Collective Intelligence in Action

Satnam Alag

SAMPLE CHAPTER

MANNING

Collective Intelligence in Action

by Satnam Alag

Chapter 2

brief contents

2 Learning from user interactions

This chapter covers

- Architecture for applying intelligence
- Basic technical concepts behind collective intelligence
- The many forms of user interaction
- A working example of how user interaction is converted into collective intelligence

Through their interactions with your web application, users provide a rich set of information that can be converted into intelligence. For example, a user rating an item provides crisp quantifiable information about the user's preferences. Aggregating the rating across all your users or a subset of relevant users is one of the simplest ways to apply collective intelligence in your application.

There are two main sources of information that can be harvested for intelligence. First is *content-based*—based on information about the item itself, usually keywords or phrases occurring in the item. Second is *collaborative-based*—based on the interactions of users. For example, if someone is looking for a hotel, the collaborative filtering engine will look for similar users based on matching profile attributes and find

hotels that these users have rated highly. Throughout the chapter, the theme of using content and collaborative approaches for harvesting intelligence will be reinforced.

First and foremost, we need to make sure that you have the right architecture in place for embedding intelligence in your application. Therefore, we begin by describing the ideal architecture for applying intelligence. This will be followed by an introduction to some of the fundamental concepts needed to understand the underlying technology. You'll be introduced to the fields of content and collaborative filtering and how intelligence is represented and extracted from text. Next, we review the many forms of user interaction and how that interaction translates into collective intelligence for your application. The main aim of this chapter is to introduce you to the fundamental concepts that we leverage to build the underlying technology in parts 2 and 3 of the book. A strong foundation leads to a stronger house, so make sure you understand the fundamental concepts introduced in this chapter before proceeding on to later chapters.

2.1 Architecture for applying intelligence

All web applications consist, at a minimum, of an application server or a web server—to serve HTTP or HTTPS requests sent from a user's browser—and a database that stores the persistent state of the application. Some applications also use a messaging server to allow asynchronous processing via an event-driven Service-Oriented Architecture (SOA). The best way to embed intelligence in your application is to build it as a set of *services*—software components that each have a well-defined interface.

In this section, we look at the two kinds of intelligence-related services and their advantages and disadvantages.

2.1.1 Synchronous and asynchronous services

For embedding intelligence in your application, you need to build two kinds of services: synchronous and asynchronous services.

Synchronous services service requests from a client in a synchronous manner: the client waits till the service returns the response back. These services need to be fast, since the longer they take to process the request, the longer the wait time for the client. Some examples of this kind of a service are the runtime of an item-recommendation engine(a service that provides a list of items related to an item of interest for a user), a service that provides a model of user's profile, and a service that provides results from a search query.

For scaling and high performance, synchronous services should be stateless—the service instance shouldn't maintain any state between service requests. All the information that the service needs to process a request should be retrieved from a persistent source, such as a database or a file, or passed to it as a part of the service request. These services also use caching to avoid round-trips to the external data store. These services can be in the same JVM as the client code or be distributed in their own set of machines. Due to their stateless nature, you can have multiple instances of the services running

servicing requests. Typically, a load balancer is used in front of the multiple instances. These services scale nearly linearly, neglecting the overhead of load-balancing among the instances.

Asynchronous services typically run in the background and take longer to process. Examples of this kind of a service include a data aggregator service(a service that crawls the web to identify, gather, and classify relevant information) as well as a service that learns the profile of a user through a predictive model or clustering, or a search engine indexing content. Asynchronous learning services need to be designed to be stateless: they receive a message, process it, and then work on the next message. There can be multiple instances of these services all listening to the same queue on the messaging server. The messaging server takes care of load balancing between the multiple instances and will queue up the messages under load.

Figure 2.1 shows an example of the two kinds of services. First, we have the runtime API that services client requests synchronously, using typically precomputed information about the user and other derived information such as search indexes or predictive models. The intelligence-learning service is an asynchronous service that analyzes information from various types of content along with user-interaction information to create models that are used by the runtime API. Content could be either contained within your system or retrieved from external sources, such as by searching the blogosphere or by web crawling.

Table 2.1 lists some of the services that you'll be able to build in your application using concepts that we develop in this book.

As new information comes in about your users, their interactions, and the content in your system, the models used by the intelligence services need to be updated. There are two approaches to updating the models: event-driven and non-event-driven. We discuss these in the next two sections.

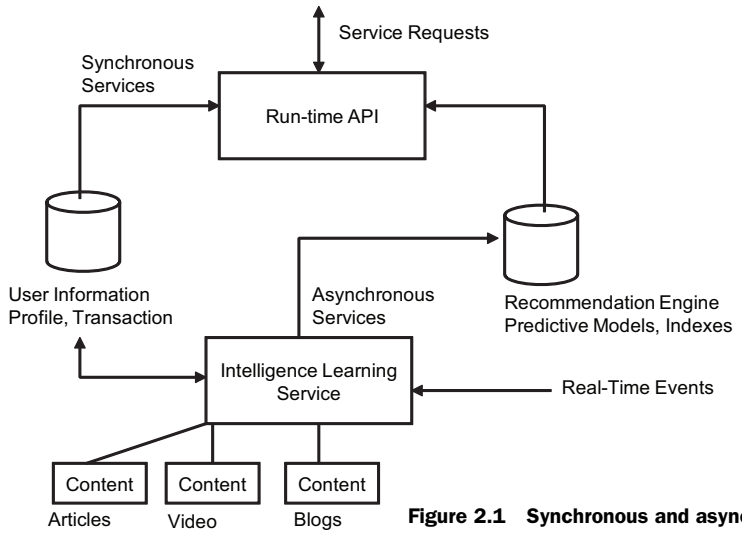

Figure 2.1 Synchronous and asynchronous learning services

Table 2.1 Summary of services that a typical application-embedding intelligence contains

Service	Processing type	Description
Intelligence Learning Service	Asynchronous	This service uses user-interaction information to build a profile of the user, update product relevance tables, transaction history, and so on.
Data Aggregator/ Classifier Service	Asynchronous	This service crawls external sites to gather information and derives intelligence from the text to classify it appropriately.
Search Service	Asynchronous Indexing Synchronous Results	Content—both user-generated and professionally developed—is indexed for search. This may be combined with user profile and transaction history to create personalized search results.
User Profile	Synchronous	Runtime model of user's profile that will be used for personalization.
Item Relevance Lookup Service	Synchronous	Runtime model for looking up related items for a given item.

2.1.2 *Real-time learning in an event-driven system*

As users interact on your site, perhaps by looking at an article or video, by rating a question, or by writing a blog entry, they're providing your application with information that can be converted into intelligence about them. As shown in figure 2.2, you can develop near–real-time intelligence in your application by using an event-driven Service-Oriented Architecture (SOA).

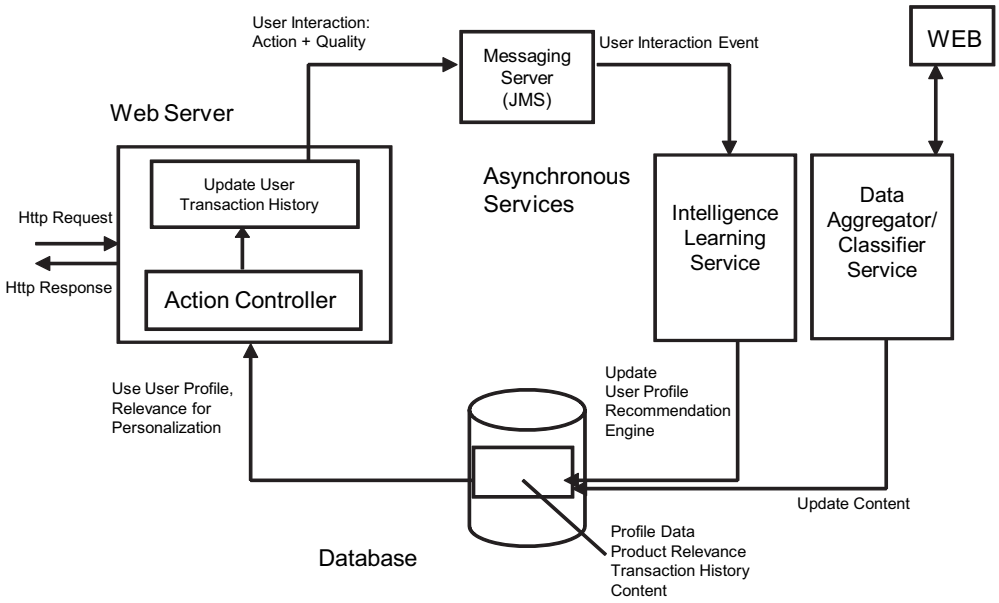

Figure 2.2 Architecture for embedding and deriving intelligence in an event-driven system

The web server receives a HTTP request from the user. Available locally in the same JVM is a service for updating the user transaction history. Depending on your architecture and your needs, the service may simply add the transaction history item to its memory and periodically flush the items out to either the database or to a messaging server.

