F

F_1-Measure

The F_1-measure is used to evaluate the accuracy of predictions in two-class (binary) ►classification problems. It originates in the field of information retrieval and is often used to evaluate ►document classification models and algorithms. It is defined as the harmonic mean of ►precision (i.e., the ratio of ►true positives to all instances predicted as positive) and ►recall (i.e., the ratio of true positives to all instances that are actually positive). As such, it lies between precision and recall, but is closer to the smaller of these two values. Therefore a system with high F_1 has both good precision and good recall. The F_1-measure is a special case of the more general family of evaluation measures:

$$F_\beta = (\beta^2 + 1)precisionrecall/(\beta^2 precision + recall)$$

Thus using $\beta >$ increases the influence of precision on the overall measure, while using $\beta < 1$ increases the influence of recall. Some authors use an alternative parameterization,

$$F_\alpha = 1/(\alpha/precision + (1t\alpha)/recall)$$

which, however, leads to the same family of measures; conversion is possible via the relationship $\alpha = 1/(\beta^2 + 1)$.

False Negative

In a two-class problem, a ►classification ►model makes two types of error: ►false positives and false negatives. A **false negative** is an example of positive class that has been incorrectly classified as negative. See ►confusion matrix for a complete range of related terms.

False Positive

In a two-class problem, a ►classification ►model makes two types of error: false positives and ►false negatives. A **false positive** is an example of negative class that has been incorrectly classified as positive. See ►confusion matrix for a complete range of related terms.

Feature

►Attribute

Feature Construction

►Data Preparation

Feature Construction in Text Mining

Janez Brank, Dunja Mladenić,
Marko Grobelnik
Jožef Stefan Insitute
Ljubljana, Slovenia

Synonyms

Feature generation in text mining

Definition

Feature construction in ►text mining consists of various techniques and approaches which convert textual data into a feature-based representation. Since traditional machine learning and data mining techniques are generally not designed to deal directly with textual data, feature construction is an important preliminary step in text mining, converting source documents

Claude Sammut & Geoffrey I. Webb (eds.), *Encyclopedia of Machine Learning*, DOI 10.1007/978-0-387-30164-8,

into a representation that a data mining algorithm can then work with. Various kinds of feature construction approaches are used in text mining depending on the task that is being addressed, the data mining algorithms used, and the nature of the dataset in question.

Motivation and Background

Text mining is the use of machine learning and data mining techniques on textual data. This data consists of natural language documents that can be more or less structured, ranging from completely unstructured plain text to documents with various kinds of tags containing machine-readable semantic information. Furthermore, documents may sometimes contain hyperlinks that connect them into a graph. Since most traditional machine learning and data mining techniques are not directly equipped to deal with this kind of data, an important first step in text mining is to extract or construct features from the input documents, thereby obtaining a feature-based representation which is suitable for handling with machine learning and data mining algorithms. Thus, the task of feature construction in text mining is inextricably connected with text mining itself and has evolved alongside it. An important trend over the years has been the development of techniques that do not process each document in isolation but make use of a corpus of documents as a whole, possibly even involving external data or background knowledge in the process.

Documents and text data provide for valuable sources of information and their growing availability in electronic form naturally led to application of different analytic methods. One of the common ways is to take a whole vocabulary of the natural language in which the text is written as a feature set, resulting in several tens of thousands of features. In a simple setting, each feature gives a count of the word occurrences in a document. In this way text of a document is represented as a vector of numbers. The representation of a particular document contains many zeros, as most of the words from the vocabulary do not occur in a particular document. In addition to the already mentioned two common specifics of text data, having a large number of features and a sparse data representation, it was observed that frequency of words in text in general follows Zipf's law – a small subset of words occur very frequently in texts while a large number of words occur only rarely. ▶Document classification takes these and some other data specifics into account when developing the appropriate classification methods.

Structure of Learning System

In a learning or mining system that deals with textual data, feature construction is usually one of the first steps that is often performed alongside typical preprocessing tasks such as data cleaning. A typical output of feature construction are feature vectors representing the input documents; these vectors themselves then form the input for a machine learning or data mining algorithm. On the other hand, sometimes feature construction is more closely integrated into the learning algorithm itself, and sometimes it can be argued that the features themselves are the desired output that is the goal of the text mining task.

Solutions

At the lowest level, text is represented as a sequence of bytes or other elementary units of information. How these bytes are to be converted into a sequence of characters depends on the *character encoding* of the text. Many standard encodings exist, such as UTF-8, the ISO-8859 family, and so on. Often, all the texts that appear as input for a specific text mining task are in the same encoding, or if various encodings are used they are specified clearly and explicitly (e.g., via the Content-Type header in the HTTP protocol), in which case the problem of conversion is straightforward. In the case of missing or faulty encoding information, various heuristics can be used to detect the encoding and convert the data to characters; it is best to think of this as a data cleaning and preprocessing step.

Word-Based Features

When we have our text represented as a sequence of characters, the usual next step is to convert it into a sequence of words. This is usually performed with heuristics which depend to some extent on the language and underlying character set; for the purposes of segmentation of text into words, a word is thought of as a sequence of alphabetic characters delimited by

whitespace and/or punctuation. Some efforts to standardize word boundary detection in a way that would work reasonably well with a large set of natural languages have also been made (see, e.g., the Unicode Standard Annex #29, *Unicode Text Segmentation*). For many (but not all) text mining tasks, the distinction between upper and lower case (if it is present in the underlying natural language) is largely or entirely irrelevant; hence, all text is often converted into lower case at this point. Another frequently used preprocessing step is *stemming*, whereby each word is replaced by its stem (e.g., *walking* → *walk*). The details of stemming depend on the natural language involved; for English, a relatively simple set of heuristics such as Porter's stemmer is sufficient. Instead of stemming, where the ending is chopped off the word, one can apply a more sophisticated transformation referred to as lemmatization that replaces the word by its normalized form (lemma). Lemmatization is especially relevant for natural languages that have many different forms of the same word (e.g., several cases, gender influence on verb form etc.). Efforts have also been made to discover stemming rules or lemmatization rules automatically using machine learning techniques (Plisson, Lavrač, Mladenić, & Erjavec, 2008).

The individual words can themselves be thought of as features of the document. In the feature-vector representation of a document d, the feature corresponding to the word w would tell something about the presence of the word w in this document: either the frequency (number of occurrences) of w in d, or a simple binary value (1 if present, 0 if absent), or it can further be modified by, e.g., the ▶TF-IDF weighting. In this kind of representation, all information about the word order in the original document is lost; hence, it is referred to as the "bag of words" model. For many tasks, the loss of word order information is not critical and the bag of words model is a staple of information retrieval, document classification, and many other text-related tasks. A downside of this approach (and many other word-based feature construction techniques) is that the resulting number of features can be very large (there are easily tens of thousands of different words in a mid-sized document corpus); see ▶Feature Selection in Text Mining.

Clearly, ignoring the word order completely can sometimes lead to the loss of valuable information. Multi-word phrases sometimes have a meaning that is not adequately covered by the individual words of the phrase (e.g., proper names, technical terms, etc.). Various ways of creating multi-word features have been considered. Let d be a document consisting of the sequence of words $(w_1, w_2, \ldots, w_m)$ (note that, this sequence might already be an output of some preprocessing operations, e.g., the removal of stopwords and of very infrequent words.). Then an *n-gram* is defined as a sequence of n adjacent words from the document, i.e., $(w_i, w_{i+1}, \ldots, w_{i+n-1})$. We can use n-grams as features in the same way as individual words, and indeed a typical approach is to use n-grams for all values of n from 1 to a certain upper limit (e.g., 5). Many of the resulting n-grams will be incidental and irrelevant, but some of them may be valuable and informative phrases; whether the text mining algorithm will be able to profit from them depends a lot on the algorithm used, and feature selection might be even more necessary than in the case of individual words. A related problem is the explosion of the number of features; if the number of different words in a corpus grows approximately with the square root of the length of the corpus (Heaps' law), the number of different n-grams is more likely to grow nearly linearly with the length of the corpus. The use of n-grams as features has been found to be beneficial, e.g., for the classification of very short documents (Mladenić & Grobelnik, 2003).

Further generalization of n-grams is possible by removing the requirement that the words of the n-gram must appear adjacently; we can allow them to be separated by other words. The weight of an occurrence of the n-gram is often defined as decreasing exponentially with the number of intervening separator words. Another direction of generalizing n-gram is to ignore the order of words within the n-gram; in effect one treats n-grams as bags (multisets) instead of sequences. This results in features sometimes called *loose phrases* or *proximity features* (i.e., every bag of words up to a certain size, occurring in sufficiently close proximity to each other, is considered to be a feature). These generalizations greatly increase the feature space as well as the number of features present in any individual document, so the risk of computational intractability is greatly increased; this can sometimes be alleviated through the use of kernels (see below).

Character-Based Features

Instead of treating the text as a sequences of words, we might choose to treat it as a sequence of characters. A sequence of n characters is also known as an n-graph. We can use n-graphs as features in the representation of text in a way analogous to the use of n-grams in the previous subsection. The weight of the feature corresponding to a particular n-graph in the feature vector of a particular document d will typically depend on the number of occurrences of that n-graph in the text of d. Sometimes noncontiguous occurrences of the n-graph are also counted (i.e., occurrences where characters from the n-graph are separated by one or more other characters), although with a lower weight; this is can be done very elegantly with kernel methods (see below). Feature selection and TF-IDF style weighting schemes can also be used as in the case of n-grams. Whether an n-graph-based representation offers any benefits compared to an n-gram-based one depends largely on the dataset and task in question. For example, the classification of English documents the usefulness of n-graphs has been found to be dubious, but they can be beneficial in highly agglutinative languages where an individual word can consist of many morphemes and it is not really useful to treat a whole word as an individual unit of information (as would be the case in a word-based feature representation). In effect, the use of n-graphs provides the learner with cheap access to the sort of information that would otherwise require more sophisticated NLP technologies (stemming, parsing, morpheme analysis, etc.); the downside is that a lot of the n-graph features are merely noise (Lodhi, Saunders, Shawe-Taylor, Cristianini, & Watkins, 2002). For some application, word suffixes can be particularly useful features, e.g., to learn lemmatization rules (Mladenić, 2002; Plisson et al., 2008).

Kernel Methods

Let ϕ be a function which assigns, to a given document d, a feature vector $\phi(d)$ from some feature space F. Assume furthermore that a dot product (a.k.a. inner product) is defined over F, denoted by $\langle \cdot, \cdot \rangle_F$. Then the function K defined by $K(d_1, d_2) = \langle \phi(d_1), \phi(d_2) \rangle_F$ is called a *kernel function*. It turns out that many machine learning and data mining methods can be described in a way such that the only operation they need to do with the data is to compute dot products of their feature vectors; in other words, they only require us to be able to compute the kernel function over our documents. These approaches are collectively known as ►*kernel methods*; a well-known example of this is the ►support vector machine (SVM) method for supervised learning, but the same principle can be used in ►clustering as well. An important advantage of this approach is that it is often possible to compute the kernel function K directly from the documents $d_{1,2}$ without explicitly generating the feature vectors $\phi(d_{1,2})$. This is especially valuable if the feature space is untractably large. Several families of kernel functions for textual data have been described in the literature, corresponding to various kinds of n-graph and n-gram based features (Brank, 2006; Lodhi et al., 2002).

Linear Algebra Methods

Assume that a corpus of n documents have already been represented by d-dimensional real feature vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n \in R^d$. If we select some direction $\mathbf{y} \in R^d$ and project a vector $\mathbf{x}_i$ in this direction, the resulting value $\mathbf{y}^T \mathbf{x}_i / ||\mathbf{y}||$ is in effect a new feature describing the document i. In other words, we have constructed a new feature as a linear combination of the existing features. This leads to the question of how to select one or more suitable directions $\mathbf{y}$; various techniques from linear algebra and statistics have been proposed for this.

A well-known example of this is *principal component analysis* (PCA) in which one or more new coordinate axes $\mathbf{y}$ are selected in such a way that the variance of the original vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$ in the directions of the new coordinate axes is maximized. As it turns out, this problem is equivalent to computing the principal eigenvectors of the covariance matrix of the original dataset.

