S

Sample Complexity

►Generalization Bounds

Samuel's Checkers Player

Definition

Samuel's Checkers Player is the first machine learning system that received public recognition. It pioneered many important ideas in game playing and machine learning. The two main papers describing his research (Samuel, 1959, 1967) became landmark papers in Artificial Intelligence. In one game, the resulting program was able to beat one of America's best players of the time.

Description of the Learning System

Samuel's checkers player featured a wide variety of learning techniques. First, his checkers player remembered positions that it frequently encountered during play. This simple form of *rote learning* allowed it to save time, and to search deeper in subsequent games whenever a stored position was encountered on the board or in some line of calculation. Next, it featured the first successful application of what is now known as ►Reinforcement Learning for tuning the weights of its evaluation function. The program trained itself by playing against a stable copy of itself. After each move, the weights of the evaluation function were adjusted in a way that moved the evaluation of the root position after a quiescence search closer to the evaluation of the root position after searching several moves deep. This technique is a variant of what is nowadays known as ►Temporal-Difference Learning and commonly used in successful game-playing programs. Samuel's program not only tuned the weights of the evaluation but also employed on-line ►Feature Selection for constructing the evaluation function with the terms that seem to be the most significant for evaluating the current board situation. ►Feature Construction was recognized as the key problem that still needs to be solved. Later, Samuel changed his evaluation function from a linear combination of terms into a structure that closely resembled a 3-layer ►Neural Network. This structure was trained with ►Preference Learning from several thousand positions from master games.

Cross References

►Machine Learning and Game Playing

Recommended Reading

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development, 3*(3), 211–229.

Samuel, A. L. (1967). Some studies in machine learning using the game of checkers. II – recent progress. *IBM Journal of Research and Development, 11*(6), 601–617.

Saturation

►Bottom Clause

SDP

►Symbolic Dynamic Programming

Search Bias

►Learning as Search

Search Engines: Applications of ML

Eric Martin
University of New South Wales,
Sydney, NSW, Australia

Definition

Search engines provide users with Internet resources – links to web sites, documents, text snippets, images, videos, etc. – in response to queries. They use techniques that are part of the field of information retrieval, and rely on statistical and pattern matching methods. Search engines have to take into account many key aspects and requirements of this specific instance of the information retrieval problem. First, the fact is that they have to be able to process hundreds of millions of searches a day and answer queries in a matter of milliseconds. Second, the resources on the World Wide Web are constantly updated, with information being continuously added, removed or changed – the overall contents changing by up to 8% a week – in a pool consisting of billions of documents. Third, the users express possibly semantically complex queries in a language with limited expressive power, and often not make use or proper use of available syntactic features of that language – for instance, the boolean *or* operator occurs in less than 3% of queries.

Motivation and Background

Web searching is technically initiated by sending a query to a search engine but the whole search process starts earlier, in the mind of the person who conducts the search. To be successful, the process needs to provide users with words, text snippets, images, or movies that fulfill the users' quest for information. Thus, though a search is technically the implementation of a procedure that maps a query to some digital material, it spans a larger spectrum of activities, from a psychological trigger to a psychological reward. For a given set of digital material that, if provided, would be deemed perfectly satisfactory by a number of users looking for the same information, different users will issue different queries. This might be because they have varying skills at conveying what they are after in the form of a few words. This, in turn, might be because their understanding of the technology prompts them to formulate what they are after in a form that, rightly or wrongly, they consider appropriate for a computing device to process. That might be for a number of different reasons that all point to the fact that the quality of the search is not determined by its adequacy to the query, but by its adequacy to the psychological trigger that produced the query. This especially makes Web searching challenging and exciting area in the field of ►information retrieval.