Real-time processing can occur when a message is sent to the messaging server, which then passes this message out to any interested intelligence-learning services. These services will process and persist the information to update the user's profile, update the recommendation engine, and update any predictive models.[1] If this learning process is sufficiently fast, there's a good chance that the updated user's profile will be reflected in the personalized information shown to the user the next time she interacts.

NOTE As an alternative to sending the complete user transaction data as a message, you can also first store the message and then send a lightweight object that's a pointer to the information in the database. The learning service will retrieve the information from the database when it receives the message. If there's a significant amount of processing and data transformation that's required before persistence, then it may be advantageous to do the processing in the asynchronous learning service.

2.1.3 Polling services for non–event-driven systems

If your application architecture doesn't use a messaging infrastructure—for example, if it consists solely of a web server and a database—you can write user transaction history to the database. In this case, the learning services use a poll-based mechanism to periodically process the data, as shown in figure 2.3.

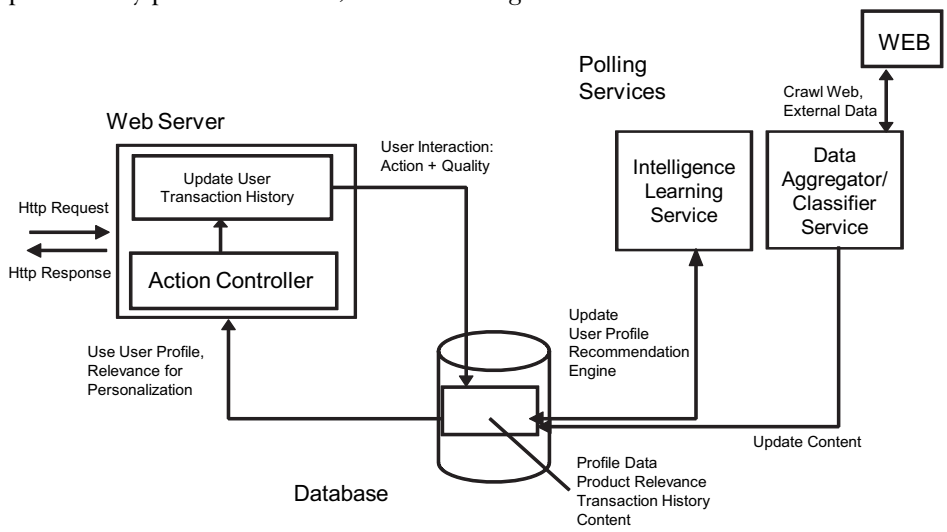

Figure 2.3 Architecture for embedding intelligence in a non-event-driven system

[1] The open source Drools complex-event-processing (CEP) framework could be useful for implementing a rule-based event-handling intelligent-learning service; see http://blog.athico.com/2007/11/pigeons-complex-event-processing-and.html.

So far we've looked at the two approaches for building intelligence learning services—event-driven and non–event-driven. Let's now look at the advantages and disadvantages of each of these approaches.

2.1.4 Advantages and disadvantages of event-based and non–event-based architectures

An event-driven SOA architecture is recommended for learning and embedding intelligence in your application because it provides the following advantages:

- *It provides more fine-grained real-time processing—every user transaction can be processed separately.* Conversely, the lag for processing data in a polling framework is dependent on the polling frequency. For some tasks such as updating a search index with changes, where the process of opening and closing a connection to the index is expensive, batching multiple updates in one event may be more efficient.
- *An event-driven architecture is a more scalable solution.* You can scale each of the services independently. Under peak conditions, the messaging server can queue up messages. Thus the maximum load generated on the system by these services will be bounded. A polling mechanism requires more continuous overhead and thus wastes resources.
- *An event-driven architecture is less complex to implement because there are standard messaging servers that are easy to integrate into your application.* Conversely, multiple instances of a polling service need to coordinate which rows of information are being processed among themselves. In this case, be careful to avoid using `select for update` to achieve this locking, because this often causes deadlocks. The polling infrastructure is often a source of bugs.

On the flip side, if you don't currently use a messaging infrastructure in your system, introducing a messaging infrastructure in your architecture can be a nontrivial task. In this case, it may be better to begin with building the learning infrastructure using a poll-based non–event-driven architecture and then upgrading to an event-driven architecture if the learning infrastructure doesn't meet your business requirements.

Now that we have an understanding of the architecture to apply intelligence in your application, let's next look at some of the fundamental concepts that we need to understand in order to apply CI.

2.2 Basics of algorithms for applying CI

In order to correlate users with content and with each other, we need a common language to compute relevance between items, between users, and between users and items. Content-based relevance is anchored in the content itself, as is done by information retrieval systems. Collaborative-based relevance leverages the user interaction data to discern meaningful relationships. Also, since a lot of content is in the form of unstructured text, it's helpful to understand how metadata can be developed from unstructured text. In this section, we cover these three fundamental concepts of learning algorithms.

We begin by abstracting the various types of content, so that the concepts and algorithms can be applied to all of them.

2.2.1 Users and items

As shown in figure 2.4, most applications generally consist of *users* and *items*. An *item* is any entity of interest in your application. Items may be articles, both user-generated and professionally developed; videos; photos; blog entries; questions and answers posted on message boards; or products and services sold in your application. If your application is a social-networking application, or you're looking to connect one user with another, then a user is also a type of item.

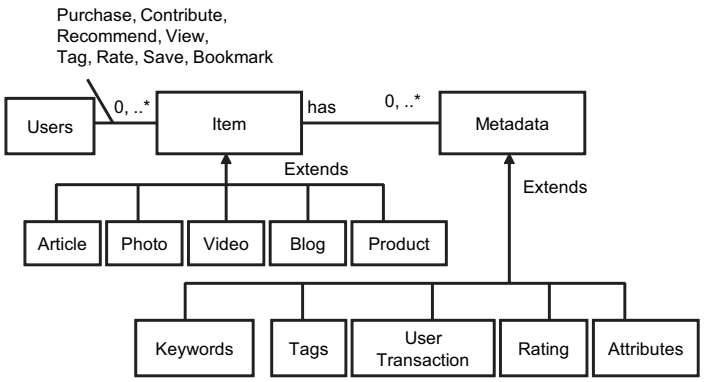

Figure 2.4 A user interacts with items, which have associated metadata.

Associated with each item is *metadata*, which may be in the form of professionally developed keywords, user-generated tags, keywords extracted by an algorithm after analyzing the text, ratings, popularity ranking, or just about anything that provides a higher level of information about the item and can be used to correlate items together. Think about metadata as a set of attributes that help qualify an item.

When an item is a user, in most applications there's no content associated with a user (unless your application has a text-based descriptive profile of the user). In this case, metadata for a user will consist of profile-based data and user-action based data. Figure 2.5 shows the three main sources of developing metadata for an item (remember a user is also an item). We look at these three sources next.

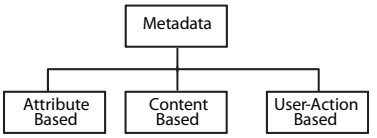

Figure 2.5 The three sources for generating metadata about an item

ATTRIBUTE-BASED

Metadata can be generated by looking at the attributes of the user or the item. The user attribute information is typically dependent on the nature of the domain of the application. It may contain information such as age, sex, geographical location, profession, annual income, or education level. Similarly, most nonuser items have attributes associated with them. For example, a product may have a price, the name of the

author or manufacturer, the geographical location where it's available, the creation or manufacturing date, and so on.