Another technique of this sort is *latent semantic indexing* (LSI) (Deerwester, Dumais, Furnas, Landauer, & Harshman, 1990). Let X be a $d \times n$ matrix with $\mathbf{x}_1, \ldots, \mathbf{x}_n$ as its columns (a.k.a. the *term-document matrix*). LSI uses singular value decomposition (SVD) to express X as the product of three matrices, $T \cdot S \cdot D$, where T is a $d \times r$ orthonormal matrix, D is a $r \times n$ orthonormal matrix, and S is a $r \times r$ diagonal matrix containing the singular values of X. Here, r denotes the rank of the original matrix X. Let $T^{(m)}$ be the matrix consisting of the left m columns of T, let $D^{(m)}$ be the matrix consisting of the top m rows of D, and let $S^{(m)}$ be

the top left $m \times m$ submatrix of S. Then it turns out that $X^{(m)} = T^{(m)}S^{(m)}D^{(m)}$ is the best rank-m approximation of the original X (best in the sense of minimizing the Frobenius norm of $X - X^{(m)}$). Thus, the i-th column of $D^{(m)}$ can be seen as a vector of m new features representing the i-th document of our original dataset, and the product $T^{(m)}S^{(m)}$ can be seen as a set of m new coordinate axes. The new feature vectors (columns of $D^{(m)}$) can be used instead of the original vectors $\mathbf{x}_i$.

Canonical correlation analysis (CCA): Sometimes several vector representations are available for the same document d_i; for example, we might have the same text in two different languages, giving rise to two feature vectors, e.g., $\mathbf{x}_i \in R^d$ and $\mathbf{y}_i \in R^{d'}$. Given such a "parallel corpus" of pairs $(\mathbf{x}_i, \mathbf{y}_i)$, $i = 1 \ldots n$, it is sometimes desirable to convert both types of representations to a "common denominator." In other words, we want to find a set of r new coordinate axes in $\mathbf{x}$-space (say the columns of $U \in R^{d \times r}$) and a set of r new coordinate axes in $\mathbf{y}$-space (say the columns of $V \in R^{d' \times r}$) such that the j-th column of U has a similar role in $\mathbf{x}$-space as the j-th column of V has in $\mathbf{y}$-space, for all j. This can be formulated as an optimization problem: find U and V such that the correlation between $U^T\mathbf{x}_i$ and $V^T\mathbf{y}_i$ (i.e., the projections of $\mathbf{x}_i$ and $\mathbf{y}_i$ onto the new sets of axes) is maximized. Once we have suitable matrices U and V, we can convert any feature vector from the original $\mathbf{x}$-space or $\mathbf{y}$-space into a common new r-dimensional space. This makes it easier to deal with multi-lingual corpora, allowing us, e.g., to retrieve documents in language $\mathbf{x}$ as a response to a query in language $\mathbf{y}$, or vice versa. The same techniques are applicable in multimodal scenarios (i.e., $\mathbf{x}_i$ and $\mathbf{y}_i$ can be any two representations of the same instance d_i from two substantially different perspectives, not necessarily textual). This method is often used in combination with kernels, in which case it is known as *kernel canonical correlation analysis* (KCCA) (Hardoon, Szedmak, & Shawe-Taylor, 2004).

Miscellaneous

There are many other ways to extract or construct features from text, depending on the use that the features are intended for. For example, a *dual representation* of a corpus may be considered, in which features are used to represent terms and not documents. The feature vector for a term t contains one feature for each document, and its value is related to the frequency of t in that document. This representation can be used to analyze which words co-occur frequently and may therefore be related in meaning. Feature construction can also utilize methods from *information extraction*, such as identifying various kinds of named entities (names of persons, places, organizations, etc.) or other interesting bits of information and introducing features which indicate the presence of particular names or other tagged entities in the document.

Cross References

►Document Classification
►Feature Selection in Text Mining
►Kernel Methods
►Support Vector Machine
►Text Mining

Recommended Reading

Brank, J. (2006). Loose phrase string kernels. In *Proceedings of SiKDD, Ljubljana, Slovenia*. Jozef Stefan Institute.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, *41*, 391–407.

Hardoon, D. R., Szedmak, S. R., & Shawe-Taylor, J. R. (2004). Canonical correlation analysis: An overview with application to learning methods. *Neural Computation*, *16*(12), 2639–2664.

Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., & Watkins, C. (2002). Text classification using string kernels. *Journal of Machine Learning Research, 2*, 419–444.

Mladenić, D. (2002). Learning word normalization using word suffix and context from unlabeled data. *Proceedings of the 19th ICML 1*(8), 427–434.

Mladenić, D., & Grobelnik, M. (2003). Feature selection on hierarchy of web documents. *Decision Support Systems, 35*(1), 45–87.

Plisson, J., Lavrač, N., Mladenić, D., & Erjavec, T. (2008). Ripple down rule learning for automated word lemmatization. *AI Communications, 21*(1), 15–26.

Shawe-Taylor, J., & Cristianini, N. (2004). *Kernel methods for pattern analysis*. Cambridge: Cambridge University Press.

Feature Extraction

►Dimensionality Reduction

Feature Reduction

►Feature Selection

Feature Selection

Huan Liu
Arizona State University
Tempe, AZ, USA

Synonyms

Attribute selection; Feature reduction; Feature subset selection; Variable selection; Variable subset selection

Definition

Feature selection is the study of algorithms for reducing dimensionality of data to improve *machine learning* performance. For a dataset with N features and M dimensions (or features, attributes), feature selection aims to reduce M to M' and $M' \leq M$. It is an important and widely used approach to ►dimensionality reduction. Another effective approach is ►*feature extraction*. One of the key distinctions of the two approaches lies at their outcomes. Assuming we have four features F_1, F_2, F_3, F_4, if both approaches result in 2 features, the 2 *selected* features are a subset of 4 original features (say, F_1, F_3), but the 2 *extracted* features are some combination of 4 original features (e.g., $F'_1 = \sum a_i F_i$ and $F'_2 = \sum b_i F_i$, where a_i, b_i are some constants). Feature selection is commonly used in applications where original features need to be retained. Some examples are document categorization, medical diagnosis and prognosis, gene-expression profiling. We focus our discussion on feature selection. The benefits of feature selection are multifold: it helps improve machine learning in terms of predictive accuracy, comprehensibility, learning efficiency, compact models, and effective data collection.

The objective of feature selection is to remove irrelevant and/or redundant features and retain only relevant features. *Irrelevant features* can be removed without affecting learning performance. *Redundant features* are a type of irrelevant features. The distinction is that a redundant feature implies the copresence of another feature; individually, each feature is relevant, but the removal of either one will not affect learning performance.

Motivation and Background

The rapid advance of computer technology and the ubiquitous use of Internet have provided unparalleled opportunities for humans to expand the capabilities in production, services, communications, and research. In this process, immense quantities of high-dimensional data are accumulated, challenging the state-of-the-art machine learning techniques to efficiently produce useful results. Machine learning can benefit from using only relevant data in terms of learning performance (e.g., better predictive accuracy and shortened training time) and learning results such as improved comprehensibility to gain insights and to facilitate validation. At first glimpse, one might think a powerful machine learning algorithm can automatically identify the useful features in its model building process. In effect, removing irrelevant and/or redundant features can affect machine learning. First, let us look at what constitutes effective learning. In essence, the ►hypothesis space is largely constrained by the number of features. Learning can be viewed as searching for a "correct" hypothesis in the hypothesis space. A hypothesis is correct if it is consistent with the training data or the majority of it in the presence of noise, and is expected to perform equally well for the unseen data (or instances that are not present in the training data). In some sense, the more instances we have in the training data, the more constraints there are in helping guide the search for a correct hypothesis.

Broadly speaking, two factors matter most for effective learning: (1) the number of features (M), and (2) the number of instances (N). For a fixed M, a larger N means more constraints and the resulting correct hypothesis is expected to be more reliable. For a fixed N, a decreased M is tantamount to a significantly increased number of instances. Consider the following *thought experiment* for a binary domain of a binary classification problem: F_1, F_2, F_3, F_4 are binary and class C is also binary (e.g., positive or negative). If the training data consists of 4 instances ($N = 4$), it is only a quarter of the total number of possible instances ($2^4 = 16$). The size of the hypothesis space is $2^{2^4} = 65,536$. If only two features are relevant, the size of the hypothesis

space becomes $2^{2^2} = 16$, an exponential reduction of the hypothesis space. Now, the only available 4 instances might suffice for perfect learning if there is no duplicate instance in the reduced training data with two features. And a resulting model of 4 features can also be more complex than that of 2 features. Hence, feature selection can effectively reduce the hypothesis space, or virtually increase the number of training instances, and help create a compact model.

An unnecessarily complex model subjects itself to oversearching an excessively large hypothesis space. Its consequence is that the learned hypothesis *overfits* the training data and is expected to perform poorly when applying the learned model to the unseen data. Another way of describing the relationship between N and M in the context of learning is the so-called *curse of dimensionality*, the need for the *exponential* increase in data size associated with *linearly* adding additional dimensions to a multidimensional space; or the concept of proximity becomes blurry in a high-dimensional space, resulting in degrading learning performance. Theoretically, the reduction of dimensionality can eventuate the exponential shrinkage of hypothesis space.

Structure of the Learning System

The structure of a feature selection system consists of four basic components: input, search, evaluation, and output. The output of any feature selection system can be either a *ranked list* of features or a *subset* of features. For the former, one can select top k highly ranked features depending on the need. In the context of learning, the *input* to a feature selection system is the data which can be (1) supervised – all instances are associated with class labels as in supervised learning; (2) unsupervised – no class labels are available as in unsupervised learning; and (3) some instances have class labels and the rest do not as in semi-supervised learning. To rank the features or select a feature subset can be phrased as a *search* problem in which various search strategies can be employed. Depending on how a feature selection system is working together with a learning system, we can study different *models* of feature selection (Kohavi & John, 1997) such as wrapper, filter, or embedded. An inevitable question about feature selection is whether the removal of features can help machine learning. This necessitates the *evaluation* of feature selection. We will review these aspects of feature selection research next.

Categories of Feature Selection

Feature selection algorithms can be categorized into supervised, unsupervised, and semi-supervised, corresponding to different types of learning algorithms. There has been a substantial gamut of research on *supervised feature selection*. As in supervised machine learning, the data available for feature selection contains class labels. The class information is used as a dominant factor in determining the feature quality. For example, we can simply measure the correlation between a feature (or a subset of features) and the class and select those features with highest correlations. Another way of using the class information is to see if a feature can help differentiate two neighboring instances with different classes: obviously, a relevant feature can, but an irrelevant feature cannot.

Unsupervised feature selection has gained much attention in the recent years. Most data collected are without class labels since labeling data can incur huge costs. The basic principle of unsupervised learning is to cluster data such that similar objects (instances) are grouped together and dissimilar objects are separated. In other words, if we had all similar objects in their corresponding designated clusters, we would have the minimum intracluster distances and the maximum intercluster distances among all objects. For data of high-dimensionality, distance calculation can be a big problem due to the ►curse of dimensionality. One idea is to find features that can promote the data separability. In Dy and Brodley (2004), the goal of unsupervised feature selection is defined as finding the smallest feature subset that best uncovers "interesting natural" clusters from data according to the chosen criterion. A variant of unsupervised feature selection is subspace clustering. It explores the fact that in a high-dimensional space, clusters can often be found in various subspaces of very low dimensionality. Some subspace clustering algorithms are reviewed in Parson, Haque, and Liu (2004).

Unsupervised feature selection is a more loosely constrained problem than supervised feature selection. When a large number of labeled instances are infeasible to obtain, could we use a small number of labeled instances? *Semi-supervised feature selection* attempts to take advantage of both the size of unlabeled data and the labeling information of a small number of labeled instances. In the same spirit of ►semi-supervised learning, semi-supervised feature selection takes advantage of the two biases inherited in labeled

and unlabeled data, respectively, in the hope that the problem of feature selection becomes more constrained. The basic idea is to find features that can not only help group the data, but also encourage to find, among many equally good groups, those groups in which instances of different classes are not in the same group (Zhao & Liu, 2007b).

Searching for Relevant Features

The search for relevant features can be realized in two ways: (1) *feature ranking* – features are ranked according to the intrinsic properties of the data so that top k features can be chosen according to the need or a given threshold; and (2) *subset selection* – a subset of feature is selected from the full set of features, and there is no relevant difference between the features in the selected subset. Subset selection can be carried out in various ways: forward selection – starting with an empty feature subset and adding more iteratively, backward elimination – beginning with a full set of features and eliminating some gradually, and random – the starting subset can be any number which is then adjusted: if this number of features suffices according to some quality measure, it may be decreased, otherwise it should be increased. The exhaustive search is usually too expensive for either forward or backward search. Hence, a sequential search strategy is often adopted. Sequential forward search (SFS) selects one feature at a time; once a feature is selected, it will always be in the selected feature subset and also helps determine which feature should be selected next within the already selected features. Sequential backward search eliminates one feature at a time; once it is ruled out, it will never be considered for selection or inclusion in the selected set of features.

