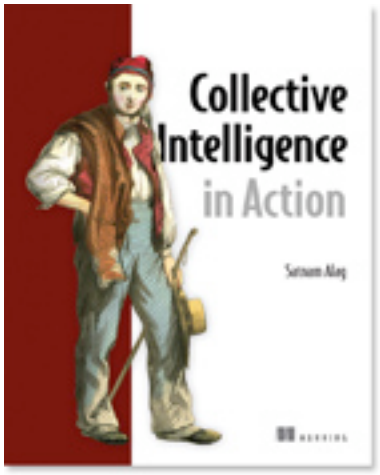
Basics of Algorithms for Applying Collective Intelligence

Excerpted from



Collective Intelligence in Action
EARLY ACCESS EDITION

Satnam Alag
MEAP Release: February 2008
Softbound print: August 2008 (est.) | 425 pages
ISBN: 1933988312

This article is taken from the forthcoming book ***Collective Intelligence in Action****. This article introduces the fields of content and collaborative filtering, and discusses how intelligence is represented and extracted from text.*

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 perhaps 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 they these users have rated highly.

In order to correlate users with content and users with other users 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 typically in the form of unstructured text, it is very helpful to understand how metadata can be developed from unstructured text. We will first begin by abstracting the various types of content, so that the concepts and algorithms can be applied to all of them.

Users and Items

As shown in Figure 1, most applications generally consist of *users* and *items.* An *item* is any entity of interest in your application. Items maybe 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 are looking to connect one user with another, then a user is also a type of item.

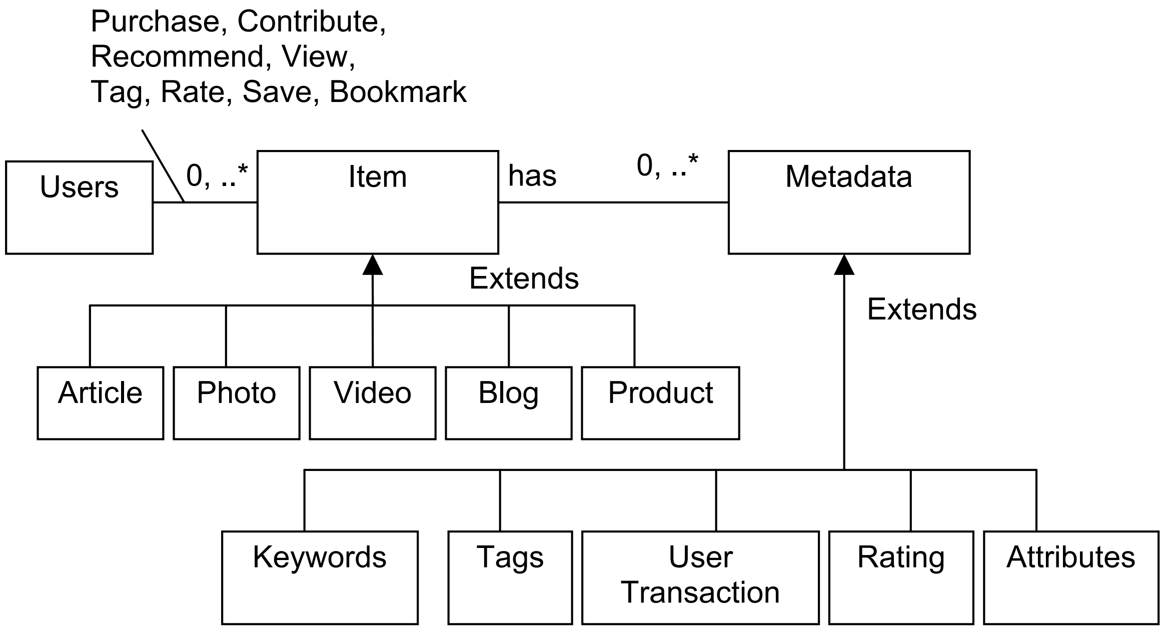


Figure 1: A user interacts with items, which has associated metadata

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, and 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 is 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 shows the three main sources of developing metadata for an item (remember a user is also an item). We shall look at these three sources next.