CONTENT-BASED

Metadata can be generated by analyzing the content of a document. As we see in the following sections, there's been a lot of work done in the area of information retrieval and text mining to extract metadata associated with unstructured text. The title, subtitles, keywords, frequency counts of words in a document and across all documents of interest, and other data provide useful information that can then be converted into metadata for that item.

USER-ACTION-BASED

Metadata can be generated by analyzing the interactions of users with items. User interactions provide valuable insight into preferences and interests. Some of the interactions are fairly explicit in terms of their intentions, such as purchasing an item, contributing content, rating an item, or voting. Other interactions are a lot more difficult to discern, such as a user clicking on an article and the system determining whether the user liked that item or not. This interaction can be used to build metadata about the user and the item. This metadata provides important information as to what kind of items the user would be interested in; which set of users would be interested in a new item, and so on.

Think about users and items having an associated vector of metadata attributes. The similarity or relevance between two users or two items or a user and item can be measured by looking at the similarity between the two vectors. Since we're interested in learning about the likes and dislikes of a user, let's next look at representing information related to a user.

2.2.2 Representing user information

A user's profile consists of a number of *attributes*—independent variables that can be used to describe the item of interest. As shown in figure 2.6, attributes can be *numerical*—have a continuous set of values, for example, the age of a user—or *nominal*—have a nonnumerical value or a set of string values associated with them. Further, nominal attributes can be either *ordinal*—enumerated values that have ordering in them, such as low, medium, and high—or *categorical*—enumerated values with no ordering, such as the color of one's eyes.

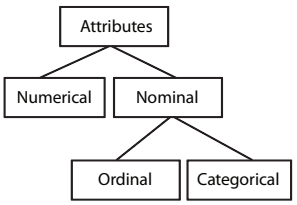

Figure 2.6 Attribute hierarchy of a user profile

All attributes are not equal in their predicting capabilities. Depending on the kind of learning algorithms used, the attributes can be *normalized*—converted to a scale of [0-1]. Different algorithms use either numerical or nominal attributes as inputs. Further, numerical and nominal attributes can be converted from one format to another depending on the kind of algorithms used. For example, the age of a user can be converted to a nominal attribute by creating *buckets*, say: "Teenager" for users under the

Table 2.2 Examples of user-profile attributes

Attribute	Type	Example	Comments
Age	Numeric	26 years old	User typically provides birth date.
Sex	Categorical	Male, Female	
Annual Income	Ordinal or Numeric	Between 50-100K or 126K	
Geographical Location	Categorical can be converted to numerical	Address, city, state, zip	The geo-codes associated with the location can be used as a distance measure to a reference point.

age of 18, "Young Person" for those between 18 and 25, and so on. Table 2.2 has a list of user attributes that may be available in your application.

In addition to user attributes, the user's interactions with your application give you important data that can be used to learn about your user, find similar users (clustering), or make a prediction. The number of times a user has logged in to your application within a period of time, his average session time, and the number of items purchased are all examples of derived attributes that can be used for clustering and building predictive models.

Through their interactions, users provide a rich set of information that can be harvested for intelligence. Table 2.3 summarizes some of the ways users provide valuable information that can be used to add intelligence to your application.

Table 2.3 The many ways users provide valuable information through their interactions

Technique	Description
Transaction history	The list of items that a user has bought in the past Items that are currently in the user's shopping cart or favorites list
Content visited	The type of content searched and read by the user The advertisements clicked
Path followed	How the user got to a particular piece of content—whether directly from an external search engine result or after searching in the application The intent of the user—proceeding to the e-commerce pages after researching a topic on the site
Profile selections	The choices that users make in selecting the defaults for their profiles and profile entries; for example, the default airport used by the user for a travel application
Feedback to polls and questions	If the user has responded to any online polls and questions
Rating	Rating of content
Tagging	Associating tags with items
Voting, bookmarking, saving	Expressing interest in an item

We've looked at how various kinds of attributes can be used to represent a user's profile and the use of user-interaction data to learn about the user. Next, let's look at how intelligence can be generated by analyzing content and by analyzing the interactions of the users. This is just a quick look at this fairly large topic and we build on it throughout the book.

2.2.3 Content-based analysis and collaborative filtering

User-centric applications aim to make the application more valuable for users by applying CI to personalize the site. There are two basic approaches to personalization: content-based and collaborative-based.

Content-based approaches analyze the content to build a representation for the content. Terms or phrases (multiple terms in a row) appearing in the document are typically used to build this representation. Terms are converted into their basic form by a process known as *stemming*. Terms with their associated weights, commonly known as *term vectors*, then represent the metadata associated with the text. Similarity between two content items is measured by measuring the similarity associated with their term vectors.

A user's profile can also be developed by analyzing the set of content the user interacted with. In this case, the user's profile will have the same set of terms as the items, enabling you to compute the similarities between a user and an item. Content-based recommendation systems do a good job of finding related items, but they can't predict the quality of the item—how popular the item is or how a user will like the item. This is where collaborative-based methods come in.

A collaborative-based approach aims to use the information provided by the interactions of users to predict items of interest for a user. For example, in a system where users rate items, a collaborative-based approach will find patterns in the way items have been rated by the user and other users to find additional items of interest for a user. This approach aims to match a user's metadata to that of other similar users and recommend items liked by them. Items that are liked by or popular with a certain segment of your user population will appear often in their interaction history—viewed often, purchased often, and so forth. The frequency of occurrence or ratings provided by users are indicative of the quality of the item to the appropriate segment of your user population. Sites that use collaborative filtering include Amazon, Google, and Netflix. Collaborative-based methods are language independent, and you don't have to worry about language issues when applying the algorithm to content in a different language.

There are two main approaches in collaborative filtering: memory-based and model-based. In memory-based systems, a similarity measure is used to find similar users and then make a prediction using a weighted average of the ratings of the similar users. This approach can have scalability issues and is sensitive to data sparseness. A model-based approach aims to build a model for prediction using a variety of approaches: linear algebra, probabilistic methods, neural networks, clustering, latent classes, and so on. They normally have fast runtime predicting capabilities. Chapter 12

covers building recommendation systems in detail; in this chapter we introduce the concepts via examples.

Since a lot of information that we deal with is in the form of unstructured text, it's helpful to review some basic concepts about how intelligence is extracted from unstructured text.

2.2.4 Representing intelligence from unstructured text

This section deals with developing a representation for unstructured text by using the content of the text. Fortunately, we can leverage a lot of work that's been done in the area of information retrieval. This section introduces you to terms and term vectors, used to represent metadata associated with text. Section 4.3 presents a detailed working example on this topic, while chapter 8 develops a toolkit that you can use in your application for representing unstructured text. Chapter 3 presents a collaborative-based approach for representing a document using user-tagging.

Now let's consider an example where the text being analyzed is the phrase "Collective Intelligence in Action."

In its most basic form, a text document consists of *terms*—words that appear in the text. In our example, there are four terms: *Collective*, *Intelligence*, *in*, and *Action*. When terms are joined together, they form *phrases*. *Collective Intelligence* and *Collective Intelligence in Action* are two useful phrases in our document.

The *Vector Space Model* representation is one of the most commonly used methods for representing a document. As shown in figure 2.7, a document is represented by a term vector, which consists of terms appearing in the document and a relative weight for each of the terms. The term vector is one representation of metadata associated with an item. The weight associated with each term is a product of two computations: *term frequency* and *inverse document frequency*.

Term frequency (TF) is a count of how often a term appears. Words that appear often may be more relevant to the topic of interest. Given a particular domain, some words appear more often than others. For example, in a set of books about Java, the word *Java* will appear often. We have to be more discriminating to find items that have these less-common terms: *Spring*, *Hibernate*, and *Intelligence*. This is the motivation behind *inverse document frequency (IDF)*. IDF aims to boost terms that are less frequent. Let the total number of documents of interest be n, and let n_i be the number of times a given term appears across the documents. Then IDF for a term is computed as follows:

$$idf_i = \log\left(\frac{n}{n_i}\right)$$

Note that if a term appears in all documents, then its IDF is $\log(1)$ which is 0.