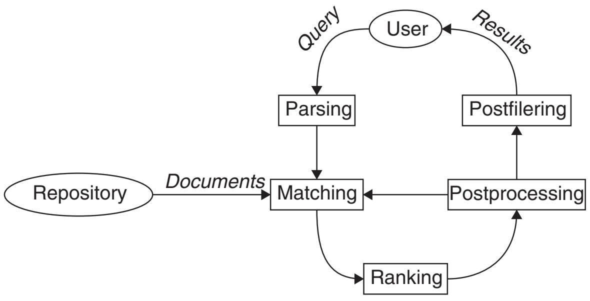
In Broder (2002), it is suggested that web queries can be classified in three classes:

- *Navigational queries* expect the search to return a particular url. For instance, http://www.cityrail.info is probably the expected result to the query `Cityrail` for a Sydneysider.
- *Transactional queries* expect the search to return links to sites that offer further interaction, for example for online shopping or to download music. For instance, http://www.magickeys.com/books/, where books for young children are available for download, is probably a good result to the query `children stories`.
- *Informational queries* expect the search to reveal the information, that is, the correct answer to a question. This information can be immediately provided in the page where the results of the search are displayed, for instance, `Bern` for the query `capital of switzerland`. Or it can be provided in the pages accessible from the first links returned by the search, as for instance `Italy` that is easily found in the web page accessed from the first link returned in response to the query `football world champion 1982`.

Answering an informational query with the information itself, rather than with links to documents where the information is to be found, is one of the most difficult challenges that search engines developers have started addressing.

Structure of the Learning System

The general structure of a search engine can be illustrated as follows:



A ►string matching algorithm is applied to the parsed query issued by the user and to an indexed representation of a set of documents, resulting in a ranked subset of the latter. This ranked set of documents can be subjected to a postprocessing procedure whose aim is to improve the results by either refining the query or by analyzing the documents further, possibly over many iterations, until the results stabilize and can be returned to the user, following a postfiltering procedure to display the information appropriately.

Retrieval Methods

The existence of hyperlinks between documents distinguishes search engines from other information retrieval applications. All techniques developed in the field of information retrieval are potentially relevant for extracting information from the web, but benefits from a proper analysis of the cross-reference structure. That is, to measure the degree of relevance of a document to a given query, one can take advantage of a prior ranking of all documents independent of that query or any other, following a sophisticated version of the PageRank (Page, Brin, Motwani, & Winograd, 1999) link analysis algorithm. One of the simplest versions of the algorithm recursively defines the PageRank $\mathrm{PR}(T)$ of a page T which pages $T_1, \ldots, T_n$ point to, amongst the $c_1, \ldots, c_n$ pages $T_1, \ldots, T_n$ point to, respectively, as

$$\frac{1-d}{N} + d(T_1/c_1 + \cdots + T_n/c_n),$$

where N is the total number of pages and d, a *damping factor*, represents the probability that a user decides to follow a link rather than randomly visit another page; normalizing the solution so that the PageRanks of all pages add up to 1, $\mathrm{PR}(T)$ then represents the probability that a user visits T by clicking on a link.

Boolean retrieval is one of the simplest methods to retrieve a set of documents that match exactly a query expressed as a boolean combination of keywords. The match is facilitated by using an *inverted file* indexing structure which associates every possible keyword with links to the documents in which it occurs. If extra information is kept on the occurrences of keywords in documents (number of occurrences, part of the document in which they occur, font size and font type used for their display, etc.) then the results can also be ranked. But *best match* models, as opposed to *exact match* models, are better suited to producing ranked results. The *vector space* model is one of the earliest and most studied models of this kind. It represents documents and queries as vectors over a space each of whose dimensions represents a possible keyword, and measures the similarity between the vectors $\vec{q}$ and $\vec{d}$ whether it occurs at least once in query and document, respectively, record for each keyword as the cosine of the angle formed by $\vec{q}$ and $\vec{d}$, namely,

$$\frac{\vec{q}\vec{d}}{\|\vec{q}\|\,\|\vec{d}\|},$$

that is all the most closer to 1 that query and document have more in common. The *term-frequency-inverse-document-frequency* (tf-idf) model refines the encoding given by $\vec{d}$ by replacing a value of 1 in the ith dimension, indicating the existence of an occurrence of the ith keyword in $\vec{d}$, with

