Feature Extraction

Dragoş-Anton Manolescu
manolesc@uiuc.edu

May 11, 1998

1 Context

Digital libraries offer access to a large collection of documents represented in electronic format. According to Bruce Schatz [Sch97]:

> A digital library enables users to interact effectively with information distributed across a network. These network information systems support search and display of items from organized collections.

An increasing number of users discover online information retrieval and interactive searches. Once comfortable with the new tools, they demand new materials to be available in digital libraries. This requires obtaining digital representations of documents. Since the process is getting cheaper and faster, extending a digital library is not a difficult task.

Obviously, this increase in the amount of information has a strong impact on the supporting software. Consider for example the case of searching—text scanning. This is a simple but basic operation for any digital library. Algorithms for full text scanning include regular expression searching, signature files or inversion [FO95]. However, these approaches do not scale well. None of them is applicable for the amounts of data typical to digital libraries.

2 Problem

Large amounts of information or complex data are typical requirements for current applications. How does software handle this information in a suitable way?

3 Forces

- Digital libraries handle large amounts of information;
- Multimedia databases handle complex information;
- Information retrieval systems require small space overhead, but also low computational overhead for queries and insertions;
- Fast response time is important;
- Designing feature extraction functions and scalable multidimensional indexing methods is hard.

4 Solution

Compute an alternative, simpler representation of data. The representation contains only the information that is relevant for the problem at hand. This computation is actually a function. It maps from the problem space into a feature space. For this reason it is also called “feature extraction function.”

A typical feature extraction function for text documents is (automatic) indexing. The function maps each document into a point in the *k*-dimensional keyword (or feature) space—*k* is the number of keywords. Automatic indexing consists of the following steps [FO95]. First, it removes the common words like “and,” “the,” etc. Next, the remaining words are reduced to their stem. For instance, both “computer” and “computation” are reduced to “comput.” Then, a dictionary of synonyms helps to assign each word-stem to a concept class. Finally, the method builds the vector in keyword space. Each vector element gives the coordinate in one of the *k* dimensions and corresponds to a concept class. There are several options for computing these values. Binary document vectors use only two values to indicate the presence of absence of a term. Vectors based on weighting functions use values corresponding to term frequency, “specificity,” etc. Figure 1 illustrates how document indexing maps from document space to 3-dimensional keyword space.

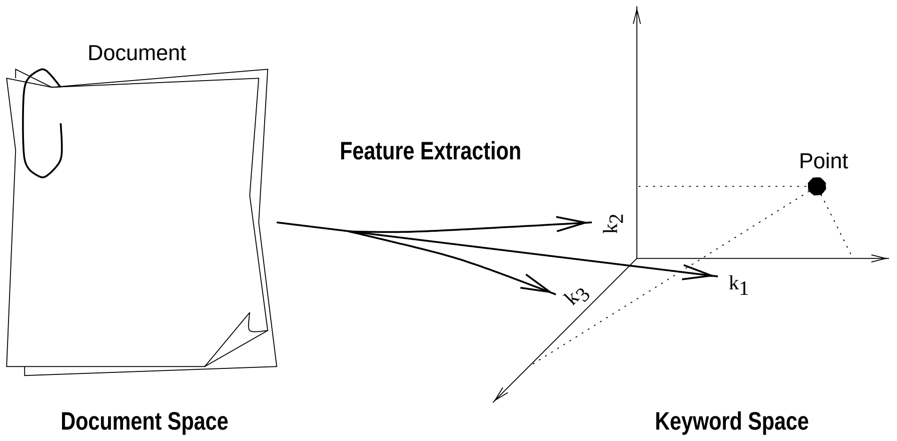

Figure 1: Mapping a document into a 3-dimensional keyword space.

Typically, feature extraction maps from a *larger* problem space into a *smaller* feature space. Consider the previous example of document indexing. In document space, one document contains a large number of words. Searching a collection of documents requires many string matching operations. However, in keyword space, documents correspond to multi-dimensional vectors. Searching for documents that contain a given set of keywords involves comparing vectors. This is a simple operation, much faster than string matching. Therefore, feature extraction (i) enables scalable solutions for problems that deal with large amounts of information.

Domain mappings are a widespread technique in mathematics. Usually they map from a *complex* domain into a *simpler* one—here, complexity refers to the operations within the domain. A well-known example is the operational method for the solution of differential equations [BS97]. The method consists in going from a differential equation, by means of an integral transformation, to a transformed equation. The transformed equation is easier to solve than its differential counterpart. Two possible integral transformations are the Laplace transform and the *z* transform.