Commonly occurring terms such as *a*, *the*, and *in* don't add much value in representing the document. These are commonly known as *stop words* and are removed from the term vector. Terms are also

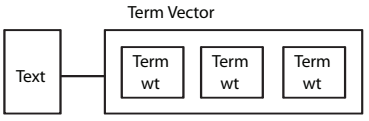

Figure 2.7 Term vector representation of text

converted to lowercase. Further, words are stemmed—brought to their root form—to handle plurals. For example, *toy* and *toys* will be stemmed to *toi*. The position of words, for example whether they appear in the title, keywords, abstract, or the body, can also influence the relative weights of the terms used to represent the document. Further, synonyms may be used to inject terms into the representation.

Figure 2.8 shows the steps involved in analyzing text. These steps are

1. *Tokenization*—Parse the text to generate terms. Sophisticated analyzers can also extract phrases from the text.
2. *Normalize*—Convert them into a normalized form such as converting text into lower case.
3. *Eliminate stop words*—Eliminate terms that appear very often.
4. *Stemming*—Convert the terms into their stemmed form to handle plurals.

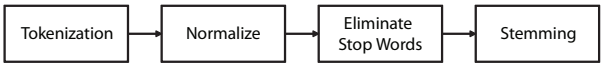

Figure 2.8 Typical steps involved in analyzing text

A large document will have more occurrences of a term than a similar document of shorter length. Therefore, within the term vector, the weights of the terms are normalized, such that the sum of the squared weights for all the terms in the term vector is equal to one. This normalization allows us to compare documents for similarities using their term vectors, which is discussed next.

The previous approach for generating metadata is content based. You can also generate metadata by analyzing user interaction with the content—we look at this in more detail in sections 2.3 and 2.4; chapter 3 deals with developing metadata from user tagging.

So far we've looked at what a term vector is and have some basic knowledge of how they're computed. Let's next look at how to compute similarities between them. An item that's very similar to another item will have a high value for the computed similarity metric. An item whose term vector has a high computed similarity to that of a user's will be very *relevant* to a user—chances are that if we can build a term vector to capture the likes of a user, then the user will like items that have a similar term vector.

2.2.5 Computing similarities

A term vector is a vector where the direction is the magnitude of the weights for each of the terms. The term vector has multiple dimensions—thousands to possibly millions, depending on your application. Multidimensional vectors are difficult to visualize, but the principles used can be illustrated by using a two-dimensional vector, as shown in figure 2.9.

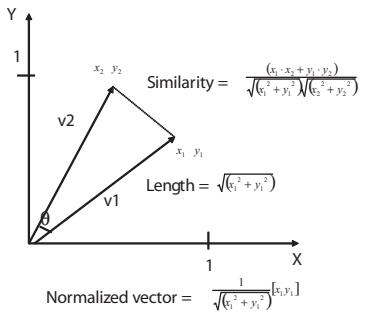

Figure 2.9 Two dimensional vectors, v1 and v2

Given a vector representation, we normalize the vector such that its length is of size 1 and compare vectors by computing the similarity between them. Chapter 8 develops the Java classes for doing this computation. For now, just think of vectors as a means to represent information with a well-developed math to compute similarities between them.

So far we've looked at the use of term vectors to represent metadata associated with content. We've also looked at how to compute similarities between term vectors. Now let's take this one step forward and introduce the concept of a dataset. Algorithms use data as input for analysis. This data consists of multiple instances represented in a tabular form. Based on how data is populated in the table, we can classify the dataset into two forms: densely populated, or high-dimensional sparsely populated datasets—similar in characteristics to a term vector.

2.2.6 Types of datasets

To illustrate the two forms of datasets used as input for learning by algorithms, let's consider the following example.

Let there be three users—John, Joe, and Jane. Each has three attributes: age, sex, and average number of minutes spent on the site. Table 2.4 shows the values for the various attributes for these users. This data can be used for clustering[2] and/or to build a predictive model.[3] For example, similar users according to age and/or sex might be a good predictor of the number of minutes a user will spend on the site.

In this example dataset, the age attribute is a good predictor for number of minutes spent—the number of minutes spent is inversely proportional to the age. The sex attribute has no effect in the prediction. In this made-up example, a simple linear model is adequate to predict the number of minutes spent (minutes spent = 50 – age of user).

	Age	Sex	Number of minutes per day spent on the site
John	25	M	25
Joe	30	M	20
Jane	20	F	30

Table 2.4 Dataset with small number of attributes

This is a densely populated dataset. Note that the number of rows in the dataset will increase as we add more users. It has the following properties:

- *It has more rows than columns*—The number of rows is typically a few orders of magnitude more than the number of columns. (Note that to keep things simple, the number of rows and columns is the same in our example.)
- *The dataset is richly populated*—There is a value for each cell.

[2] Chapter 9 covers clustering algorithms.

[3] Chapter 10 deals with building predictive models.

The other kind of dataset (high-dimensional, sparsely populated) is a generalization of the term vector representation. To understand this dataset, consider a window of time such as the past week. We consider the set of users who've viewed any of the videos on our site within this timeframe. Let n be the total number of videos in our application, represented as columns, while the users are represented as rows. Table 2.5 shows the dataset created by adding a 1 in the cell if a user has viewed a video. This representation is useful to find similar users and is known as the *User-Item matrix.*

	Video 1	Video 2	...	...	Video n
John	1				
Joe	1	1			
Jane					1

Table 2.5 Dataset with large number of attributes

Alternatively, when the users are represented as columns and the videos as rows, we can determine videos that are similar based on the user interaction: "Users who have viewed this video have also viewed these other videos." Such an analysis would be helpful in finding related videos on a site such as YouTube. Figure 2.10 shows a screenshot of such a feature at YouTube. It shows related videos for a video.

Figure 2.10 Screenshot from YouTube showing related videos for a video

This dataset has the following properties:

- *The number of columns is large*—For example, the number of products in a site like Amazon.com is in millions, as is the number of videos at YouTube.
- *The dataset is sparsely populated with nonzero entries in a few columns.*
- *You can visualize this dataset as a multidimensional vector*—Columns correspond to the dimensions and the cell entry corresponds to the weight associated for that dimension.

We develop a toolkit to analyze this kind of dataset in chapter 8. The dot product or cosine between two vectors is used as a similarity metric to compare two vectors.

Note the similarity of this dataset with the term vector we introduced in section 2.2.3. Let there be *m* terms that occur in all our documents. Then the term vectors corresponding to all our documents have the same characteristics as the previous dataset, as shown in table 2.6.

	Term 1	Term 2	..	..	Term m
Document 1	0.8				0.6
Document 2		0.7	0.7		
Document 3					1

Table 2.6 Sparsely populated dataset corresponding to term vectors

Now that we have a basic understanding of how metadata is generated and represented, let's look at the many forms of user interaction in your application and how they are converted to collective intelligence.

2.3 Forms of user interaction

To extract intelligence from a user's interaction in your application, it isn't enough to know what content the user looked at or visited. You also need to quantify the quality of the interaction. A user may like the article or may dislike it, these being two extremes. What one needs is a quantification of how the user liked the item relative to other items.

Remember, we're trying to ascertain what kind of information is of interest to the user. The user may provide this directly by rating or voting for an article, or it may need to be derived, for example, by looking at the content that the user has consumed. We can also learn about the item that the user is interacting with in the process.

In this section, we look at how users provide quantifiable information through their interactions; in section 2.4 we look at how these interactions fit in with collective intelligence. Some of the interactions such as ratings and voting are explicit in the user's intent, while other interactions such as using clicks are noisy—the intent of the user isn't known perfectly and is implicit. If you're thinking of making your application more interactive or intelligent, you may want to consider adding some of the functionality mentioned in this section. We also look at the underlying persistence architecture that's required to support the functionality. Let's begin with ratings and voting.

2.3.1 Rating and voting

Asking the user to rate an item of interest is an explicit way of getting feedback on how well the user liked the item. The advantage with a user rating content is that the information provided is quantifiable and can be used directly.

It's interesting to note that most ratings in a system tend to be positive, especially since people rate items that they've bought/interacted with and they typically buy/interact with items that they like.

Next, let's look at how you can build this functionality in your application.

PERSISTENCE MODEL[4]

Figure 2.11 shows the persistence model for storing ratings. Let's introduce two entities: `user` and `item`. `user_item_rating` is a mapping table that has a composite key, consisting of the user ID and content ID. A brief look at the cardinality between the entities show that

- Each user may rate 0 or more items.
- Each rating is associated with only one user.
- An item may contain 0 or more ratings.
- Each rating is associated with only one item.