Here we briefly illustrate two effective and efficient algorithms with disparate ideas. One is ReliefF (Robnik-Sikonja & Kononenko, 2003). It selects a feature by checking how effectively it can differentiate the neighboring data points with different classes. A feature's weight is increased if it is effective in doing so. The features are then ranked according to their weights, and the top ranked features are deemed relevant. The second is FCBF (Yu & Liu, 2004), which adds the feature-feature correlation into the selection process with feature-class correlations. The basic idea is that for two features and the class, we can consider not only each feature's correlation with the class, but also the correlation between the two features. If the feature-feature correlation is greater than the smaller of the two feature-class correlations, the feature with smaller feature-class correlation is redundant and can thus be removed.

Models of Feature Selection

Three classic models of feature selection are filter, wrapper, and embedded. Research shows that, generally speaking, even for a classifier with embedded feature selection capability, it can benefit from feature selection in terms of learning performance. A filter model relies on measures about the intrinsic data properties. Mutual information and data consistency are two examples of measures about data properties. A wrapper model involves a learning algorithm (e.g., a classifier, or a clustering algorithm) in determining the feature quality. For instance, if removing a feature does not affect the classifier's accuracy, the feature can be removed. Obviously, this way feature selection is adapted to improving a particular classification algorithm. To determine if the feature should be selected or removed, it needs to build a classifier every time when a feature is considered. Hence, the wrapper model can be quite costly. An embedded model embeds feature selection in the learning of a classifier. A best example can be found in decision tree induction in which, at each branching point, a feature has to be selected first. When feature selection is performed for data preprocessing, filter and wrapper models are often employed. When the purpose of feature selection goes beyond improving learning performance (e.g., classification accuracy), the most applied is the filter model.

Evaluation of Feature Selection

The efficacy of feature selection can be validated via empirical evaluation. Two natural questions related to classification learning are (1) whether using selected features can do as well as using the full set of features, and (2) how to compare two feature selection algorithms when one wants to figure out which is more effective. The first question can be considered as a special form of the second one if we assume the full set of features is selected by a dummy feature selection algorithm that simply selects all given features. We therefore address

only the second question here. When we need to compare two feature selection algorithms (A_1, A_2), if the relevant features are known (the ground truth) as in the experiments using synthetic data, we can directly compare the selected features by A_1 and A_2, respectively, and check which result is closer to the ground truth. In practice, one seldom knows what the relevant features are. A conventional way of evaluating two algorithms is to evaluate the effect of selected features on classification accuracy. It is a two-step procedure: first, selecting features from data D to form D'_i with reduced dimensionality; and second, obtaining estimated predictive accuracy of a classifier on D and D'_i, respectively. Which algorithm is superior can be statistically measured by accuracy difference between A_1 and A_2. If there is no significant difference, one cannot tell which one of the two feature selection algorithms is better; otherwise, the algorithm resulting in better predictive accuracy is better.

Another issue arising from feature selection evaluation is about *feature selection bias*. Using the *same* training data in both feature selection and classification learning can result in this selection bias. According to statistical theory based on regression research, this bias can exacerbate data overfitting and negatively affect classification performance. A recommended practice is to use separate data for feature selection and for learning. In reality, however, separate datasets are rarely used in the selection and learning steps. This is because we often want to use as much data as possible in both selection and learning. It is against this intuition to divide the training data into two datasets leading to the reduced data in either task. The work presented in Singhi and Liu (2006) convincingly demonstrates that in regression, feature selection bias caused by using the same data for feature selection and classification learning does not negatively impact classification as expected.

Feature Selection Development and Applications

The advancement of feature selection research enables us to tackle new challenges. Feature interaction presents a challenge to feature selection. If we define relevance using correlation, a feature by itself might have little correlation with the target concept as in classification learning, but can be very relevant if it is combined with some other features, because the subset can be strongly correlated with the target concept. The unintentional removal of these features can eventuate poor learning performance. It is, in general, computationally intractable to handle feature interaction. In Zhao and Liu (2007a), it is shown that it is feasible to identify interacting features, in the case of using data consistency as a feature quality measure, by designing a special data structure for linear-time backward elimination in terms of M features and by employing an information-theoretic feature ranking heuristic. The authors also point out that the key challenge of employing the ranking heuristic is the *feature order problem* – a lowly ranked feature is more likely to be eliminated first.

Data fusion of multiple data sources presents another challenge for feature selection. Multiple data sources, each with its own features, need to be integrated in order to perform an inference task with the same objective optimally. Instead of selecting the most relevant features from each data source, one now needs to consider selecting *complementary features*. It is also very likely that performing conventional feature selection on the single aggregated dataset by combining all the data sources cannot accomplish the task. The problem of complementary feature selection seems related to that of finding interacting features. It will be worthwhile to examine how both research efforts can bootstrap each other in attacking the two recent challenges.

The recent developments in feature selection witness many new efforts on studying *causal relationships* among features (Guyon, Aliferis, & Elisseeff, 2007) to distinguish actual features and experimental artifacts; on *text feature selection* (Forman, 2007) that were widely employed in thwarting spam emails, automatic sorting of news articles, Web content management, and customer support; on *small data sample* problems that present challenges to reliable estimation of feature quality and detection of feature interactions, and on connecting feature selection and feature extraction. Both feature selection and extraction aim to reduce the dimensionality of the data by removing the nonessential (redundant or noisy) information, but the two areas have been researched largely independently. They can, however, indeed complement each other. On the one hand, feature extraction approaches, such as linear discriminant analysis, are effective for reducing data dimensionality, but suffer from the high computational complexity, especially for high-dimensional data; on the other hand, feature selection algorithms can

handle large-scale data and also lead to easy interpretation of the resulting learning model, but they may not select interacting features. The challenges of high-dimensional data suggest a need for the two to work together.

There is little work in the literature discussing about selecting structural features and sequential features. When data evolve, the variety of data types increases. Semi-structural or structural data now become increasingly common. Consequently, some features in these data may contain a structure (e.g., a hierarchy that defines the relationships between some atomic features). It can be commonly seen in the data with meta data. Clearly, extant feature selection algorithms have to evolve in order to handle structural feature selection. Another area that requires more research attention is the study of sequential features for data streams and for ►time series. They are very different from the types of data that are well studied. Data streams are massive, transient, and often from multiple sources. Time series data present their continuous temporal patterns.

Feature selection research has found its application in many fields where the presence of large (either row-wise or column-wise) volumes of data presents challenges to effective data analysis and processing. High-throughput technologies allow for parallel measurements of massive biological variables describing biological processes. The inordinate number of the biological measurements can contain noise, irrelevance, and redundancy. Feature selection can help focus on relevant biological variables in genomics and proteomics research. The pervasive use of Internet and Web technologies has been bringing about a great number of new services and applications, ranging from recent Web 2.0 applications to traditional Web services where multimedia data are ubiquitous and abundant. Feature selection is widely applied to find topical terms, establish group profiles, assist categorization, simplify descriptions, facilitate personalization and visualization, among others.

Cross References

►Classification
►Clustering
►Cross Validation
►Curse of Dimensionality
►Dimensionality Reduction
►Semi-Supervised Learning

Recommended Reading

Dy, J. G., & Brodley, C. E. (2004). Feature selection for unsupervised learning. *Journal of Machine Learning Research, 5*, 845–889.

Forman, G. (2007). Feature selection for text classification. In H. Liu & Motoda, H. (Eds.), *Computational methods of feature selection*. Boca Raton, FL: Chapman and Hall/CRC Press.

Guyon, I., Aliferis, C., & Elisseeff, A. (2007). Causal feature selection. In *(LM07)*, A longer technical report is available: http://clopinet.com/isabelle/Papers/causalFS.pdf.

Guyon, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research, 3*, 1157–1182.

Kohavi, R., & John, G. H. (1997). Wrappers for feature subset selection. *Artificial Intelligence, 97*(1–2), 273–324.

Liu, H., & Motoda, H. (1998). *Feature Selection for knowledge discovery & data mining*. Boston, MA: Kluwer Academic Publishers.

Liu, H., & Motoda, H. (Eds.). (2007). *Computational methods of feature selection*. Boca Raton, FL: Chapman and Hall/CRC Press.

Liu, H., & Yu, L. (2005). Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering, 17*(3), 1–12.

Parson, L., Haque, E., & Liu, H. (2004). Subspace clustering for high dimensional data – a review. *ACM SIGKDD Explorations Newsletter Archive special issue on learning from imbalanced datasets*, 6(1): 90–105. ISSN: 1931–0145

Robnik-Sikonja, M., & Kononenko, I. (2003). Theoretical and empirical analysis of Relief and ReliefF. *Machine Learning, 53*, 23–69.

Singhi, S., & Liu, H. (2006). Feature subset selection bias for classification learning. In *Proceeding of the 23rd international conference on machine learning ACM international conference proceeding series*, (Vol. 148, pp. 849–856). Pittsburg, PA. ISBN: 1-59593-383-2

Yu, L., & Liu, H. (2004). Efficient feature selection via analysis of relevance and redundancy. *Journal of Machine Learning Research, 5*(October), 1205–1224.

Zhao, Z., & Liu, H. (2007a). Searching for interacting features. In *Proceedings of the 20th international joint conference on artificial intelligence*, (pp. 1156–1161). Hydrabad, India.

Zhao, Z., & Liu, H. (2007b). Semi-supervised feature selection via spectral analysis. In *Proceedings of 24th international conference on machine learning* (1151–1157). Corvalis, OR. ISBN: 978-1-59593-793-3

Feature Selection in Text Mining

Dunja Mladenić
Jožef Stefan Insitute
Ljubljana, Slovenia

Synonyms

Dimensionality reduction on text via feature selection

Definition

The term *feature selection* is used in machine learning for the process of selecting a subset of features

(dimensions) used to represent the data (see ►Feature Selection, and ►Dimensionality Reduction). Feature selection can be seen as a part of data pre-processing potentially followed or coupled with feature construction ►Feature Construction in Text Mining, but can also be coupled with the learning phase if embedded in the learning algorithm. An Assumption of feature selection is that we have defined an original feature space that can be used to represent the data, and our goal is to reduce its dimensionality by selecting a subset of original features. The original feature space of the data is then mapped onto a new feature space. *Feature selection in text mining* is addressed here separately due to the specificity of textual data compared to the data commonly addressed in machine learning.

Motivation and Background

Tasks addressed in machine learning on text are often characterized by a high number of features used to represent the data. However, these features are not necessarily all relevant and beneficial for the task, and may slow down the applied methods giving similar results as a much smaller feature set. The main reasons for using feature selection in machine learning are Mladenić 2006: to improve performance, to improve learning efficiency, to provide faster models possibly requesting less information on the original data, and to reduce the complexity of the learned results and enable better understanding of the underlying process.

Feature selection in text mining was applied in a simple form from the start of applying machine learning methods on text data; for instance, feature selection by keeping the most frequent features and learning decision rules ►Rule Learning proposed in Apte, Damerau, and Weiss (1994) or keeping the most informative features for learning decision trees ►Decision Trees or ►Naïve Bayes ►Bayes Rule proposed in Lewis and Ringuette (1994). The reason is that the number of features used to represent text data for machine learning tasks is high, as the basic approach of learning on text defines a feature for each word that occurs in the given text. This can easily result in several tens of thousands of features, compared to several tens or hundreds of features, as commonly observed on most machine learning tasks at the time.

Most methods for feature subset selection that are used on text are very simple compared to the feature selection methods developed in machine learning. They perform a filtering of features assuming feature independence, so that a score is assigned to each feature independently and the features with high scores are selected. However, there are also more sophisticated methods for feature selection on text data that take into account interactions between the features. Embedded feature selection methods were successfully used on text data, either by applying a learning algorithm that has feature selection embedded (pre-processing step) or by inspecting a model generated by such an algorithm to extract feature scores. On the other hand, approaches to feature selection that search a space of all possible feature subsets can be rather time consuming when dealing with a high number of features, and are rarely used on text data.