In a software context, mapping from problem space into feature space also enables computers to manipulate complex information. Digital images are one example. Current image databases employ this pattern to obtain simplified representations for images. Unlike the typical domain mappings from mathematics, these representations are lossy. They consider only a *subset* of the image features. Common features for images are color histograms, textures, shapes or a combination of these. Therefore, feature extraction (ii) enables software systems to process different types of complex information without “understanding” the contents.

When it maps from a large problem space, feature extraction considers only a few “significant” features in feature space, discarding the rest. This truncation yields a non-injective mapping. For example, two documents can map into the same point in keyword space. However, this does not mean that they are identical. Since the function is not injective, there is no inverse mapping. Several points in problem space can map into a single point in feature space. This property affects all applications that employ this pattern to provide answers to queries. The solution is to add a post-processing step that filters out the “false alarms.” Since the typical number of false alarms is small, the post-processing step usually performs a sequential search to eliminate them.

Besides post-processing, this pattern requires some other additional processing. For feature extraction to be applicable, the features of the working set of items (documents, images, etc.) have to be available. The system computes the coordinates in feature space for any new items. Each insertion needs this extra step. Another operation that changes is query processing. The fundamental idea of feature extraction is to perform all computations in a smaller, simpler space. Processing then takes place in this space. Consequently, answering a query requires its representation in feature space as well. To summarize, feature extraction complicates processing since it requires two additional processing blocks—Figure 2.

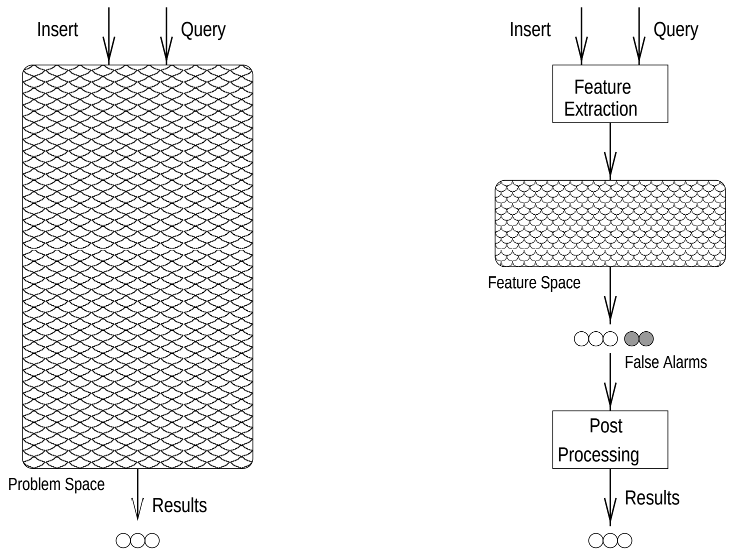

Figure 2: Insertion and query processing without and with feature extraction.

Information retrieval (IR) is one of the domains that employs feature extraction extensively. For such applications, the objective is to minimize response times for different sorts of queries. Performance depends on how fast the system performs searches in the multidimensional feature space. Therefore, the choice of a multidimensional indexing method is critical. However, this is not an easy task. Good unidimensional indexing methods scale exponentially for high dimensionalities, eventually reducing to sequential scanning [AFS93]. Consequently, they are applicable only when a small number of dimensions is sufficient to differentiate between data items. The solution is to use R-tree variants, which are usable for a fairly large number of dimensions.

The hard part about this pattern is obtaining a suitable feature extraction function. Obviously, this is domain and problem dependent. Feature extraction is mainly used for information retrieval and data mining.

One of the main requirements for these domains is correctness. A query should return all the qualifying information, without any "misses." The "false alarms" due to the non-injective mapping are not a problem: post-processing (Figure 2) removes them. However, a formal proof is required to demonstrate correctness. Alternatively, a domain-specific algorithm may automatically construct a correct feature-extraction function for a given problem. For example, [FL95] describes such an algorithm for indexing, data-mining and visualization of traditional and multimedia datasets.

The Discrete Fourier Transform (DFT) is an example of feature extraction function. This function is suitable for pink noise "signals," whose energy spectrum follows $O(f^{-1})$. A wide range of data (e.g., stock prices, musical scores, etc.) fits this description. Consequently, DFT is usable in many different domains. The transform is particularly useful for similarity search [AFS93]. Its properties guarantee the completeness of feature extraction—i.e., correctness. Since DFT is orthonormal (i.e., distance-preserving), the distance between two data items in problem space is the same as the distance between their corresponding points in feature space. Therefore, DFT is applicable with any similarity measure that can be expressed as the Euclidean distance between feature vectors in some feature space.