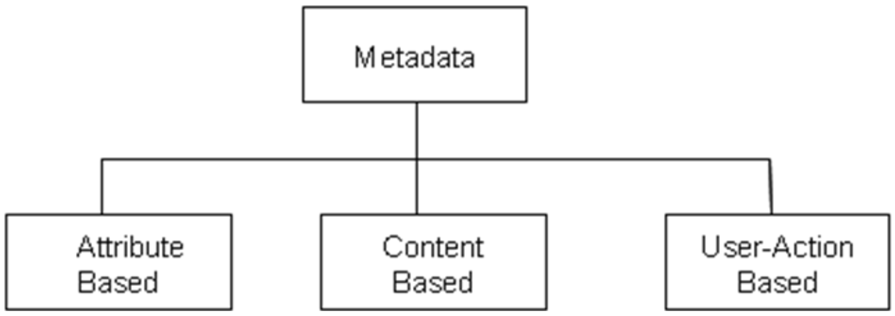


Figure 2: The three forms of generating metadata about an item

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 the age, sex, geographical location, profession, annual income, or education level. Similarly, most non-user items have attributes associated with them. For example, a product may have a price, a name of the author or manufacturer, the geographical location where it is available, the creation or manufacturing date, etc.

Content-Based

Metadata can be generated by analyzing the content of a document. As we will see in the following sections there has been a lot of work done in the area of information retrieval and text mining to extract meta-data associated with unstructured text. The title, sub-titles, keywords, frequency counts of words in a document and across all documents of interest all 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 to what kind of items the user would be interested in; who are the set of users who would be interested in a new *item*, and so on.

Think about *user* and *item* 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 are interested in learning about the likes and dislikes of a user let us next look at representing information related to a user.

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 3 attributes can be *numerical* – have a continuous set of values, for example the age of a user; or *nominal*— have a non-numerical 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.

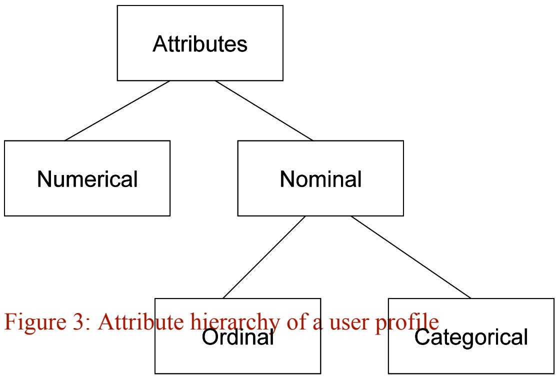


Figure 3: Attribute hierarchy of a user profile

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" when age < 18 years; "Young Person" when 18 < age < 25; and so on. Table 1 has a list of user-attributes that maybe available in your application.

Table 1: 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

In addition to user attributes, the user's interaction in your application gives 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 into your application within a period of time, their 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 provides a summary of some of the ways a user provides valuable information that can be used to add intelligence to your application

Table 2: The many ways users provide valuable information through their interaction

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 ecommerce pages after researching a topic on the site
Profile selections	The choices that the user makes 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 have 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 us take a quick look at how intelligence can be generated by analyzing content and by analyzing the interactions of the users.

Content-Based Analysis and Collaborative Filtering

User-centric applications aim to make the application more valuable for a user by applying CI to personalize the site for the user. 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 one to compute the similarities between a user and an item. Content-based recommendation systems do a good job of finding related items. However, they cannot 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 the 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 those of other similar users and recommend items liked by them. Items that are liked or popular with a certain segment of your user population will appear often in their interaction history – viewed often, purchased often, etc. The frequency of occurrence or ratings provided by users is all 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 one doesn'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. These approaches can have scalability issues and are sensitive to data sparseness. Model-based approaches aim to build a model for prediction using a variety of approaches: linear algebra, probabilistic methods, neural networks, clustering, latent classes, etc. They normally have a fast run-time predicting capabilities.