Structure of Learning System

Feature selection in text mining is mainly used in connection with applying known machine learning and statistical methods on text when addressing tasks such as ►Document Clustering or ►Document Classification. This is also the focus of this chapter. However, we may need to perform some kind of feature selection on different text mining tasks where features are not necessary words or phrases, in which case we should reconsider the appropriate feature selection methods in the light of the task properties, including the number and type of features.

As already pointed out, the common way of document text representation is by defining a feature for each word in the document collection and feature selection by assuming feature independence, assigning score to the features, and selecting features with high scores. Scoring of individual features is performed either in an unsupervised way, ignoring the class information, or in a supervised way, taking into account class information. Surprisingly both kind of approaches have been shown to perform comparably on document classification tasks, even though supervised scoring uses more information. Here we discuss several feature scoring measures and their performance on document classification, as reported in different researcher papers.

One of the first scoring measures used on text data is scoring by the number of documents that contain a particular word. This was applied after removing very frequent words, as given in a standard "stop-list" for English. An alternative is scoring by frequency – that is, by the number of times a feature occurs in a document collection. Both were shown to work well in document classification Mladenić & Grobelnik (2003); Yang & Pedersen (1997).

Information gain is commonly used in decision tree induction (Quinlan, 1993). It was reported to work well as a feature scoring measure on text data (Yang & Pedersen, 1997) in some domains (news articles of in a collection named Reuters-22173, abstracts of medical articles in a subset of the MEDLINE collection), where a multiclass problem was addressed using the nearest neighbor algorithm ►Nearest Neighbor. The same feature scoring almost completely failed when using Naïve Bayes ►Bayes Rule on a binary classification problem on a hierarchical topic taxonomy of Web pages (Mladenić & Grobelnik, 2003). This difference in performance can be partially attributed to the classification algorithm and domain characteristics.

It is interesting to notice that information gain takes into account all values for each feature. In the case of document classification, these are two values: occurs or does not occur in a document. On the other hand, expected cross entropy as used on text data (Koller & Sahami, 1997; Mladenić & Grobelnik, 2003) is similar in nature to information gain, but only uses the situation when the feature occurred in a document. Experiments on classifying document into a hierarchical topic taxonomy (Mladenić & Grobelnik, 2003) have show that this significantly improves performance. Expected cross entropy is related to information gain as follows: InfGain(F) = CrossEntropyTxt(F) + CrossEntropyTxt(F), where F is a binary feature (usually representing a word's occurrence).

The odds ratio was reported to outperform many other measures (Mladenić & Grobelnik, 2003) in combination with Naïve Bayes, used for document classification on data with highly imbalanced class distribution. A characteristic of Naïve Bayes used for text classification is that, once the model has been generated, the classification is based on the features that occur in a document to be classified. This means that an empty document will be classified into the majority class. Consequently, having a highly imbalanced class distribution, if we want to identify documents from the under-represented class value, we need to have a model sensitive to the features that occur in such documents. If most of the selected features are representative for the majority class value, the documents from other classes will be almost empty when represented using the selected features.

Experimental comparison of different feature selection measures in combination with the support vector machines ►Support Vector Machines classification algorithm (SVM) on news articles from the Reuters-2000 collection (Brank, Grobelnik, Milič-Frayling, & Mladenić, 2002) has shown that using all or almost all the features yields the best performance. The same finding was confirmed in experimental evaluation of different feature selection measures on a number of text classification problems (Forman, 2003). In addition, in Forman (2003) a new feature selection measure was introduced: Bi-Normal Separation, which was reported to improve the performance of SVM, especially with problems where the class distribution is highly imbalanced. Interestingly, they also report that information gain is outperforming the other tested measures in the situation when using only a small number of selected features (20–50 features).

Another feature scoring measure for text data, called the Fisher Index, was proposed as part of a document retrieval system based on organizing large text databases into hierarchical topic taxonomies (Chakrabarti, Dom, Agrawal, & Raghavan, 1998). Similar to Mladenić (1998), for each internal node in the topic taxonomy, a separate feature subset is used to build a Naïve Bayes model for that node. This is sometimes referred to as local feature selection or, alternatively, context sensitive feature selection. The feature set used in each node is relatively small and tuned to the node context.

What follows are formulas of the described scoring measures as given in Mladenić and Grobelnik (2003).

$$\begin{aligned}\text{InfGain}(F) = {} & P(F)\sum_i P(C_i|F)\log(P(C_i|F)/P(C_i)) \\ & + P(\neg F)\sum_i P(C_i|\neg F) \\ & \times \log P(C_i|\neg F)/P(C_i))\end{aligned}$$

$$\begin{aligned}\text{CrossEntropyTxt}(F) = {} & P(F)\sum_i P(C_i|F) \\ & \log(P(C_i|F)/P(C_i))\end{aligned}$$

$$\text{MutualInfoTxt}(F) = \sum_i P(C_i)\log(P(F|C_i)/P(F))$$
$$\text{OddsRatio}(F) = \log(P(F|C_{\text{pos}})(1-P(F|C_{\text{neg}})))$$
$$-\log((1-P(F|C_{\text{pos}}))P(F|C_{\text{neg}}))$$
$$\text{Bi-NormalSeparation}(F) = Z^{-1}(P(F|C_{\text{pos}}))$$
$$-Z^{-1}(P(F|C_{\text{neg}}))$$
$$\text{FisherIndexTxt}(F) = (\sum_{\text{pos,neg}}(P(F|C_{\text{pos}})$$
$$-P(F|C_{\text{neg}}))^2)/\sum_{\text{Ci}\varepsilon\text{pos,neg}}|C_i|^{-1}$$
$$\times\sum_{\text{d}\varepsilon\text{Ci}}(n(F,d)$$
$$-P(F|C_i))2$$

where $P(F)$ is the probability that feature F occurred, $\neg F$ means that the feature does not occur, $P(C_i)$ is the probability of the ith class value, $P(C_i|F)$ is the conditional probability of the ith class value given that feature F occurred, $P(F|C_i)$ is the conditional probability of feature occurrence given the ith class value, $P(F|C_{\text{pos}})$ is the conditional probability of feature F occurring given the class value "positive," $P(F|C_{\text{neg}})$ is the conditional probability of feature F occurring given the class value "negative," $Z^{-1}(x)$ is the standard Normal distribution's inverse cumulative probability function (z-score), $|C_i|$ is the number of documents in class C_i, and $n(F, d)$ is 1 if the document d contains feature F and 0 otherwise.

As already highlighted in text classification, most of the feature selection methods evaluate each feature independently. A more sophisticated approach is proposed in Brank et al. (2002), where a linear SVM is first trained using all the features, and the induced model is then used to score the features (weight assigned to each feature in the normal to the induced hyper plane is used as a feature score). Experimental evaluation using that feature selection in combination with SVM, Perceptron, and Naïve Bayes has shown that the best performance is achieved by SVM when using almost all the features. The experiments have confirmed the previous findings on feature subset selection improving the performance of Naïve Bayes, but the overall performance is lower than using SVM on all the features.

Much the same as in Brank et al. (2002), feature selection was performed using a linear SVM to rank the features in Bi, Bennett, Embrechts, Breneman, and Song (2003). However, the experiments in Bi et al. (2003) were performed on a regression problem, and the final model was induced using a nonlinear SVM. The feature selection was shown to improve performance.

Distributional clustering of words with an agglomerative approach (words are viewed as distributions over document categories) is used for dimensionality reduction via feature construction (Bekkerman, El-Yaniv, Tishby, & Winter, 2003) that preserves the mutual information between the features as much as possible. This representation was shown to achieve comparable or better results than the bag-of-words document representation using feature selection based on Mutual information for text; a linear SVM was used as the classifier. A related approach, also based on preserving the mutual information between the features (Globerson & Tishby, 2003), finds new dimensions by using an iterative projection algorithm instead of clustering. It was shown to achieve performance comparable to the bag-of-words representation with all the original features, using significantly less features (e.g., on one dataset, four constructed features achieved 98% of performance of 500 original features) using the linear SVM classifier.

Divisive clustering for feature construction (Dhillon, Mallela, & Kumar, 2003) was shown to outperform distributional clustering when used for dimensionality reduction on text data. The approach uses the Kullback-Leibler divergence as a distance function, and minimizes within-cluster divergence while maximizing between-cluster divergence. Experiments on two datasets have shown that this dimensionality reduction slightly improves the performance of Naïve Bayes (compared to using all the original features), outperforming the agglomerative clustering of words combined with Naïve Bayes and achieving considerably higher classification accuracy for the same number of features than feature subset selection using information gain or mutual information (in combination with Naïve Bayes or SVM).

Recommended Reading

Apte, C., Damerau, F., & Weiss, S. M. (1994). Toward language independent automated learning of text categorization models. In *Proceedings of the 17th annual International ACM SIGIR conference on research and development in Information Retrieval*, pp. 23–30, Dublin, Ireland, 1994.

Brank, J., Grobelnik, M., Milič-Frayling, N., & Mladenić, D. (2002). Feature selection using support vector machines. In A. Zanasi (Ed.), *Data mining III* (pp. 261–273). Southampton, UK: WIT.

Bi, J., Bennett, K. P., Embrechts, M., Breneman, C. M., & Song, M. (2003). Dimensionality reduction via sparse support vector machines. *Journal of Machine Learning Research*, *3*, 1229–1243.

Bekkerman, R., El-Yaniv, R., Tishby, N., & Winter, Y. (2003). Distributional word clusters vs. words for text categorization. *Journal of Machine Learning Research*, *3*, 1183–1208.

Chakrabarti, S., Dom, B., Agrawal, R., & Raghavan, P. (1998). Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies. *The VLDB Journal*, *7*, 163–178.

Dhillon, I., Mallela, S., & Kumar, R. (2003). A divisive information-theoretic feature clustering algorithm for text classification. *Journal of Machine Learning Research*, *3*, 1265–1287.

Forman, G. (2003). An extensive empirical study of feature selection metrics for text classification. *Journal of Machine Learning Research*, *3*, 1289–1305.

Globerson, A., & Tishby, N. (2003). Sufficient dimensionality reduction. *Journal of Machine Learning Research*, *3*, 1307–1331.

Koller, D., & Sahami, M. (1997). Hierarchically classifying documents using very few words. In *Proceedings of the 14th international conference on machine learning ICML'97* (pp. 170–178). Nashrille, TN.

Lewis, D. D., & Ringuette, M. (1994). Comparison of two learning algorithms for text categorization. In *Proceedings of the 3rd annual symposium on document analysis and information retrieval SDAIR-1994*. Las Vegas, NV.

Mladenić, D. (1998). Feature subset selection in text-learning. In *Proceedings of the 10th European conference on machine learning ECML'98*. Chemnitz, Germany.

Mladenić, D. (2006). Feature selection for dimensionality reduction. In C. Saunders, S. Gunn, J. Shawe-Taylor, & M. Grobelink (Eds.), *Subspace, Latent Structure and Feature Selection: Statistical and Optimization Perspectives Workshop: Lecture notes in computer science* (Vol. 3940, pp. 84–102). Berlin, Heidelberg: Springer.

Mladenić, D., & Grobelnik, M. (2003). Feature selection on hierarchy of web documents. *Journal of Decision Support Systems*, *35*, 45–87.

Quinlan, J. R. (1993). Constructing decision tree. In *C4.5: Programs for machine learning*. San Francisco: Morgan Kaufman Publishers.

Yang, Y., & Pedersen, J. O. (1997). A comparative study on feature selection in text categorization. In *Proceedings of the 14th international conference on machine learning ICML'97* (pp. 412–420). Las Vegas, NV.

Feature Subset Selection

►Feature Selection

Feedforward Recurrent Network

►Simple Recurrent Network

Finite Mixture Model

►Mixture Model

First-Order Logic

Peter A. Flach
University of Bristol
Bristol, UK

Synonyms

First-order predicate calculus; First-order predicate logic; Predicate calculus; Predicate logic; Resolution

Definition

First-order predicate logic – first-order logic for short – is the logic of properties of, and relations between, objects and their parts. Like any logic, it consists of three parts: *syntax* governs the formation of *well-formed formulae*, *semantics* ascribes meaning to well-formed formulae and formalizes the notion of *deductive consequence*, and *proof procedures* allow the inference of deductive consequences by syntactic means. A number of variants of first-order logic exist, mainly differing in their syntax and proof systems. In machine learning, the main use of first-order logic is in ►learning from structured data, ►inductive logic programming and ►relational data mining.