Based on your application, you may alternatively want to also classify the items in your application. It's also helpful to have a generic table to store the ratings associated with the items. Computing a user's average rating for an item or item type is then a simple database query.

In this design, answers to the following questions amount to a simple database query:

- What is the average rating for a given item?
- What is the average rating for a given item from users who are between the ages of 25 and 35?
- What are the top 10 rated items?

The last query can be slow, but faster performance can be obtained by having a `user_item_rating_statistic` table, as shown in figure 2.10. This table gets updated by a trigger every time a new row is inserted in the `user_item_rating` table. The average

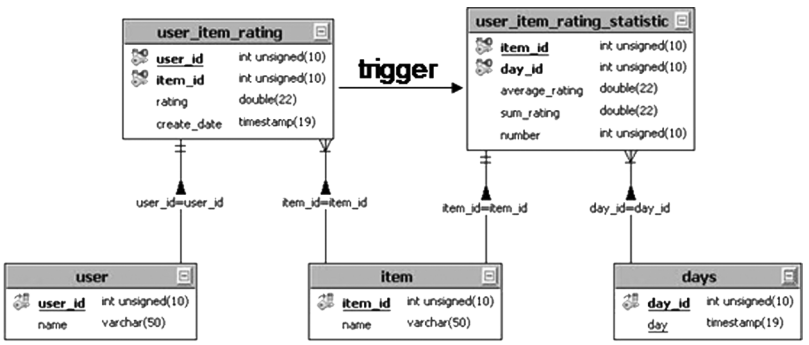

Figure 2.11 Persistence of ratings in a table that stores each user's ratings in a separate table

[4] The code to create the tables, populate the database with test data, and run the queries is available from the code download site for this book.

is precomputed and is calculated by dividing the cumulative sum by the number of ratings. If you want to trend the ratings of an item on a daily basis, you can augment the `user_item_rating_statistic` to have the day as another key.

VOTING—"DIGG IT"

Most applications that allow users to rate use a scale from zero to five. Allowing a user to vote is another way to involve and obtain useful information from the user. Digg, a website that allows users to contribute and vote on interesting articles, uses this idea. As shown in figure 2.12, a user can either *digg* an article, casting a positive vote, or *bury* it, casting a negative vote. There are a number of heuristics applied to selecting which articles make it to the top, some being the number of positive votes received by the article along with the date the article was submitted in Digg.

Figure 2.12 At Digg.com, users are allowed to vote on how they like an article—"digg it" is a positive vote, while "Bury" is a negative vote.

Voting is similar to rating. However, a vote can have only two values—1 for a positive vote and -1 for a negative vote.

2.3.2 Emailing or forwarding a link

As a part of viral marketing efforts, it's common for websites to allow users to email or forward the contents of a page to others. Similar to voting, forwarding the content to others can be considered a positive vote for the item by the user. Figure 2.13 is a screenshot from *The Wall Street Journal* showing how a user can forward an article to another user.

Figure 2.13 Screenshot from *The Wall Street Journal* (wsj.com) that shows how a user can forward/email an article to another user

2.3.3 Bookmarking and saving

Online bookmarking services such as del.icio.us and spurl.net allow users to store and retrieve URLs, also known as bookmarks. Users can discover other interesting links that other users have bookmarked through

recommendations, hot lists, and other such features. By bookmarking URLs, a user is explicitly expressing interest in the material associated with the bookmark. URLs that are commonly bookmarked bubble up higher in the site.

The process of saving an item or adding it to a list is similar to bookmarking and provides similar information. Figure 2.14 is an example from *The New York Times*, where a user can save an item of interest. As shown, this can then be used to build a recommendation engine where a user is shown related items that other users who saved that item have also saved.

Figure 2.14 Saving an item to a list (NY Times.com)

If a user has a large number of bookmarks, it can become cumbersome for the user to find and manage bookmarked or saved items. For this reason, applications allow their users to create *folders*—a collection of items bookmarked or saved together. As shown in figure 2.15, folders follow the composite design pattern,[5] where they're composed of bookmarked items. A folder is just another kind of item in your application that can be shared, bookmarked, and rated in your application. Based on their composition, folders have metadata associated with them.

Next, let's look at how a user purchasing an item also provides useful information.

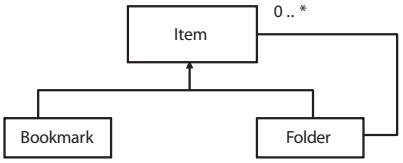

Figure 2.15 Composite pattern for organizing bookmarks together

2.3.4 Purchasing items

In an e-commerce site, when users purchase items, they're casting an explicit vote of confidence in the item—unless the item is returned after purchase, in which case it's a negative vote. Recommendation engines, for example the one used by Amazon (Item-to-Item recommendation engine; see section 12.4.1) can be built from analyzing the procurement history of users. Users that buy similar items can be correlated and items that have been bought by other users can be recommended to a user.

2.3.5 Click-stream

So far we've looked at fairly explicit ways of determining whether a user liked or disliked a particular item, through ratings, voting, forwarding, and purchasing items.

[5] Refer to the Composite Pattern in the Gang of Four design patterns.

When a list of items is presented to a user, there's a good chance that the user will click on one of them based on the title and description. But after quickly scanning the item, the user may find the item to be not relevant and may browse back or search for other items.

A simple way to quantify an article's relevance is to record a positive vote for any item clicked. This approach is used by Google News to personalize the site (see section 12.4.2). To further filter out noise, such as items the user didn't really like, you could look at the amount of time the user spent on the article. Of course, this isn't fail proof. For example, the user could have left the room to get some coffee or been interrupted while looking at the article. But on average, simply looking at whether an item was visited and the time spent on it provides useful information that can be mined later. You can also gather useful statistics from this data:

- What is the average time a user spends on a particular item?
- For a user, what is the average time spent on any given article?

One of the ways to validate the data and clear out outliers is to use a *validation window*. To build a validation window, treat the amount of time spent by a user as a normal distribution (see figure 2.16) and compute the mean and standard deviation from the samples.

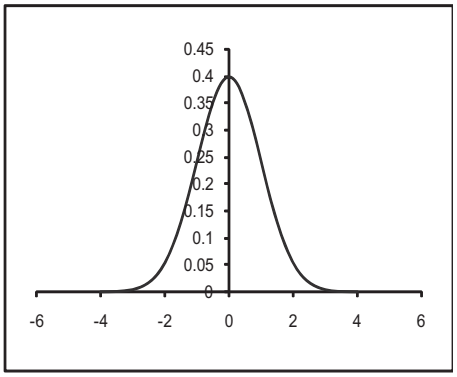

Figure 2.16 A normal distribution with a mean of 0 and standard deviation of 1

Let's demonstrate this with a simple example—it's fictitious, but illustrates the point well. Let the amount of time spent by nine readers on an article be [5, 47, 50, 55, 47, 54, 100, 45, 50] seconds. Computing the mean is simple (add them all up and divide it by nine, the number of samples); it's 50.33 seconds. Next, let's compute the standard deviation. For this, take the difference of each of the samples from its mean and square it. This leads to [2055.11, 11.11, 0.11, 21.78, 11.11, 13.44, 2466.78, 28.44, 0.11]. Add them up and divide it by eight, the number of samples minus one. This gives us 576, and the square root of this is the standard deviation, which comes out to be 24. Now you can create a validation window two or three times the standard deviation from the mean. For our example, we take two times the standard deviation, which gives us a confidence level of 95 percent. For our example, this is [2.33 98]. Anything outside this range is an outlier.

So we flag the seventh sample of 100 seconds as an outlier—perhaps the user had stepped out or was interrupted while reading the article. Next, continue the same process with the remaining eight samples [5, 47, 50, 55, 47, 54, 45, 50]. The new mean and standard deviation is 44.125 and 16.18. The new confidence window is [11.76 76.49]. The first sample is an outlier; perhaps the user didn't find the article relevant.

Now let's remove this outlier and recompute the validation window for the sample set of [47, 50, 55, 47, 54, 45, 50]. The new mean and standard deviation is 49.71 and 3.73

respectively. The new confidence window is [42.26 57.17]. Most users will spend time within this window. Users that spent less time were probably not interested in the content of the article.

