11, 2001, it was noticed that there were four people enrolled in different flight schools, learning how to pilot commercial aircraft, although they were not affiliated with any airline. It was conjectured that the information needed to predict and foil the attack was available in data,

but that there was then no way to examine the data and detect suspicious events. The response was a program called TIA, or *Total Information Awareness*, which was intended 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. Now *information integration* – the idea of relating and combining different data sources to obtain insights that are not available from any one source – is often a key step on the way to solving an important problem.

TIA naturally caused great concern among privacy advocates, and the project was eventually killed by Congress. 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 many technical questions about its feasibility. In this section, we wish to focus on one particular technical problem: if you look in your data for too many things at the same time, you will see things that look interesting, but are in fact simply statistical artifacts and have no significance. That is, if you search your data for 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, for avoiding this sort of error.

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 give an extended example below.

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 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.

[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.

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 1.4 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 1.5 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 well 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 well in this case. □

The generalization of Example 1.5 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