Motivation and Background

The interest in logic arises from a desire to formalize human, mathematical and scientific reasoning, and goes back to at least the Greek philosophers. Aristotle devised a form of propositional reasoning called *syllogisms* in the fourth century BC. Aristotle was held in very high esteem by medieval scholars, and so further significant advances were not made until after the Middle Ages. Leibniz wrote of an "algebra of thought"

and linked reasoning to calculation in the late seventeenth century. Boole and De Morgan developed the algebraic point of view in the mid-nineteenth century.

Universally quantified variables, which form the main innovation in first-order logic as compared to ▶propositional logic, were invented by Gottlob Frege in his *Begriffsschrift* ("concept notation") from 1879, and independently by Charles Sanders Peirce in 1885, who introduced the notation $\prod_x$ and $\sum_x$ for universal and existential quantification. Frege's work went largely unnoticed until it was developed further by Alfred North Whitehead and Bertrand Russell in their *Principia Mathematica* (1903). Seminal contributions were made, among many others: by Giuseppe Peano, who axiomatized number theory and introduced the notation (x) and $\exists x$; by Kurt Gödel, who established the completeness of first-order logic as well as the incompleteness of any system incorporating Peano arithmetic; by Alonzo Church, who proved that first-order logic is undecidable, and who introduced λ-calculus, a form of ▶higher-order logic that allows quantification over predicates and functions (as opposed to first-order logic, which only allows quantification over objects); and by Alfred Tarski, who pioneered logical semantics through model theory, and the notion of logical consequence. The now universally accepted notation $\forall x$ was introduced by Gerhard Gentzen.

Logic plays an important role in any approach to symbolic AI that employs a formal language for knowledge representation and inference. A significant, relatively recent development was the introduction of logic programming languages such as ▶Prolog, which turn logical inference into computation. In machine learning, the use of a first-order language is essential in order to handle domains in which objects have inherent structure; the availability of Prolog as a common language and programming platform gave rise to the field of inductive logic programming.

Theory

Syntax

A first-order logical language is built from *constant symbols*, *variable symbols*, *predicate symbols* and *function symbols*; the latter two kinds of symbols have an associated *arity*, which is the number of arguments they take. *Terms* are either constant symbols, variable symbols, or of the form $f(t_1, \ldots, t_n)$ where f is a function symbol with arity n, and $t_1, \ldots, t_n$ is a sequence of n terms. Using the logical connectives $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction) and $\rightarrow$ (material implication) and the quantifiers $\forall$ (universal quantifier) and $\exists$ (existential quantifier), well-formed formulae or *wffs* are defined recursively as follows: (1) if P is a predicate symbol with arity n, and $t_1, \ldots, t_n$ is a sequence of n terms, then $P(t_1, \ldots, t_n)$ is a wff, also referred to as an atomic formula or *atom*; (2) if ϕ_1 and ϕ_2 are wff's, then $(\neg\phi_1)$, $(\phi_1 \wedge \phi_2)$, $(\phi_1 \vee \phi_2)$ and $(\phi_1 \rightarrow \phi_2)$ are wffs; (3) if x is a variable and ϕ is a wff, then $(\forall x : \phi)$ and $(\exists x : \phi)$ are wffs; (4) nothing else is a wff. Brackets are usually dropped as much as it is possible without causing confusion.

Example 1 *Let "man," "single," and "partner" be two unary and one binary predicate symbol, respectively, and let "x" and "y" be variable symbols, then the following is a wff ϕ expressing that men who are not single have a partner:*

$$(\forall x : (man(x) \wedge (\neg single(x))) \rightarrow (\exists y : partner(x, y))).$$

Assuming that $\neg$ binds strongest, then $\wedge$, then $\rightarrow$, the brackets can be dropped:

$$\forall x : man(x) \wedge \neg single(x) \rightarrow \exists y : partner(x, y).$$

A *propositional language* is a special case of a predicate-logical language, built only from predicate symbols with arity 0, referred to as *proposition symbols* or propositional atoms, and connectives. So, for instance, assuming the proposition symbols "*man*," "*single*" and "*has_partner*," the following is a propositional wff: $man \wedge \neg single \rightarrow has\_partner$. The main difference is that in propositional logic references to objects cannot be expressed and therefore have to be understood implicitly.

Semantics

First-order wffs express statements that can be true or false and so a first-order semantics consists in constructing a mapping from wffs to truth-values, given an interpretation, which is a possible state of affairs in the domain of discourse, mapping constant, predicate and

function symbols to elements, relations and functions in and over the domain. To deal with variables, a valuation function is employed. Once this mapping is defined, the meaning of a wff consists in the set of interpretations in which the wff maps to true, also called its models. The intuition is that the more "knowledge" a wff contains, the fewer models it has. The key notion of logical consequence is then defined in terms of models: one wff is a logical consequence of another if the set of models of the first contains the set of models of the second; hence the second wff contains at least the same, if not more, knowledge than the first.

Formally, a *predicate-logical interpretation*, or *interpretation* for short, is a pair (D, i), where D is a non-empty *domain* of individuals, and i is a function assigning to every constant symbol an element of D, to every function symbol with arity n a mapping from D^n to D, and to every predicate symbol with arity n a subset of D^n, called the *extension* of the predicate. A *valuation* is a function v assigning to every variable symbol an element of D.

Given an interpretation $I = (D, i)$ and a valuation v, a mapping i_v from terms to individuals is defined as follows: (1) if t is a constant symbol, $i_v(t) = i(t)$; (2) if t is a variable symbol, $i_v(t) = v(t)$; (3) if t is a term $f(t_1, \ldots, t_n)$, $i_v(t) = i(f)(i_v(t_1), \ldots, i_v(t_n))$. The mapping is extended to a mapping from wffs to truthvalues as follows: (4) if ϕ is an atom $P(t_1, \ldots, t_n)$, $i_v(\phi) = i(P)(i_v(t_1), \ldots, i_v(t_n))$; (5) $i_v(\neg\phi) = T$ if $i_v(\phi) = F$, and F otherwise; (6) $i_v(\phi_1 \wedge \phi_2) = T$ if $i_v(\phi_1) = T$ and $i_v(\phi_2) = T$, and F otherwise; (7) $i_v(\forall x : \phi) = T$ if $i_{v_{x \to d}}(\phi) = T$ for all $d \in D$, and F otherwise, where $v_{x \to d}$ is v except that x is assigned d. The remaining connectives and quantifier are evaluated by rewriting: (8) $i_v(\phi_1 \vee \phi_2) = i_v(\neg(\neg\phi_1 \wedge \neg\phi_2))$; (9) $i_v(\phi_1 \to \phi_2) = i_v(\neg\phi_1 \vee \phi_2)$; (10) $i_v(\exists x : \phi) = i_v(\neg\forall x : \neg\phi)$.

An interpretation I *satisfies* a wff ϕ, notation $I \models \phi$, if $i_v(\phi) = T$ for all valuations v; we say that I is a *model* of ϕ, and that ϕ is *satisfiable*. If all models of a set of wffs Σ are also models of ϕ, we say that Σ *logically entails* ϕ or ϕ is a *logical consequence* of Σ, and write $\Sigma \models \phi$. If $\Sigma = \varnothing$, ϕ is called a *tautology* and we write $\models \phi$. A wff ψ is a *contradiction* if $\neg\psi$ is a tautology. Contradictions do not have any models, and consequently $\psi \models \alpha$ for any wff α. The *deduction theorem* says that $\Sigma \models \alpha \to \beta$ if and only if $\Sigma \cup \{\alpha\} \models \beta$. Another useful fact is that, if $\Sigma \cup \{\neg\gamma\}$ is a contradiction, $\Sigma \models \gamma$; this gives rise to a proof technique known as *Reductio ad absurdum* or *proof by contradiction* (see below).

Example 2 *We continue the previous example. Let* $D = \{Peter, Paul, Mary\}$*, and let the function* i *be defined as follows:* $i(man) = \{Peter, Paul\}$*;* $i(single) = \{Paul\}$*;* $i(partner) = \{(Peter, Mary)\}$*. We then have that the interpretation* $I = (D, i)$ *is a model for the wff* ϕ *above. On the other hand,* I *doesn't satisfy* $\psi = \forall x : \exists y : partner(x, y)$*, and therefore* $\phi \not\models \psi$*. However, the reverse does hold: there is no interpretation that satisfies* ψ *and not* ϕ*, and therefore* $\psi \models \phi$*.*

In case of a propositional logic this semantics can be considerably simplified. Since there are no terms the domain D plays no role, and an interpretation simply assigns truth-values to proposition symbols. Wffs can then be evaluated using rules (5–6) and (8–9). For example, if $i(man) = T$, $i(single) = T$ and $i(has\_partner) = T$, then $i(man \wedge \neg single \to has\_partner) = T$ (if this seems counter-intuitive, this is probably because the reader's knowledge of the domain suggests another wff $\neg(single \wedge has\_partner)$, which is false in this particular interpretation).

Proofs

A *proof procedure* consists of a set of *axioms* and a set of *inference rules*. Given a proof procedure P, we say that ϕ is *provable* from Σ and write $\Sigma \vdash_P \phi$ if there exists a finite sequence of wffs $\phi_1, \phi_2, \ldots, \phi_{n-1}, \phi$ which is obtained by successive applications of inference rules to axioms, *premisses* in Σ, and/or previous wffs in the sequence. Such a sequence of wffs, if it exists, is called a *proof* of ϕ from Σ. A proof procedure P is *sound*, with respect to the semantics established by predicate-logical interpretations, if $\Sigma \models \phi$ whenever $\Sigma \vdash_P \phi$; it is *complete* if $\Sigma \vdash_P \phi$ whenever $\Sigma \models \phi$. For a sound and complete proof procedure for first-order predicate logic, see e.g., Turner, 1984, p. 15.

A set of wffs Σ is *consistent*, with respect to a proof procedure P, if not both $\Sigma \vdash_P \phi$ and $\Sigma \vdash_P \neg\phi$ for some wff ϕ. Given a sound and complete proof procedure, the proof-theoretic notion of consistency coincides with the semantic notion of satisfiability. In particular, if we can prove that $\Sigma \cup \{\neg\gamma\}$ is inconsistent, then we know that $\Sigma \cup \{\neg\gamma\}$ is not satisfiable, hence a contradiction, and thus $\Sigma \models \gamma$. This still holds if the proof procedure

is only complete in the weaker sense of being able to demonstrate the inconsistency of arbitrary sets of wffs (see the resolution inference rule, below).

Example 3 *One useful inference rule for predicate logic replaces a universally quantified variable with an arbitrary term, which is called* universal elimination. *So, if "c" is a constant symbol in our language, then we can infer*

$$man(c) \wedge \neg single(c) \rightarrow \exists y : partner(c, y)$$

from ϕ above by universal elimination. Another inference rule, which was called Modus Ponens *by Aristotle, allows us to infer β from α and $\alpha \rightarrow \beta$. So, if we additionally have $man(c) \wedge \neg single(c)$, then we can conclude*

$$\exists y : partner(c, y)$$

by Modus Ponens. This rule is also applicable to propositional logic. An example of an axiom is $c = c$ for any constant symbol c (strictly speaking this is an axiom schema, giving rise to an axiom for every constant symbol in the language).

Programming in Logic

Syntax, semantics and proof procedures for first-order logic can be simplified and made more amenable to computation if we limit the number of ways of expressing the same thing. This can be achieved by restricting wffs to a normal form called *prenex conjunctive normal form* (PCNF). This means that all quantifiers occur at the start of the wff and are followed by a conjunction of disjunctions of atoms and negated atoms, jointly called *literals*. An example of a formula in PCNF is

$$\forall x : \exists y : \neg man(x) \vee single(x) \vee partner(x, y).$$

This formula is equivalent to the wff ϕ in Example 1, in the sense that it has the same set of models, and so either one logically entails the other. Every first-order wff can be transformed into a logically equivalent formula in PCNF, which is unique up to the order of conjuncts and disjuncts. A transformation procedure can be found in Flach (1994).

PCNF can be further simplified if we use function symbols instead of existential quantifiers. For instance, instead of $\exists y : partner(x, y)$, we can say $partner(x, partner\_of(x))$, where *partner_of* is a unary function symbol called a *Skolem function*, after the Norwegian logician Thoralf Skolem. The two statements are not logically equivalent, as the second entails the first but not vice versa, but this difference is of little practical consequence. Since all variables are now universally quantified the quantifiers are usually omitted, leading to *clausal form*:

$$\neg man(x) \vee single(x) \vee partner(x, partner\_of(x)).$$

To sum up, a wff in clausal form is a conjunction of disjunctions of literals, of which the variables are implicitly universally quantified. The individual disjunctions are called clauses.