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

Representing Intelligence from Unstructured Text

This section deals with developing a representation for unstructured text by using the content of the text, and introduces you to *terms* and *term vectors, used* to represent metadata associated with text. Let us consider an example where the text being analyzed is "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 4, 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 related to Java, the word "Java" will appear very often. Less frequent words such as "Spring", "Hibernate", "Intelligence" will be a lot more discriminating in finding items that have these terms in common. 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:

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

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

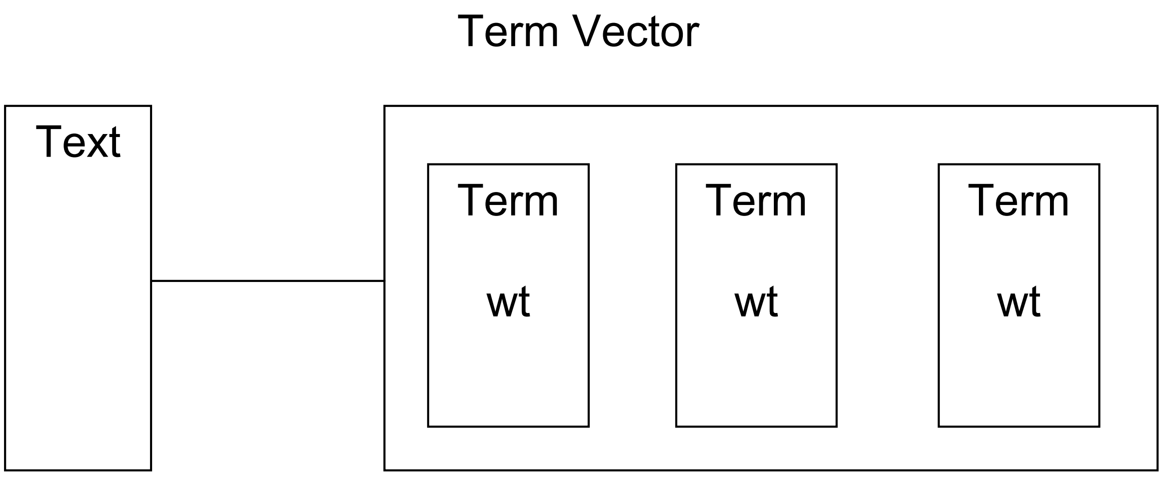


Figure 4: Term vector representation of text

Commonly occurring terms such as “a”, “the”, “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 converted to their lower case. 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* maybe used to inject terms into the representation.

Figure 5 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

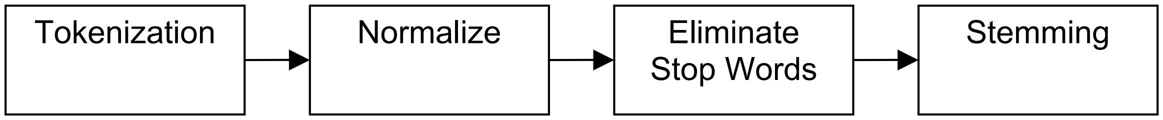


Figure 5: Typical steps involved in analyzing text

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 above approach for generating metadata is content based. You can also generate metadata by analyzing user interaction with the content.

So far we have looked at what a term vector is and we've gained some basic knowledge on how they are computed. Let us next look at how to compute *similarities* between them. An item which is very similar to another item will have a high value for the computed similarity metric. Similarly, an item whose term vector has a very 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.

Computing Similarities

A term vector consists of a vector where the direction of the vector is the magnitude of the weights for each of the terms. The term vector has multiple dimensions – thousands to even millions, depending on your application. Multi-dimension vectors are difficult to visualize, but the principles used can be illustrated by using a two-dimensional vector as shown in Figure 6.

Given a vector representation, we will normalize the vector such that its length is of size 1 and compute the similarity between vectors as a means to compare two vectors. For now just think of vectors as a means to represent information with a well-developed math to compute similarities between them.