The next step after finding a feature extraction function is to decide on the features to consider further. As stated before, not all features are used. For example, systems that use DFT keep only a few low-frequency coefficients. This "lossy" part of the pattern ensures that feature space is smaller than problem space. Deciding on the number of features is a tradeoff between accuracy and speed. At one extreme, the system is "lossless" and keeps all features. This ensures no false alarms. However, searching a large (feature) space is what the pattern is trying to avoid. At the other extreme, only one feature is used. In this case, the degenerate searching in feature space is fast—it simply returns everything. Post-processing takes a long time though, since it filters all data items. Therefore, the number of features determines the balance between searching time in feature space and the post-processing time.

Choosing a suitable multidimensional indexing method is the last step. The choice depends on the number of features—dimensions of the feature space. Many methods are available for indexing low dimensionality domains. However, as the number of dimensions grows, they degenerate into sequential scanning. R-tree variants (e.g., R*-trees [BKSS90] and SS-trees [WJ96]) offer good performance for a fairly large number of dimensions.

To summarize, the feature extraction pattern has the following **benefits** (✔) and **liabilities** (✘):

✔ Large amounts of data are manageable and no longer bring software to its knees. Compared to sequential searching, applications using this pattern obtain an increasingly better performance as the volume of data increases [AFS93].

✔ Software can manipulate complex information without having to decode its semantics. This is key to implementing multimedia databases.

✔ Users can easily refine queries. Once results are available, they mark only the ones that are relevant. The system adjusts the original query and performs a new search. Provided that the user's feedback is consistent, such queries converge in a few iterations. This mode of operation is also known as "relevance feedback" [SB88]. In feature space, relevance feedback is simple and consists of adding the selected vectors to the query vector.

✘ Efficient search in feature space requires multidimensional indexing methods. Not all good indexing methods scale well with the number of dimensions. Obtaining an efficient and scalable multidimensional index structure is not easy.

✘ Inserting new items and answering queries require additional processing. The architect has to determine the right balance between the number of features and the post-processing time.

5 Implementation Notes

[Not yet written; will add some Smalltalk code.]

6 Examples

Feature extraction is not new. One of the pioneers of this pattern was Gerald Salton. He employed it in the SMART system [Sal69] at Cornell, a long time before the term "digital library" was coined.

Professional recruiters use feature extraction as well. Hundreds of resumés are first scanned and particular keywords or patterns are searched for. Next, each individual in the recruiter's database is represented as a point in a multidimensional "proficiencies" space. Here, each dimension corresponds to some skill or characteristic. For example, knowledge of a particular programming language may correspond to one dimension, while team-work experience and willingness to travel to others. A company looking for new employees supplies a "wish-list" to the recruiter. Following the same procedure, this maps into a region in the proficiencies space. All the individuals within that region are potential candidates for the company's openings.

Since all the information currently produced is available in electronic format, many other fields use feature extraction. These include telecommunications, multimedia, medicine, business, etc. However, despite its widespread use, few studies document this aspect. Three documented examples from different domains follow.

1. The authors of [KJF97] use feature extraction to perform ad-hoc queries on large datasets of time sequences. The data consists of customer calling patterns from AT&T and is in the order of hundreds of gigabytes. Calling patterns are stored in a matrix where each element has a numeric value. The rows correspond to customers (in the order of hundreds of thousands) and the columns correspond to days (in the order of hundreds).

   In this case, the problem is the compression of a matrix which consists of time sequences, while maintaining "random access." Generic compression algorithms (e.g., Lempel-Ziv-Huffman, etc.) achieve good compression ratios. However, queries do not work on compressed data and require uncompression. This is not viable for the amounts of data corresponding to calling patterns.

   Feature extraction avoids the need for uncompression. The function for feature extraction is singular value decomposition (SVD). This truncates the original matrix by keeping only the principal components of each row and achieves a 40:1 compression ratio. Therefore, SVD maps the large customer calling pattern matrix into a smaller matrix in feature space. The compressed format is lossy, but supports queries on specific cells of the data matrix, as well as aggregate queries. For example, a query on a specific cell is "What was the amount of sales to ACME, Inc. on August 16th, 1997?" The method yields an average of less than 5% error in any data value.

2. Large amounts of data are also typical in the financial domain. Feature extraction provides a fast way for searching stock prices [FRM94] and is useful for any other time-series databases (e.g., weather, geological, environmental, astrophysics or DNA data).