Further simplifications include dispensing with equality, which means that terms involving function symbols, such as $partner\_of(c)$, are not evaluated and in effect treated as names of objects (in this case, the function symbols are called *functors* or *data constructors*). Under this assumption each *ground term* (a term without variables) denotes a different object, which means that we can take the set of ground terms as the domain D of an interpretation; this is called a *Herbrand interpretation*, after the French logician Jacques Herbrand.

The main advantage of clausal logic is the existence of a proof procedure consisting of a single inference rule and no axioms. This inference rule, which is called *resolution*, was introduced by Robinson (1965). In propositional logic, given two clauses $P \vee Q$ and $\neg Q \vee R$ containing *complementary* literals Q and $\neg Q$, resolution infers the *resolvent* $P \vee R$ (P and/or R may themselves contain several disjuncts). For instance, given $\neg man \vee single \vee has\_partner$ and $man \vee woman$, we can infer $woman \vee single \vee has\_partner$ by resolution. In first-order logic, Q and $\neg Q'$ are complementary if Q and Q' are *unifiable*, i.e., there exists a *substitution* θ of terms for variables such that $Q\theta = Q'\theta$, where $Q\theta$ denotes the application of substitution θ to Q; in this case, the resolvent of $P \vee Q$ and $\neg Q' \vee R$ is $P\theta \vee R\theta$. For instance, from the following two clauses:

$$\neg man(x) \vee single(x) \vee partner(x, partner\_of(x))$$

$$\neg single(father\_of(c))$$

we can infer

$$\neg man(father\_of(c)) \vee partner(father\_of(c), partner\_of(father\_of(c))).$$

The resolution inference rule is sound but not complete: for instance, it is unable to produce tautologies such as $man(c) \vee \neg man(c)$ if no clauses involving the predicate *man* are given. However, it is *refutation-complete*, which means it can demonstrate the unsatisfiability of any set of clauses by deriving the *empty clause*, indicated by $\square$. For instance, $man(c) \wedge \neg man(c)$ is a wff consisting of two clauses which are complementary literals, so by resolution we infer the empty clause in one step.

Refutation by resolution is the way in which queries are answered in the logic programming language Prolog. Prolog works with a subset of clausal logic called *Horn logic*, named after the logician Alfred Horn. A *Horn clause* is a disjunction of literals with at most one positive (un-negated) literal; Horn clauses can be further divided into *definite clauses*, which have one positive literal, and *goal clauses* which have none A Prolog program consists of definite clauses, and a goal clause functions as a procedure call. Notice that resolving a goal clause with a definite clause result in another goal clause, because the positive literal in the definite clause (also called its *head*) must be one of the complementary literals. The idea is that the resolution step reformulates the original goal into a new goal that is one step closer to the solution. A refutation is then a sequence of goals $G, G_1, G_2, \ldots, G_n$ such that G is the original goal, each G_i is obtained by resolving G_{i-1} with a clause from the program P, and $G_n = \square$. Such a refutation demonstrates that $P \cup \{G\}$ is inconsistent, and therefore $P \models \neg G$.

Finding a refutation amounts to a search problem, because there are typically several program clauses that could be resolved against the current goal. Virtually all Prolog interpreters apply a depth-first search procedure, searching the goal literals left-to-right and the program clauses top-down. Once a refutation is found the substitutions collected in all resolution steps are composed to obtain an *answer substitution*. One unique feature of logic programming is that a goal may have more than one (or, indeed, less than one) refutation and answer substitution from a given program.

Example 4 *Consider the following Prolog program:*

```
peano_sum(0,Y,Y).
peano_sum(s(X),Y,s(Z)):-
    peano_sum(X,Y,Z).
```

This program defines addition in Peano arithmetic. We follow Prolog syntax: variables start with an uppercase letter, and `:-` *stands for reversed implication* $\leftarrow$ *or "if." The unary functor* `s` *represents the successor function. So the first rule reads "the sum of 0 and an arbitrary number y is y," and the second rule reads "the sum of* $x + 1$ *and* y *is* $z + 1$ *if the sum of* x *and* y *is* z*."*

The goal `:-peano_sum(s(0),s(s(0)),Q)` *states "there are no numbers q such that* $1 + 2 = q$*." We first resolve this goal with the second program clause to obtain* `:-peano_sum(0,s(s(0)),Z)` *under the substitution* {`Q/s(Z)`}*. This new goal states "there are no numbers z such that* $0 + 2 = z$*." It is resolved with the first clause to yield the empty clause under the substitution* {`Y/s(s(0))`, `Z/s(s(0))`}*. The resulting answer substitution is* {`Q/s(s(s(0)))`}*, i.e.,* $q = 3$*.*

As another example, goal `:-peano_sum(A,B,s(s(0)))` *states "there are no numbers a and b such that* $a + b = 2$*." This goal has three refutations: one involving the first clause only, yielding the answer substitution* {`A/0`, `B/s(s(0))`}*; one involving the second clause then the first, resulting in* {`A/s(0)`, `B/s(0)`}*; and the third applying the second clause twice followed by the first, yielding* {`A/s(s(0))`, `B/0`}*. Prolog will return these three answers in this order.*

Induction in first-order logic amount to reconstructing a logical theory from some of its logical consequences. For techniques to induce a Prolog program given examples such as `peano_sum(s(0),s(0),s(s(0)))`, see inductive logic programming.

For general introductions to logic and its use in Artificial Intelligence, see Genesereth and Nilsson (1987) and Turner (1984). Kowalski's classic text *Logic for problem solving* focusses on clausal logic and resolution theorem proving (Kowalski, 1979). For introductions to Prolog programming, see Bratko (2001) and Flach (1994).

Cross References

►Abduction
►Entailment
►Higher-Order Logic
►Hypothesis Language
►Inductive Logic Programming
►Learning from Structured Data
►Logic Program
►Propositionalization
►Relational Data Mining

Recommended Reading

Bratko, I. (2001). *Prolog programming for artificial intelligence* (3rd ed.). Boston: Addison Wesley.
Flach, P. (1994). *Simply logical: Intelligent reasoning by example*. New York: Wiley.
Genesereth, M., & Nilsson, N. (1987). *Logical foundations of artificial intelligence*. San Francisco: Morgan Kaufmann.
Kowalski, R. (1979). *Logic for problem solving*. New York: North-Holland.
Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *Journal of the ACM, 12*(1), 23–41.
Turner, R. (1984). *Logics for artificial intelligence*. Chichester: Ellis Horwood.

First-Order Predicate Calculus

►First-Order Logic

First-Order Predicate Logic

►First-Order Logic

First-Order Regression Tree

Synonyms

Logical regression tree; Relational regression tree

Definition

A first-order regression tree can be defined as follows:

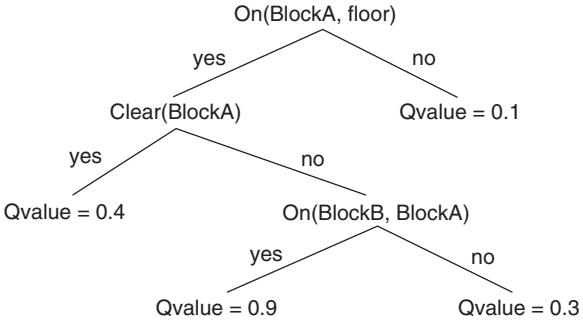
Definition 1 (First-Order Regression Tree) *A first-order regression tree is a binary tree in which*

- *Every internal node contains a test which is a conjunction of first-order literals.*
- *Every leaf (terminal node) of the tree contains a real valued prediction.*

An extra constraint placed on the first-order literals that are used as tests in internal nodes is that a variable that is introduced in a node (i.e., it does not occur in higher nodes) does not occur in the right subtree ofl the node.

Figure 1 gives an example of a first-order regression tree. The test in a node should be read as the existentially quantified conjunction of all literals in the nodes in the path from the root of the tree to that node. In the left subtree of a node, the test of the node is added to the conjunction, for the right subtree, the negation of the test should be added. For the example state description of Fig. 2, the tree would predict a *Qvalue* = 0.9, since there exists no block that is both on the floor and clear, but there is a block which is on the floor and has another block on top of it. To see this, substitute BlockA in the tree with 2 (or 4) and BlockB with 1 (or 4).

The constraint on the use of variables stems from the fact that variables in the tests of internal nodes are existentially quantified. Suppose a node introduces a new variable X. Where the left subtree of a node corresponds to the fact that a substitution for X has been found to make the conjunction true, the right side corresponds to the situation where no substitution for X exists, i.e.,



First-Order Regression Tree. Figure 1. A relational regression tree

```
on(1,2).        clear(1).
on(2,floor).    clear(3).
on(3,4).        clear(floor).
on(4,floor).
```

First-Order Regression Tree. Figure 2. State description

there is no such X. Therefore, it makes no sense to refer to X in the right subtree.

Cross References

►First-Order Rule
►Inductive Logic Programming
►Relational Reinforcement Learning

F-Measure

A measure of information retrieval performance. See ►Precision and Recall.

Foil

►Rule Learning

Formal Concept Analysis

Gemma C. Garriga
Universite Pierre et Marie Curie
Paris, France

Definition

Formal concept analysis is a mathematical theory of concept hierarchies that builds on order theory; it can be seen as an unsupervised machine learning technique and is typically used as a method of knowledge representation. The approach takes an input binary relation (binary matrix) specifying a set of objects (rows) and a set of attributes for those objects (columns), finds the natural concepts described in the data, and then organizes the concepts in a partial order structure or Hasse diagram. Each concept in the final diagram is a pair of sets of objects and attributes that are maximally contained one in each other.

Theory

The above intuition can be formalized through a Galois connection as follows. Let R be the binary relation between a set of objects and a set of attributes, that is, $R \subseteq \mathcal{O} \times \mathcal{A}$. Two mappings $\alpha : \mathcal{O} \mapsto \mathcal{A}$ and $\beta : \mathcal{A} \mapsto \mathcal{O}$ are defined so that the operator $\alpha(O)$, for some $O \subseteq \mathcal{O}$, returns the maximal set of attributes common to all objects in O; dually, the operator $\beta(A)$, for some $A \subseteq \mathcal{A}$, returns the maximal set of objects containing all attributes in A. These two mappings induce a Galois connection between the powerset of objects and the powerset of attributes, that is, they satisfy $O \subseteq \beta(A) \Leftrightarrow A \subseteq \alpha(O)$ for a set of objects O and a set of attributes A.

From here, a formal concept is a pair of sets of objects and attributes (O, A) from the binary relation that satisfy $\alpha(O) = A$ and $\beta(A) = O$. Typically, O is called the extent of the concept and A the intent of the concept. Note that concepts can be interpreted from the geometrical point of view, they are maximal rectangles of ones (not necessarily consecutive) in the input binary table R. The organization of all the formal concepts in a Hasse diagram is called the concept lattice. This lattice corresponds to a partial order structure of concepts where edges between concepts correspond to the standard inclusion of the sets.

A small toy example in Figs. 1 and 2 illustrates the formal concepts and their organization in a Hasse diagram.

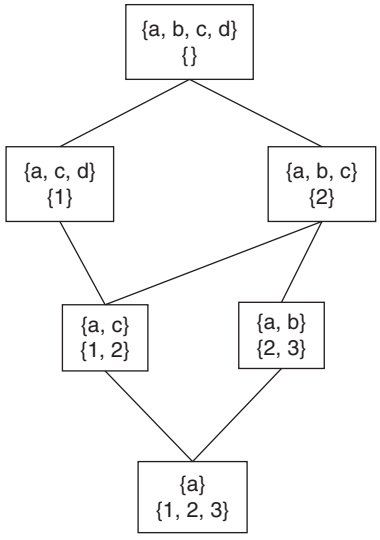
Motivation and Background

Formal concept analysis has been applied to a variety of disciplines, from psychology, sociology, biology, medicine, linguistics, or industrial engineering, to cite some, for the interactive exploration of implicit and explicit structures in the data.