Of course, if you wanted to get more sophisticated (and a lot more complex), you could try to model the average time that a user spends on an item and correlate it with average time spent by a typical user to shrink or expand the validation window. But for most applications, the preceding process of validation window should work well. Or if you want to keep things even simpler, simply consider whether the article has been visited, irrespective of the time spent[6] reading it.

2.3.6 Reviews

Web 2.0 is all about connecting people with similar people. This similarity may be based on similar tastes, positions, opinions, or geographic location. Tastes and opinions are often expressed through reviews and recommendations. These have the greatest impact on other users when

- They're unbiased
- The reviews are from similar users
- They're from a person of influence

Depending on the application, the information provided by a user may be available to the entire population of users, or may be privately available only to a select group of users. This is especially the case for software-as-a-service (SaaS) applications, where a company or enterprise subscribing to the service forms a natural grouping of users. In such applications, information isn't usually shared across domains. The information is more contextually relevant to users within the company, anyway.

Perhaps the biggest reasons why people review items and share their experiences are to be discovered by others and for boasting rights. Reviewers enjoy the recognition, and typically like the site and want to contribute to it. Most of them enjoy doing it. A number of applications highlight the contributions made by users, by having a Top Reviewers list. Reviews from top reviewers are also typically placed toward the top and featured more prominently. Sites may also feature one of their top reviewers on the site as an incentive to contribute.

Some sites may also provide an incentive, perhaps monetary, for users to contribute content and reviews. Epinions.com pays a small amount to its reviewers. Similarly, Revver, a video sharing site, pays its users for contributed videos. It's interesting to note that even though sites like Epinions.com pay money to their reviewers, while Amazon doesn't, Amazon still has on order of magnitude more reviews from its users.

Users tend to contribute more to sites that have the biggest audience.

In a site where anyone can contribute content, is there anything that stops your competitors from giving you an unjustified low rating? Good reviewers, especially those that are featured toward the top, try to build a good reputation. Typically, an

[6] Google News, which we look at in chapter 12, simply uses a click as a positive vote for the item.

application has links to the reviewer's profile along with other reviews that he's written. Other users can also write comments about a review. Further, just like voting for articles at Digg, other users can endorse a reviewer or vote on his reviews. As shown in figure 2.17, taken from epinions.com, users can "Trust" or "Block" reviewers to vote on whether a reviewer can be trusted.

Figure 2.17 Epinions.com allows users to place a positive or negative vote of confidence in a reviewer.

The feedback from other users about how helpful the review was helps to weed out biased and unhelpful reviews. Sites also allow users to report reviewers who don't follow their guidelines, in essence allowing the community to police itself.

MODELING THE REVIEWER AND ITEM RELATIONSHIP

We need to introduce another entity—the reviewer, who may or may not be a user of your application. The association between a reviewer, an item, and an ItemReview is shown in figure 2.18. This is similar to the relationship between a user and ratings.

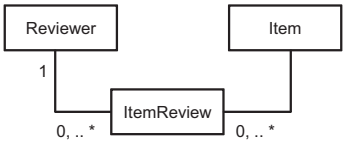

Figure 2.18 The association between a reviewer, an item, and the review of an item

- Each reviewer may write zero or more reviews.
- Each review is written by a reviewer.
- Each item may have zero or more reviews.
- Each review is associated with one item.

The persistence design for storing reviews is shown in figure 2.19, and is similar to the one we developed for ratings. Item reviews are in the form of unstructured text and thus need to be indexed by search engines.

So far, we've looked at the many forms of user interaction and the persistence architecture to build it in your application. Next, let's look at how this user-interaction information gets converted into collective intelligence.

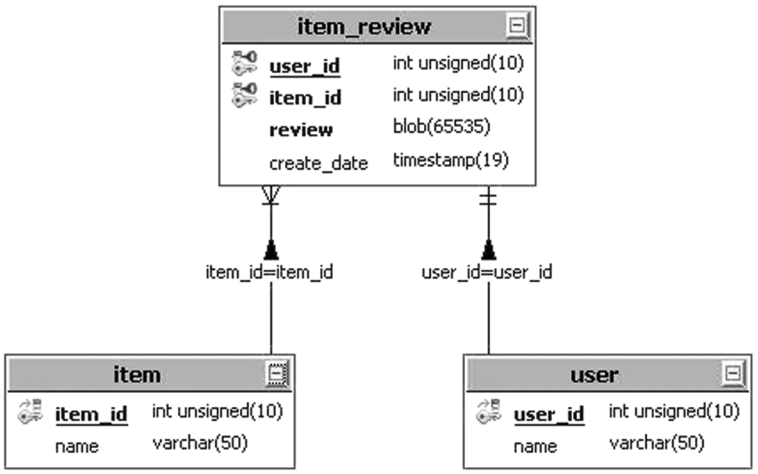

Figure 2.19 Schema design for persisting reviews

2.4 Converting user interaction into collective intelligence

In section 2.2.6, we looked at the two forms of data representation that are used by learning algorithms. User interaction manifests itself in the form of the sparsely populated dataset. In this section, we look at how user interaction gets converted into a dataset for learning.

To illustrate the concepts, we use a simple example dealing with three users who've rated photographs. In addition to the cosine-based similarity computation we introduced in section 2.2.5, we introduce two new similarity computations: correlation-based similarity computation and adjusted-cosine similarity computation. In this section, we spend more time on this example which deals with ratings to illustrate the concepts. We then briefly cover how these concepts can be generalized to analyze other user interactions in section 2.4.2. That section forms the basis for building a recommendation engine, which we cover in chapter 12.

2.4.1 Intelligence from ratings via an example

There are a number of ways to transform raw ratings from users into intelligence. First, you can simply aggregate all the ratings about the item and provide the average as the item's rating. This can be used to create a Top 10 Rated Items list. Averages work well, but then you're constantly promoting the popular content. How do you reach the potential of The Long Tail? A user is really interested in the average rating for content by users who have similar tastes.

Clustering is a technique that can help find a group of users similar to the user. The average rating of an item by a group of users similar to a user is more relevant to the user than a general average rating. Ratings provide a good quantitative feedback of how good the content is.

Let's consider a simple example to understand the basic concepts associated with using ratings for learning about the users and items of interest. This section introduces you to some of the basic concepts.

Let there be three users: John, Jane, and Doe, who each rate three items. As per our discussion in section 2.2.1, items could be anything—blog entries, message board questions, video, photos, reviews, and so on. For our example, let them rate three photos: Photo1, Photo2, and Photo3, as shown in table 2.7. The table also shows the average rating for each photo and the average rating given by each user. We revisit this example in section 12.3.1 when we discuss recommendation engines.

	Photo1	Photo2	Photo3	Average
John	3	4	2	3
Jane	2	2	4	8/3
Doe	1	3	5	3
Average	2	3	11/3	26/3

Table 2.7 Ratings data used in the example

Given this set of data, we answer two questions in our example:

- What are the set of related items for a given item?
- For a user, who are the other users that are similar to the user?

We answer these questions using three approaches: cosine-based similarity, correlation-based similarity, and adjusted-cosine-based similarity.

COSINE-BASED SIMILARITY COMPUTATION

Cosine-based similarity takes the dot product of two vectors as described in section 2.2.4. First, to learn about the photos, we transpose the matrix, so that a row corresponds to a photo while the columns (users) correspond to dimensions that describe the photo, as shown in table 2.8.

	John	Jane	Doe	Average
Photo1	3	2	1	2
Photo 2	4	2	3	3
Photo 3	2	4	5	11/3
Average	3	8/3	3	26/3

Table 2.8 Dataset to describe photos

Next, we normalize the values for each of the rows. This is done by dividing each of the cell entries by the square root of the sum of the squares of entries in a particular row. For example, each of the terms in the first row is divided by $\sqrt{3^2+2^2+1^2} = \sqrt{14} = 3.74$ to get the normalized dataset shown in table 2.9[7].

	John	Jane	Doe
Photo1	0.8018	0.5345	0.2673
Photo2	0.7428	0.3714	0.557
Photo3	0.2981	0.5963	0.7454

Table 2.9 Normalized dataset for the photos using raw ratings

We can find the similarities between the items by taking the dot product of their vectors. For example, the similarity between Photo 1 and Photo 2 is computed as (0.8018 * 0.7428) + (0.5345 * 0.3714) + (0.2673 * 0.557) = 0.943.