   The problem here is to find a fast method for locating subsequences in time-series databases. The system needs to answer queries like "Find companies that have similar sales patterns with ACME, Inc." Sequential scanning is not viable for several reasons. First, it does not scale for large amounts of data. Second, it has a large space overhead, since each search requires the availability of the entire time sequence.

Feature extraction provides a fast and dynamic solution. In this case, the feature extraction function is an *n*-point Discrete Fourier Transform (DFT). This maps each time-series into a trace in a multi-dimensional feature space. Since the method considers only a few low-frequency coefficients, queries return a superset of the actual results. However, post-processing eliminates all "false alarms." The space overhead is small, and the response times are orders of magnitude faster than a sequential scan.

3. Besides handling large amounts of data, feature extraction is also applicable for software systems that manipulate complex information. Digital images are a typical example. Computers are good at manipulating the basic image components like luminance and chrominance. However, decoding the semantics of the information contained within an image (its contents) is still a research issue.

   In [PF97], the authors employ this pattern for similarity searching in image databases. The problem is to support queries by image content for a database of medical images. A typical query is "Find all X-rays that are similar to Bob's X-ray." This problem has the following requirements. First, it needs to be accurate. The results of a query must return all qualifying images. Second, query formulation must be flexible and convenient. The user should be able to specify queries by example, through a GUI. Finally, response times and scalability are important. Performance must remain consistently better than sequential scanning as the size of the database grows.

   The system represents image content by attributed relational graphs holding features of objects and relationships between them. This representation relies on the (realistic) assumption that a fixed number of objects are common in many images—e.g., liver, lungs, heart, etc. All these common objects are "labeled." The method considers five features for each labeled object in the image. Five features are sufficient for medical purposes. However, the method can handle any other additional features that the domain expert may want to consider. This approach outperforms sequential scanning and scales well with the size of the database.

7 Related Patterns

- Most systems that use feature extraction provide answers to different types of queries. In such systems, you can think of feature extraction as an "approximate proxy." Each query returns a proxy object [GHJV95]. This represents a "gateway" to feature space and encapsulates the post-processing step.

- The pattern is independent of the feature extraction function. Domain experts select any function that is suitable for some problem, ensuring that it produces correct results. A flexible solution is to represent the various feature extraction functions as strategy objects [GHJV95].

References

[AFS93] Rakesh Agrawal, Christos Faloutsos, and Arun Swami. Efficient similarity search in sequence databases. In *Proc. FODO conference*, 1993.

[BKSS90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In Hector Garcia-Molina and H. V. Jagadish, editors, *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 322–331, Atlantic City, NJ, May 1990.

[BS97] I. N .Bronshtein and K. A. Semendyayev. *Handbook of Mathematics*. Springer-Verlag, third edition, 1997.

[FL95] Christos Faloutsos and King-Ip (David) Lin. Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proc. ACM SIGMOD*, pages 163–174, May 1995. Also available as technical report (CS-TR-3383, UMIACS-TR-94-132; Institite for Systems Research: TR 94-80).

[FO95] Christos Faloutsos and Douglas W. Oard. A survey of information retrieval and filtering methods. Technical Report 3514, Department of Computer Science, University of Maryland, College Park, MD 20742, August 1995.

[FRM94] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. Fast subsequence matching in time-series databases. In *Proc. SIGMOD Conference*, pages 419–429, 1994.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns—Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[KJF97] Flip Korn, H. V. Jagadish, and Christos Faloutsos. Efficiently supporting ad hoc queries in large datasets of time sequences. In *Proc. SIGMOD Conference*, 1997.

[PF97] Euripides G. M. Petrakis and Christos Faloutsos. Similarity searching in medical image databases. *IEEE Trans. on Knowledge and Data Engineering*, 9(3):435–447, May/June 1997.

[Sal69] Gerard Salton. Interactive information retrieval. Technical Report TR69-40, Cornell University, Computer Science Department, August 1969.

[SB88] Gerard Salton and Chris Buckley. Improving retrieval performance by relevance feedback. Technical Report TR88-898, Cornell University, Computer Science Department, February 1988.

[Sch97] Bruce R. Schatz. Information retrieval in digital libraries: Bringing search to the net. *Science*, 275:327–334, January 1997.

[WJ96] David A. White and Ramesh Jain. Similarity indexing with the SS-tree. In *Proceedings of the 12th International Conference on Data Engineering*, pages 516–523. IEEE Computer Society, February 1996.