From the point of view of machine learning and data mining, the connection between the formal concepts of the lattice and the so-called, closed sets of items is remarkable. Closed sets of items appear in the context of ►constraint-based mining, in which the user provides

	a	b	c	d
1	1	0	1	1
2	1	1	1	0
3	1	1	0	0

Formal Concept Analysis. Figure 1. A binary relation $R \subseteq \{1, 2, 3\} \times \{a, b, c, d\}$



Formal Concept Analysis. Figure 2. Concepts of the relation *R* organized in a Hasse diagram

restraints that guide a search of patterns in the data. They are maximal sets of attributes occuring frequently in the data; they correspond to a compacted representation of the frequent sets from ►frequent itemset mining. It is well known that closed sets correspond exactly to the intents of the concepts derived via formal concept analysis, and therefore, from the formal concepts it is possible to construct bases of minimal nonredundant sets of association rules from which all other rules holding in the data can be derived.

Also, formal concept analysis has been typically seen as a type of conceptual ►clustering. Each concept or groups of concepts form a cluster of objects sharing similar properties. The diagrams obtained from this sort of clustering can then be used in class discovery and class prediction. Although a diagram of concepts can become large and complex, different approaches have worked toward reducing the complexity of concept lattices via conceptual scaling.

We refer the reader to Ganter & Wille (1998) for a general reference on formal concept analysis, and to Davey & Priestly (2002) for the basic concepts on order theory. For more thorough descriptions of different applications of formal concept analysis in the computer science field, see Carpineto & Romano (2004).

Cross References

►Clustering
►Constraint-Based Mining
►Frequent Itemset Mining

Recommended Reading

Carpineto, C., & Romano, G. (2004). *Concept data analysis. Theory and applications*. New York: Wiley.
Davey, B. A., & Priestly, H. A. (2002). *Introduction to lattices and order*. Cambridge: Cambridge University Press.
Ganter, B. & Wille, R. (1998). *Formal concept analysis. Mathematical foundations*. Heidelberg: Springer.

Frequent Itemset

Hannu Toivonen
University of Helsinki
Helsinki, Finland

Synonyms

Frequent set

Definition

Frequent itemsets (Agrawal et al., 1993, 1996) are a form of ►frequent pattern. Given examples that are sets of items and a minimum frequency, any set of items that occurs at least in the minimum number of examples is a frequent itemset.

For instance, customers of an on-line bookstore could be considered examples, each represented by the set of books he or she has purchased. A set of books, such as {"*Machine Learning*," "*The Elements of Statistical Learning*," "*Pattern Classification*,"} is a frequent itemset if it has been bought by sufficiently many customers. Given a frequency threshold, perhaps only 0.1 or 0.01% for an on-line store, *all* sets of books that have been bought by at least that many customers are called frequent. Discovery of all frequent itemsets is a typical data mining task. The original use has been as part of ►association rule discovery. ►Apriori is a classical algorithm for finding frequent itemsets.

The idea generalizes far beyond examples consisting of sets. The pattern class can be re-defined, e.g., to be (frequent) subsequences rather than itemsets; or original data can often be transformed to a suitable representation, e.g., by considering each discrete attribute-value pair or an interval of a continuous attribute as an individual item. In such more general settings, the term ►frequent pattern is often used. Another direction to generalize frequent itemsets is to consider other conditions than frequency on the patterns to be discovered; see ►constraint-based mining for more details.

Cross References

►Apriori Algorithm
►Association Rule
►Constraint-Based Mining
►Frequent Pattern

Recommended Reading

Agrawal, R., Imieliński, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD international conference on management of data, Washington, DC* (pp. 207–216). New York: ACM.

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., & Verkamo, A. I. (1996). Fast discovery of association rules. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, & R. Uthurusamy (Eds.), *Advances in knowledge discovery and data mining* (pp. 307–328). Menlo Park: AAAI Press.

Frequent Pattern

Hannu Toivonen
University of Helsinki
Finland

Definition

Given a set $\mathcal{D}$ of examples, a language $\mathcal{L}$ of possible patterns, and a minimum frequency *min_fr*, every pattern $\theta \in \mathcal{L}$ that occurs at least in the minimum number of examples, i.e., $|\{e \in \mathcal{D} \mid \theta \text{ occurs in } e\}| \geq min\_fr$, is a frequent pattern. Discovery of all frequent patterns is a common data mining task. In its most typical form, the patterns are ►frequent itemsets. A more general formulation of the problem is ►constraint-based mining.

Motivation and Background

Frequent patterns can be used to characterize a given set of examples: they are the most typical feature combinations in the data.

Frequent patterns are often used as components in larger data mining or machine learning tasks. In particular, discovery of ►frequent itemsets was actually first introduced as an intermediate step in ►association rule mining (Agrawal, Imieliński & Swami, 1993) ("frequent itemsets" were then called "large"). The frequency and confidence of every valid association rule $X \rightarrow Y$ are obtained simply as the frequency of $X \cup Y$ and the ratio of frequencies of $X \cup Y$ and X, respectively.

Frequent patterns can be useful as ►features for further learning tasks. They may capture shared properties of examples better than individual original features, while the frequency threshold gives some guarantee that the constructed features are not so likely just noise. However, other criteria besides frequency are often used to choose a good set of candidate patterns.

Structure of Problem

A frequent pattern often is essentially a set of binary ►features. Given a set $\mathcal{I}$ of all available features, the pattern language $\mathcal{L}$ then is the power set of $\mathcal{I}$. An example in data $\mathcal{D}$ covers a pattern $\theta \in \mathcal{L}$ if it has all the features of θ. In such cases, the frequent pattern discovery task reduces to the task of discovering ►frequent itemsets. Therefore, the structure of the frequent pattern discovery problem is best described using the elementary case of frequent itemsets.

Let $\mathcal{I}$ be the set of all items (or binary features); subsets of $\mathcal{I}$ are called itemsets (or examples or patterns, depending on the context). The input to the frequent itemset mining problem is a multiset $\mathcal{D}$ of itemsets (examples described by their features), and a frequency threshold. The task is to output *all* frequent itemsets (patterns) and their frequencies, i.e., all subsets of $\mathcal{I}$ that exceed the given frequency threshold in the given data $\mathcal{D}$.

Example 1 *Assume the following problem specification:*

- *Set of all items* $\mathcal{I} = \{A, B, C, D\}$.
- *Data* $\mathcal{D} = \{\{A, B, C\}, \{A, D\}, \{B, C, D\}, \{A, B, C\}, \{C, D\}, \{B, C\}\}$.
- *Frequency threshold is 2.*

All possible itemsets and their frequencies:

Itemset	Frequency
$\{A\}$	**3**
$\{B\}$	**4**
$\{C\}$	**5**
$\{D\}$	**3**
$\{A,B\}$	**2**
$\{A,C\}$	**2**
$\{A,D\}$	1
$\{B,C\}$	**4**

Itemset	Frequency
$\{B,D\}$	1
$\{C,D\}$	**2**
$\{A,B,C\}$	**2**
$\{A,B,D\}$	0
$\{A,C,D\}$	0
$\{B,C,D\}$	1
$\{A,B,C,D\}$	0

The frequent itemsets are $\{A\}$, $\{B\}$, $\{C\}$, $\{D\}$, $\{A,B\}$, $\{A,C\}$, $\{B,C\}$, $\{C,D\}$, $\{A,B,C\}$.

The ▶hypothesis space for itemsets obviously is the power set of $\mathcal{I}$, and it has an exponential size ($2^{|\mathcal{I}|}$) in the number of items. Since all frequent itemsets are output, this is also the size of the output in the worst case (e.g., if the frequency threshold is zero, or if all examples in $\mathcal{D}$ equal $\mathcal{I}$), as well as the worst case time complexity.

In practical applications of frequent itemset mining, the size of the output as well as the running times are much smaller, but they strongly depend on the properties of the data and the frequency threshold. The useful range of thresholds varies enormously among different datasets. In many applications – such as ▶basket analysis – the number $|\mathcal{I}|$ of different items can be in thousands, even millions, while the typical sizes of examples are at most in dozens. In such sparse datasets a relatively small number of frequent itemsets can reveal the most outstanding co-occurrences; e.g., there are not likely to be very large sets of books typically bought by the same customers. In dense datasets, in turn, the number of frequent patterns can be overwhelming and also relatively uninformative. E.g., consider the dense dataset of books that have *not* been purchased by a customer: there are a huge number of sets of books that have not been bought by the same customers.

Theory/solutions

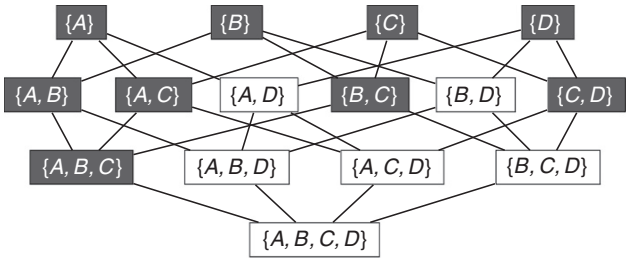
The most widely known solution for finding all frequent itemsets is the ▶Apriori algorithm (Agrawal, Mannila, Srikant, Toivonen, & Verkamo, 1996). It is based on the monotonicity of itemset frequencies (a ▶generalization relation): the frequency of a set is at most as high as the frequency of any of its subsets. Conversely, if a set is known to be infrequent, then none of its supersets can be frequent.

Apriori views the ▶hypothesis space of itemsets as a (refinement) lattice defined by set containment, and performs a ▶general-to-specific search using ▶breadth-first search. In other words, it starts with singleton itemsets, the most general and frequent sets, and proceeds to larger and less frequent sets. The search is pruned whenever a set does not reach the frequency threshold: all supersets of such sets are excluded from further search. Apriori deviates from standard breadth-first search by evaluating all sets of equal size in a single batch, i.e., it proceeds in a levelwise manner. This has no effect on the search structure or results, but can reduce disk access considerably for large databases. See the entry ▶Apriori Algorithm for an outline of the method.

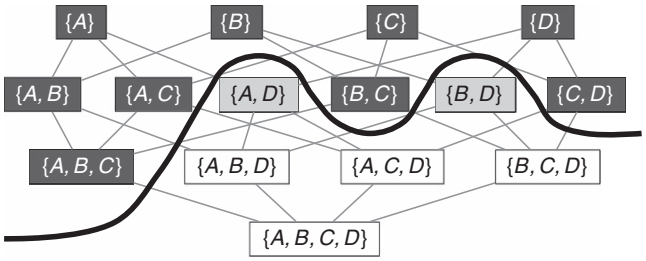
Example 2 *Figure 1 illustrates the search space for the data* $\mathcal{D}$ *of Example 1. Dark nodes represent frequent itemsets, i.e., the answer to the frequent itemset mining problem. Apriori traverses the space a level at a time. For instance, on the second level, it finds out that* $\{A,D\}$ *and* $\{B,D\}$ *are not frequent. It therefore prunes all their supersets, i.e., does not evaluate sets* $\{A,B,D\}$, $\{A,C,D\}$, *and* $\{B,C,D\}$ *on the third level.*

Other search strategies have also been applied. A ▶depth-first search without the subset check allows faster identification of candidates, at the expense of having more candidates to evaluate and doing that without natural batches (e.g., Zaki, 2000). FP-growth (Han, Pei, Yin, & Mao, 2004) uses a tree structure to store the information in the dataset, and uses it to recursively search for frequent itemsets.

The search strategy of Apriori is optimal in a certain sense. Consider the number of sets evaluated, and assume that for any already evaluated set we know whether it was frequent or not but do not consider its frequency. Apriori evaluates the frequencies of all frequent itemsets plus a number of candidates that turn out to be infrequent. It turns out that every infrequent candidate must actually be evaluated under the given assumptions: knowing which other sets are frequent and which are not does not help, regardless of the search



Frequent Pattern. Figure 1. The search space of frequent itemsets for data *D* of the running example. Dark nodes: frequent itemsets; white nodes: infrequent itemsets



Frequent Pattern. Figure 2. The positive border ($\{A,B,C\}$, $\{C,D\}$) and negative border ($\{A,D\}$, $\{B,D\}$) of frequent itemsets

order. This observation leads to the concept of *border*: the border consists of all those itemsets whose all proper subsets are frequent and whose all proper supersets are infrequent (Gunopulos et al., 2003; Mannila & Toivonen, 1997). The border can further be divided into two: the positive border contains those itemsets in the border that are frequent, the negative border contains those that are not. The positive border thus consists of the most specific patterns that are frequent, and corresponds to the "S" set of ▶version spaces.