Using this, we can develop the item-to-item similarity table, shown in table 2.10. This table also answers our first question: what are the set of related items for a given item? According to this, Photo1 and Photo2 are very similar. The closer to 1 a value in the similarity table is, the more similar the items are to each other.

	Photo1	Photo2	Photo3
Photo1	1	0.943	0.757
Photo2	0.943	1	0.858
Photo3	0.757	0.858	1

Table 2.10 Item-to-item using raw ratings

[7] There is a unit test in the downloadable code that implements this example.

To determine similar users, we need to consider the original data in table 2.7. Here, associated with each user is a vector, where the rating associated with each item corresponds to a dimension in the vector. The analysis process is similar to our approach for calculating the item-to-item similarity table. We first need to normalize the vectors and then take a dot product between two normalized vectors to compute their similarities.

Table 2.11 contains the normalized vectors associated with each user. The process is similar to the approach taken to compute table 2.9 from table 2.8. For example, $\sqrt{3^2+4^2+2^2} = \sqrt{29} = 5.385$ is the normalizing factor for John's vector in table 2.7.

	Photo1	Photo2	Photo3
John	0.5571	0.7428	0.3714
Jane	0.4082	0.4082	0.8165
Doe	0.1690	0.5071	0.8452

Table 2.11 Normalized rating vectors for each user

Next, a user-to-user similarity table can be computed as shown in table 2.12 by taking the dot product of the normalized vectors for two users.

	John	Jane	Doe
John	1	0.83	0.78
Jane	0.83	1	0.97
Doe	0.78	0.97	1.00

Table 2.12 User-to-user similarity table

As shown in table 2.12, Jane and Doe are very similar. The preceding approach uses the raw ratings assigned by a user to an item. Another alternative is to focus on the deviations in the rating from the average values that a user provides. We look at this next.

CORRELATION-BASED SIMILARITY COMPUTATION

Similar to the dot product or cosine of two vectors, one can compute the correlation between two items as a measure of their similarity—the *Pearson-r correlation*. This correlation between two items is a number between –1 and 1, and it tells us the direction and magnitude of association between two items or users. The higher the magnitude—closer to either –1 or 1—the higher the association between the two items. The direction of the correlation tells us how the variables vary. A negative number means one variable increases as the other decreases, or in this example, the rating of one item decreases as the rating of another increases.

To compute the correlation, we need to isolate those cases where the users corated items—in our case, it's the complete set, as all the users have rated all the content. Let U be the set of users that have rated both item i and j.

Now the scary-looking formula to compute the correlation:

$$corr(i,j) = \frac{\sum_{ueU}(R_{ui}-\overline{R}_i)(R_{uj}-\overline{R}_j)}{\sqrt{\sum_{ueU}(R_{ui}-\overline{R}_i)^2}\sqrt{\sum_{ueU}(R_{uj}-\overline{R}_j)^2}}$$

where $R_{u,i}$ is the rating of user u for item i and $\overline{R}_i$ is the average rating of item i. The correlation computation looks for variances from the mean value for the items.

Let's look at the correlation of Photo 1 and Photo 2.

$$\text{Numerator} = (3-2)(4-3)+(2-2)(2-3)+(1-2)(3-3) = 1$$

$$\text{Denominator} = \sqrt{(3-2)^2+(2-2)^2+(1-2)^2}\sqrt{(4-3)^2+(2-3)^2+(3-3)^2} = 2$$

$$Corr(1,2)=0.5$$

Alternatively, for the computation, it's useful to subtract the average value for a row as shown in table 2.13. Note that the sum of the numbers for each row is equal to zero.

	John	Jane	Doe
Photo1	1	0	-1
Photo 2	1	-1	0
Photo 3	-5/3	1/3	4/3

Table 2.13 Normalized matrix for the correlation computation

Table 2.14 shows the correlation matrix between the items and provides answers to our first question: what are the set of related items for a given item? According to this, Photo 1 and Photo 3 are strongly negatively correlated.

	Photo1	Photo2	Photo3
Photo1	1	0.5	-0.982
Photo2	0.5	1	-0.655
Photo3	-0.982	-0.655	1

Table 2.14 Correlation matrix for the items

Similarly, the correlation matrix between the users is computed along the rows of the data shown in table 2.7. Table 2.15 contains the normalized rating vectors for each user that will be used for computing the correlation. Note that the sum of the values for each row is 0.

	Photo1	Photo2	Photo3
John	0	0.7071	-0.7071
Jane	-0.4083	-0.4083	0.8166
Doe	-0.7071	0	0.7071

Table 2.15 Normalized rating vectors for each user

The resulting correlation matrix is shown in table 2.16 and provides answers to our second question: given a user, who are the other users that are similar to that user? Note that users Jane and Doe are highly correlated—if one likes an item, chances are the other likes it, too. John is negatively correlated—he dislikes what Jane and Doe like.

Since users Jane and Doe are highly correlated, you can recommend items that are rated highly by Jane to Doe and vice versa.

	John	Jane	Doe
John	1	-0.866	-0.5
Jane	-0.866	1	0.87
Doe	-0.5	0.87	1

Table 2.16 Correlation matrix for the users

ADJUSTED COSINE-BASED SIMILARITY COMPUTATION

One drawback of computing the correlation between items is that it doesn't take into account the difference in rating scale between users. For example, in the example data, user Doe is correlated highly with Jane but tends to give ratings toward the extremes.

An alternative formula, known as *adjusted cosine* is used, which is

$$similarity(i,j) = \frac{\sum_{ueU}(R_{u,i} - \overline{R}_u)(R_{u,j} - \overline{R}_u)}{\sqrt{\sum_{ueU}(R_{u,i} - \overline{R}_u)^2}\sqrt{\sum_{ueU}(R_{u,ij} - \overline{R}_u)^2}}$$

where R_u is the average rating for user u. Here, instead of subtracting the average value for a row, the average value provided by a user is used.

To compute this, it's again useful to normalize the dataset by removing the average rating value from the column values. This leads to the data shown in table 2.17. Note that the sum of the entries for a column is equal to zero.

	John	Jane	Doe
Photo1	0	-2/3	-2
Photo2	1	-2/3	0
Photo3	-1	4/3	2

Table 2.17 Normalized matrix for the adjusted cosine-based computation

Table 2.18 shows the item-to-item similarity for the three items. Again, Photo1 and Photo3 are strongly negatively correlated, while Photo2 and Photo3 are similar.

	Photo1	Photo2	Photo3
Photo1	1	0.1754	-0.891
Photo2	0.1754	1	.604
Photo3	-0.891	.604	1

Table 2.18 Similarity between items using correlation similarity

Along the same lines, to compute the similarity between users, we subtract the average rating associated with each item in table 2.7. Table 2.19 shows the resulting table. Note that the sum of the values for a column is equal to 0.

	Photo1	Photo2	Photo3
John	1	1	-5/3
Jane	0	-1	1/3
Doe	-1	0	4/3

Table 2.19 Normalized rating vectors for each user

Again, normalizing each of the vectors to unit length leads to table 2.20.

	Photo1	Photo2	Photo3
John	0.4575	0.4575	-0.7625
Jane	0	-0.9486	0.3162
Doe	-0.6	0	0.8

Table 2.20 Normalizing the vectors to unit lengthr

Finally, table 2.21 contains the similarity matrix between the users by taking the dot product of their vectors.

	John	Jane	Doe
John	1	-0.675	-0.884
Jane	-0.675	1	-0.253
Doe	-0.884	-0.253	1.00

Table 2.21 Adjusted cosine similarity matrix for the users

So far in this section, we've looked at how to transform user rating data into a dataset for analysis, and we used three different similarity metrics to compute the similarities between various items and users. The method used for computing the similarity does have an effect on the result. Next, let's look at how this approach can be generalized for other interactions such as voting.

The analysis for using voting information is similar to that for rating. The only difference is that the cell values will be either 1 or –1 depending on whether the user voted for or against the item. The persistence model for representing voting is similar to that developed in the previous section for persisting ratings.