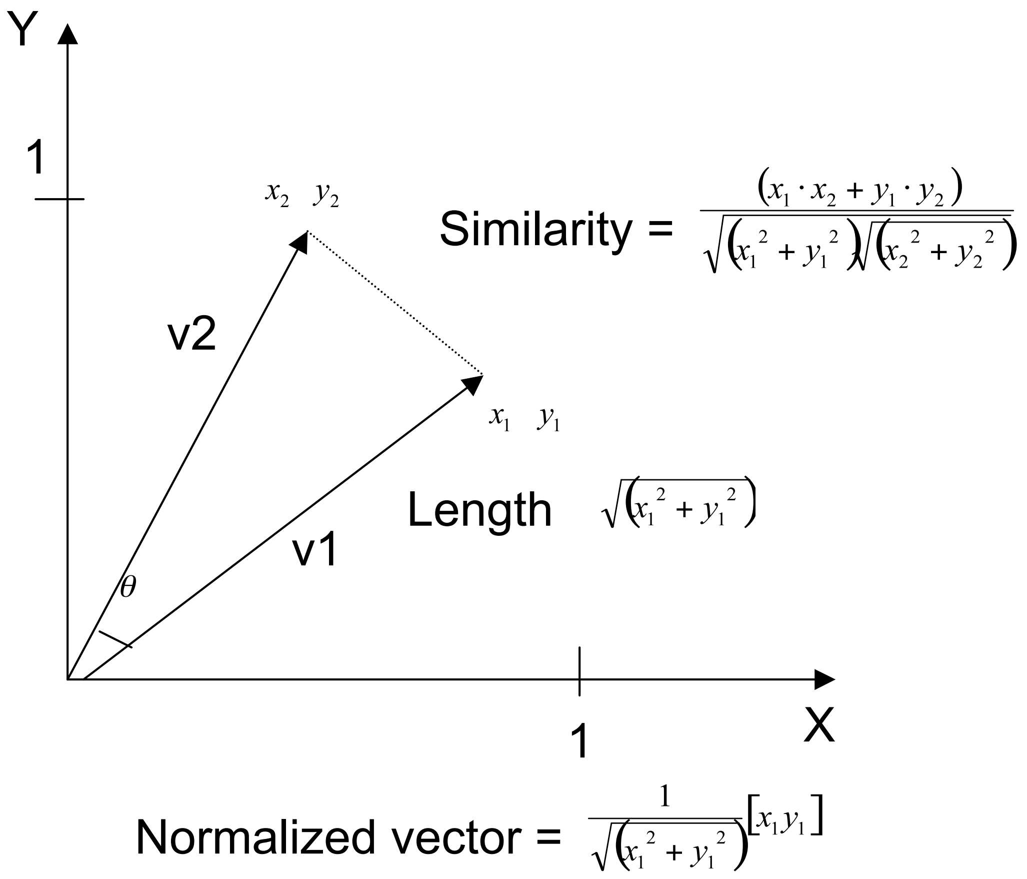

Y
X
1
1
v1
v2
θ
$x_2 \quad y_2$
$x_1 \quad y_1$
Similarity = $\frac{(x_1 \cdot x_2 + y_1 \cdot y_2)}{\sqrt{(x_1^2 + y_1^2)}\sqrt{(x_2^2 + y_2^2)}}$
Length $\sqrt{(x_1^2 + y_1^2)}$
Normalized vector = $\frac{1}{\sqrt{(x_1^2 + y_1^2)}}[x_1 y_1]$

Figure 6: Two dimensional vectors, v1 and v2

So far we have looked at the use of the term vector to represent metadata associated with content and 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 and high-dimensional sparsely populated datasets – this dataset is similar in characteristics to a term vector.

Types of Dataset

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

Let there be three users - John, Joe, and Jane. Each of them has three attributes: age, sex, and average number of minutes spent on the site. Table 3 shows the value for the various attributes for these users. This data can be used for clustering[1] to build a predictive model. 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 attribute "sex" has no effect in the prediction. In this made-up example, a simple linear model is adequate to predict the number of minutes spent (= 50 – age of user).

Table 3 Dataset with small number of attributes

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

This is the first kind of dataset, which is densely populated. Note that the number of rows in the dataset will increase as we have more users in the dataset. It has the following properties

- Has more number of rows than the number of columns. The number of rows is typically a few orders of magnitude more than the number of columns.
- The dataset is richly populated – there is a value for each cell

The second 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 last one week. We will consider the set of users who have 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 4 shows the dataset created by having a 1 in the cell if a user has viewed the video. This representation is useful to find similar users and is known as the User-Item matrix.

Table 4: Dataset with large number of attributes

	Video 1	**Video 2**	**…**	**…**	**Video n**
John	1				
Joe	1	1			
Jane					1

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.

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 are the number of videos at YouTube
- The dataset is sparsely populated with non-zero entries in a few columns
- You can visualize this dataset as a multi-dimensional vector, with the columns corresponding to the dimensions and the cell entry corresponding to the weight associated for that dimension

Note the similarity of this dataset with the term vector we introduced earlier. Let there be *m* terms that occur in all our documents. Then the term vectors corresponding to all our documents has the same characteristics as the above dataset as shown in Table 5.

Table 5: Sparsely populated dataset corresponding to term vectors

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