Example 3 *Continuing our running example, Figure 2 illustrates the border between the frequent and infrequent sets. Either the positive or the negative border can alone be used to specify the collection of frequent itemsets: every frequent itemset is a subset of a set in the positive border* ($\{A,B,C\}$, $\{C,D\}$)*, while every infrequent itemset is a superset of a set in the negative border* ($\{A,D\}$, $\{B,D\}$)*.*

One variant of frequent itemset mining is to output the positive border only, i.e., to find the *maximal frequent itemsets* (Bayardo, 1998). This can be implemented with search strategies that do not need to evaluate the whole space of frequent patterns. This can be useful especially if the number of frequent itemsets is very large, or if the maximal frequent itemsets are large (in which case the number of frequent itemsets is large, too, since the number of subsets is exponential in the length of the maximal set). As a trade-off, the result does not directly indicate frequencies of itemsets.

Condensed Representations: Closed Sets and Nonderivable Sets Closed sets and nonderivable sets are a powerful concept for working with frequent itemsets, especially if the data is relatively dense or there are strong dependencies. Unlike the aforementioned simple model for borders, here also the known frequencies of sets are used to make inferences about frequencies of other sets.

As a motivation for closed sets (Pasquier, Bastide, Taouil, & Lakhal, 1999), consider a situation where the frequency of itemset $\{i,j\}$ equals the frequency of item j. This implies that whenever j occurs, so does i. Thus, any set $A \cup \{j\}$ that contains item j also contains item i, and the frequencies of sets $A \cup \{j\}$ and $A \cup \{i,j\}$ must be equal. As a result, it sufficies to evaluate sets $A \cup \{j\}$ to obtain the frequencies of sets $A \cup \{i,j\}$, too.

More formally, the *closure* of set A is its largest superset with identical frequency. A is *closed* iff it is its own closure, i.e., if every proper superset of A has a smaller

frequency than A. The utility of closed sets comes from the fact that frequent closed sets and their frequencies are a sufficient representation of all frequent sets. Namely, if B is a frequent set then its closure is a frequent closed set in $\mathcal{CL}$, where $\mathcal{CL}$ denotes the collection of all frequent closed itemsets. B's frequency is obtained as $\mathrm{fr}(B) = \max\{\mathrm{fr}(A) \mid A \in \mathcal{CL} \text{ and } B \subseteq A\}$. If B is not a frequent set, then it has no superset in $\mathcal{CL}$. ▶Formal concept analysis studies and uses closed sets and other related concepts.

Generators are a complementary concept, and also constitute a sufficient representation of frequent itemsets. (To be more exact, in addition to frequent generators, generators in the border are also needed). Set A is a *generator* (also known as a key pattern or a free set) if all its proper subsets have a larger frequency than A has. Thus, in an equivalence class of itemsets, defined by the set of examples in which they occur, the maximal element is unique and is the closed set, and the minimal elements are generators. The property of being a generator is monotone in the same way that being frequent is, and generators can be found with simple modifications to the Apriori algorithm.

Example 4 *Figure 3 illustrates the equivalence classes of itemsets by circles. For instance, the closure of itemset $\{A, B\}$ is $\{A, B, C\}$, i.e., whenever $\{A, B\}$ occurs in the data, C also occurs, but no other items. Given just the frequent closed sets and their frequencies, the frequency of, say, $\{B\}$ is obtained by finding its smallest frequent closed superset. It is $\{B, C\}$, with frequency 4, which is also B's frequency. Alternatively, using generators as the condensed representation, the frequency of itemset $\{B, C\}$ can be obtained by finding its maximal generator subset, i.e., $\{B\}$, with which it shares the same frequency.*

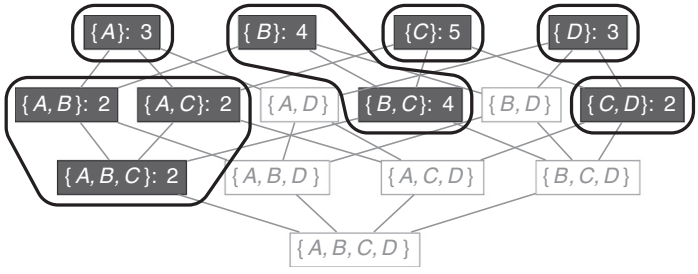
Nonderivability of an itemset (Calders & Goethals, 2002) is a more complex but often also a more powerful concept than closed sets. Given the frequencies of (some) subsets of itemset A, the frequency of A may actually be uniquely determined, i.e., there is only one possible consistent value. A practical method of trying to determine the frequency is based on deriving upper and lower bounds with inclusion–exclusion formula from the known frequencies of some subsets, and checking if these coincide. An itemset is derivable if this is indeed the case, otherwise it is *nonderivable*. Obviously, the collection of nonderivable frequent sets is a sufficient representation for all frequent sets.

Bounds for the absolute frequency of set I are obtained from its subsets as follows, for any $X \subseteq I$:

$$\mathrm{fr}(I) \leq \sum_{J: X \subseteq J \subset I} (-1)^{|I \setminus J|+1} \mathrm{fr}(J) \text{ if } |I \setminus X| \text{ is odd}, \quad (1)$$

$$\mathrm{fr}(I) \geq \sum_{J: X \subseteq J \subset I} (-1)^{|I \setminus J|+1} \mathrm{fr}(J) \text{ if } |I \setminus X| \text{ is even}. \quad (2)$$

Using all subsets X of I, one can obtain a number of upper and lower bounds. If the least upper bound equals the greatest lower bound, then set I is derivable. The conceptual elegance of this solution lies in the fact that derivable sets follow logically from the nonderivable ones – the aforementioned formula is one way of finding (some) such situations – whereas with closed sets the user must know the closure properties.



Frequent closed set $\mathcal{CL}$ = {{A},{C},{D},{B, C},{C, D},{A, B, C}}.
Frequent generators: {{A},{B},{C},{D},{A,B},{A,C},{C,D}}.

Frequent Pattern. Figure 3. Frequencies and equivalence classes of frequent itemsets in data $\mathcal{D}$ of the running example, and the corresponding closed sets and generators

Generalizations of Frequent Patterns The concept of frequent patterns has been extended in two largely orthogonal directions. One is to more complex patterns and data, such as frequent sequences, trees (see ►tree mining), graphs (see ►graph mining), and first-order logic (Dehaspe & Toivonen, 1999). The other direction to generalize the concept is to ►constraint-based mining, where other and more complex conditions are considered beyond frequency. We encourage the interested reader to continue at the entry for ►constraint-based mining, which also gives further insight into many of the more theoretical aspects of frequent pattern mining.

Programs and Data

Frequent itemset mining implementations repository: http://fimi.cs.helsinki.fi/
Weka: http://www.cs.waikato.ac.nz/ml/weka/
Christian Borgelt's implementations: http://www.borgelt.net/software.html
Data mining template library: http://dmtl.sourceforge.net/

Applications

Frequent patterns are a general purpose tool for data exploration, with applications virtually everywhere. Market ►basket analysis was the first application, telecom alarm correlation and gene mapping are examples of quite different application fields.

Future Directions

Work on frequent pattern mining is being expanded in several directions. New types of pattern languages are being developed, either to meet some specific needs or to increase the expressive power. Many of these developments are motivated by different types of data and applications. Within machine learning, frequent patterns are increasingly being used as a tool for feature construction in complex domains. For an end-user application, methods for choosing and ranking the most interesting patterns among thousands or millions of them is a crucial problem, for which there are no perfect solutions (cf. Geng & Hamilton, 2006). At the same time, theoretical understanding of the problem and solutions of frequent pattern discovery still has room for improvement.

Cross References

►Apriori Algorithm
►Association Rule
►Basket Analysis
►Constraint-Based Mining
►Data Mining
►Frequent Itemset
►Graph Mining
►Knowledge Discovery in Databases
►Tree Mining

Recommended Reading

Agrawal, R., Imielinski, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD international conference on management of data, Washington, DC* (pp. 207–216). New York: ACM.

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., & Verkamo, A. I. (1996). Fast discovery of association rules. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, & R. Uthurusamy (Eds.), *Advances in knowledge discovery and data mining* (pp. 307–328). Menlo Park, CA: AVAAI Press.

Bayardo, R. J. Jr. (1998). Efficiently mining long patterns from databases. In *Proceedings of the 1998 ACM SIGMOD international conference on management of data, Seatle, Washington, DC* (pp. 85–93). New York: ACM.

Calders, T., & Goethals, B. (2002). Mining all non-derivable frequent itemsets. In *Proceedings of the 6th European Conference on principles of data mining and knowledge discovery, Helsinki, Finland*. Lecture Notes in Computer Science (vol. 2431, pp. 74–85). London: Springer.

Ceglar, A., & Roddick, J. F. (2006). Association mining. *ACM Computing Surveys 38*(2): Article No. 5.

Dehaspe, L., & Toivonen, H. (1999). Discovery of frequent datalog patterns. *Data mining and knowledge discovery 3*(1): 7–36.

Geng, L., & Hamilton, H. J. (2006). Interestingness measures for data mining: A survey. *ACM Computing Surveys 38*(3): Article No. 9.

Gunopulos, D., Khardon, R., Mannila, H., Saluja, S., Toivonen, H., & Sharma, R. S. (2003). Discovering all most specific sentences. *ACM transactions on database systems 28*(2): 140–174.

Han, J., Pei, J., Yin, Y., & Mao, R. (2004). Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery 8*(1): 53–87.

Mannila, H., & Toivonen, H. (1997). Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery 1*(3): 241–258.

Pasquier, N., Bastide, Y., Taouil, R., & Lakhal, L. (1999). Discovering frequent closed itemsets for association rules. In *Proceedings of 7th international conference on database theory, Jerusalem, Israel*. Lecture Notes in Computer Science (vol. 1540, pp. 398–416). London: Springer.

Zaki, M. J. (2000). Scalable algorithms for association mining. In *IEEE transactions on knowledge and data engineering* 12(3): 372–390.

Frequent Set

►Frequent Itemset

Functional Trees

►Model Trees

Fuzzy Sets

Fuzzy sets were introduced by Lofti Zadeh as a generalization of the concept of a regular set. A fuzzy set is characterized by a membership function that assigns a degree (or grade) of membership to all the elements in the universe of discourse. The membership value is a real number in the range [0, 1], where 0 denotes no definite membership, 1 denotes definite membership, and intermediate values denote partial membership to the set. In this way, the transition from nonmembership to membership in a fuzzy set is gradual and not abrupt like in a regular set, allowing the representation of imprecise concepts like "small," "cold," "large," or "very" for example.

A variable with its values defined by fuzzy sets is called a linguistic variable. For example, a linguistic variable used to represent a temperature can be defined as taking the values "cold," "comfortable," and "warm," each one of them defined as a fuzzy set. These linguistic labels, which are imprecise by their own nature, are, however, defined very precisely by using fuzzy set concepts.

Based on the concepts of fuzzy sets and linguistic variables, it is possible to define a complete fuzzy logic, which is an extension of the classical logic but appropriate to deal with approximate knowledge, uncertainty, and imprecision.

Recommended Reading

Zadeh, L. A. (1965). Fuzzy sets. *Information and control*. 8(3): 338–353.

Fuzzy Systems

A fuzzy system is a computing framework based on the concepts of the theory of ►fuzzy sets, fuzzy rules, and fuzzy inference. It is structured in four main components: a knowledge base, a fuzzification interface, an inference engine, and a defuzzification interface. The knowledge base consists of a rule base defined in terms of fuzzy rules, and a database that contains the definitions of the linguistic terms for each input and output linguistic variable. The fuzzification interface transforms the (crisp) input values into fuzzy values, by computing their membership to all linguistic terms defined in the corresponding input domain. The inference engine performs the fuzzy inference process, by computing the activation degree and the output of each rule. The defuzzification interface computes the (crisp) output values by combining the output of the rules and performing a specific transformation.

Fuzzy systems can be classified in different categories. The most widely used are the Mamdani and the Takagi-Sugeno models. In a Mamdani fuzzy system the output variables are defined as linguistic variables while in a Takagi-Sugeno fuzzy system they are defined as a linear combination of the input variables.

Fuzzy systems can model nonlinear functions of arbitrary complexity, however, their main strength comes from their ability to represent imprecise concepts and to establish relations between them.

Recommended Reading

Mamdani, E. H., & Assilian, S. (1975). An experiment in linguistic synthesis with a fuzzy logic controller. *International journal of man-machine studies*. 7(1): 1–13.

Sugeno, M. (1985) *Industrial applications of fuzzy control*. Elsevier Science Publishers, New York.