2.4.2 *Intelligence from bookmarking, saving, purchasing Items, forwarding, click-stream, and reviews*

In this section, we quickly look at how other forms of user-interaction get transformed into metadata. There are two main approaches to using information from users' interaction: content-based and collaboration-based.

CONTENT-BASED APPROACH

As shown in figure 2.20, metadata is associated with each item. This term vector could be created by analyzing the content of the item or using tagging information by users, as we discuss in the next chapter. The term vector consists of keywords or tags with a relative weight associated with each term. As the user saves content, visits content, or writes recommendations, she inherits the metadata associated with each.

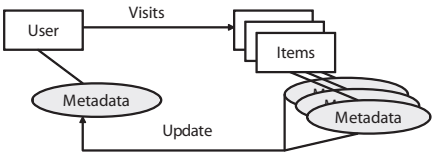

Figure 2.20 A user's metadata vector is created using the metadata vector associated with all the items visited.

This implies that both users and items are represented by the same set of dimensions—tags. Using this representation, one

can compare users with other users, users with items, and items with other items using cosine-based similarity. We see an example of this in the next chapter.

COLLABORATION-BASED APPROACH

The analysis of data collected by bookmarking, saving an item, recommending an item to another user, purchasing an item, or click-stream analysis is similar. To understand this, let's use the following example.

Consider data collected in a window of time. Again, let our three users John, Jane, and Doe bookmark three articles Article 1, Article 2, and Article 3, as shown in table 2.22. We've placed a 1 for articles that the user has bookmarked. This is a sparsely populated dataset as described in section 2.2.6.Using this data, you maybe interested in finding the

	Article 1	Article 2	Article 3
John	1		
Jane		1	1
Doe	1		1

Table 2.22 Bookmarking data for analysis

following answers:

- *What are other items that have been bookmarked by other users who bookmarked the same articles as a specific user?*—When the user is John, the answer is Article 3—Doe has bookmarked Article 1 and also Article 3.
- *What are the related items based on the bookmarking patterns of the users?*

To determine the answer to the last question, it's again useful to invert the dataset to the one shown in table 2.23. Again, the users correspond to the dimensions of the vector for an article. Similarities between two items are measured by computing the dot product between them.

	John	Jane	Doe
Article 1	1		1
Article 2		1	
Article 3		1	1

Table 2.23 Adjusted cosine similarity matrix for the users

The normalized matrix is shown in table 2.24.

	John	Jane	Doe
Article 1	0.7071		0.7071
Article 2		1	
Article 3		0.7071	0.7071

Table 2.24 Normalized dataset for finding related articles

The item-to-item similarity matrix based on this data is shown in table 2.25. According to this, if someone bookmarks Article 1, you should recommend Article 3 to the user, and if the user bookmarks Article 2, you should also recommend Article 3.

	Article 1	Article 2	Article 3
Article 1	1	0	0.5
Article 2	0	1	0.7071
Article 3	0.5	0.7071	1

Table 2.25 Related articles based on bookmarking

A similar analysis can be performed by using information from the items the user saves, purchases, and recommends. You can further refine your analysis by associating data only from users that are similar to a user based on user-profile information. In section 12.3, we further discuss this approach when we discuss building recommendation engines.

In this section, we looked at how we can convert user interactions into intelligence using a simple example of rating photos. We looked at finding items and users of interest for a user. We computed this by using three similarity computations.

2.5 Summary

Services for embedding intelligence in your applications can be divided into two types. First, synchronous services get invoked when the web server processes a request for a user. Second, asynchronous services typically run in the background and take longer to process. An event-driven SOA architecture is recommended for embedding intelligence.

There's a rich amount of information that can be used from user interaction. Metadata attributes can be used to describe items and users. Some interactions such as ratings, voting, buying, recommendations, and so forth are fairly explicit as to whether the user likes the item or not. There are two main approaches to finding items of interest for a user: content-based and collaborative-based. Content-based techniques build a term vector—a multidimensional representation for the item based on the frequency of occurrence and the relative frequency of the terms—and then associate similar items based on the similarities between term vectors. Collaborative-based techniques tend to automate "word of mouth recommendations" to find related items and users.

Metadata associated with users and items can be used to derive intelligence in the form of building recommendation engines and predictive models for personalization, and for enhancing search.

Tagging is another way that users interact with items and provide a rich set of information. We look at tagging next in chapter 3.

2.6 Resources

"All of Web2.0." Chrisekblog. http://chrisek.com/wordpress/2006/10/03/all-of-web-20/

Arthur, Charles. "What is the 1% rule?" July 2006. *The Guardian.* http://technology.guardian.co.uk/weekly/story/0,,1823959,00.html

Baeza-Yates, Ricardo, and Berthier Ribeiro-Neto. *Modern Information Retrieval.* Paperback, May 15, 1999.

Goldberg, David, David Nichols, Brian M. Oki, and Douglas Terry. *Using collaborative filtering to weave an information tapestry*. Communications of the ACM, 35(12):61-70, 1992.

Grossman, David A., and Ophir Frieder. *Information Retrieval: Algorithms and Heuristics (The Information Retrieval Series) (2nd Edition)*. Paperback, Jan 23, 2006.

"Joshua Schachter." Joho the Blog. http://www.hyperorg.com/blogger/mtarchive/berkman_joshua_schachter.html

Kelly, Kevin. "A review of review sites." http://www.kk.org/cooltools/archives/000549.php

Kopelman, Josh. "53,651." Blog. May 2006 . http://redeye.firstround.com/2006/05/53651.html

Pulier, Eric, and Hugh Taylor. 2005. *Understanding Enterprise SOA*. Manning.

Sarwar, Badru, George Karypis, Joseph Konstan, and John Riedl. *Item-based Collaborative Filtering Recommendation Algorithms*. ACM, 2001. http://www10.org/cdrom/papers/519/node1.html

"Should top users be paid?" Stewtopia, Blog. September 11, 2006. http://blog.stewtopia.com/2006/09/11/should-top-users-be-paid/

Thornton, James. *Collaborative Filtering Research Papers*. http://jamesthornton.com/cf/

Wang, Jun, Arjen P. de Vries, and Marcel J.T. Reinders. *Unifying User-based and Item-based Collaborative Filtering Approaches by Similarity Fusion*. 2006. http://ict.ewi.tudelft.nl/pub/jun/sigir06_similarityfuson.pdf.

JAVA/WEB

Collective Intelligence IN ACTION

Satnam Alag

On the internet, harnessing the collective wisdom of users can be a key to success. A new category of programming techniques lets you discover the valuable patterns, inter-relationships, and individual profiles—the collective intelligence—locked in the data people leave behind as they traverse and interact on the web.

> "Use these CI techniques to extract valuable data from your applications."
>
> —From the Foreword by Richard MacManus, ReadWriteWeb

Collective Intelligence in Action starts with the principles of the subject and ideas for building more interactive sites. It then follows a running example to develop an immediately useful Java-based CI toolkit. You'll learn to mine both your own site and the wider web to uncover trends and make practical predictions and recommendations. Along the way, you'll work with numerous helpful APIs and open source toolkits that substantially reduce development effort. This book is written for Java web developers.

What's Inside

- Reusable code for intelligent
  - search
  - recommendations
  - predictions
- Web crawling and text analysis using Lucene and Nutch
- Machine learning using WEKA
- How to implement the Java Data Mining (JDM) standard

About the Author

Satnam Alag is currently the vice president of engineering at NextBio. He was formerly the Chief Software Architect at Rearden Commerce and has a PhD from UC Berkeley.

"It's technical, it's theoretical—but most importantly, it's practical."
—Taran Rampersand
KnowProse.com

"Harness the untapped power of your imagination."
—John Tyler
UBS Investment Bank

"Learn practical, hands-on, machine learning."
—Robi Sen, Twin Technologies

"This is the right book on collective intelligence. I wish I'd had it a few years ago."
—Jérôme Bernard
Elastic Grid LLC

"I recommend this book for any developer of social networking sites."
—Sopan Shewale
TWIKI.NET-Enterprise WIKI

Online access to the author, code samples, and (for owners of this book) a free ebook at www.manning.com/alag or www.manning.com/CollectiveIntelligenceinAction

ISBN-13: 978-1933988313
ISBN-10: 1933988312
54499
9 781933 988313

MANNING $44.99 / Can $44.99 [INCLUDES eBOOK]