$$c_1 \log\left(\frac{N}{c_2}\right),$$

where c_1 is the number of occurrences of the ith keyword in the document, N is the total number of documents, and c_2 is the number of documents in the whole collection that contain at least one occurrence of the ith keyword; so more importance is given to keywords that occur more and that occur "almost exclusively" in the document under consideration. One of the most obvious issues with this approach is that the number of dimensions is huge and the vectors are sparse. Another important issue is that set of vectors determined by the set of keywords is not orthogonal, and not even linearly independent, because two given keywords can be synonyms (sick and ill), not semantically related (garlic and manifold), or more or less semantically related (wheel and tire).

The *extended vector space* model (Robertson, Walker, & Beaulieu, 1999b) addresses this issue assuming that the similarity between two keywords is captured by the symmetric difference between the set of documents that contain a keyword and the set of documents that contain the other, ranging from identical sets (similar keywords) to disjoint sets (unrelated keywords). Let $D_1, \dots, D_{N'}$ be an enumeration of the quotient relation over the set of all documents such that two documents are equivalent if they contain precisely the same keywords (so N' is at most equal to N, the number of documents in the whole collection). Conceive of an N'-dimensional vector space S of which $D_1, \dots, D_{N'}$ is a basis. Associate the ith keyword with the vector $\vec{v}_i$ of S defined as $1/\sqrt{w_1^2 + \cdots + w_{N'}^2}(w_1, \dots, w_{N'})$ where for all nonzero $k \leq N'$, w_k is the number of occurrences of the ith keyword in all documents that belong to class D_k. Then associate a document with the vector $\vec{d}$ of S defined as $\alpha_1\vec{v}_1 + \cdots + \alpha_{N''}\vec{v}_{N''}$ where N'' is the number of keywords and for all nonzero $k \leq N''$, α_k is the number of occurrences of the ith keyword in that document, and associate a query with the vector $\vec{q}$ of S defined as $\beta_1\vec{v}_1 + \cdots + \beta_{N''}\vec{v}_{N''}$ where for all nonzero $k \leq N''$, β_k is equal to 1 if the ith keyword occurs in the query, and to 0 otherwise. The similarity between $\vec{q}$ and $\vec{d}$ is then measured as described for the simple vector space method.

The *topic-based vector space* model (Becker & Kuropka, 2003) also replaces the original vector space with a different vector space of a different dimension, addressing the issue of nonorthogonality between keywords thanks to *fundamental topics*, assumed to be pairwise independent, using ontologies; the fundamental topics then provide the vector basis which is a linear combination of a given keyword. So the topic-based vector space model conceives of the meaning of words as the semantic relationships that emerge from the common use of a language by the members of a given community, whereas the extended vector space model conceives of the meaning of words as the syntactic relationship of term co-occurrence with respect to the repository of documents being processed.

Probabilistic retrieval frameworks aim at estimating the probability that a given document is relevant to a given query. Given a keyword w, denote by p_w^+ the probability that w occurs in a document relevant to w, and denote by p_w^- the probability that w occurs in a document not relevant to w. Many probabilistic retrieval frameworks then define the relevance of a document to a query as follows, where $w_1, \dots, w_n$ are the keywords that occur both in the query and in the document:

$$\sum_{i=1}^{n} \log\left(\frac{p_{w_i}^+(1-p_{w_i}^-)}{p_{w_i}^-(1-p_{w_i}^+)}\right).$$

This quantity increases all the more that the document contains more words that are more likely to occur in relevant documents, and more words less likely to occur in irrelevant documents. Different frameworks suggest different ways to evaluate the values of $p_{w_i}^+$ and $p_{w_i}^-$. For instance, p_i is sometimes assumed to be constant and $p_{w_i}^-$ defined as n_i/N where N is the total number of documents and n_i the number of documents in which w_i occurs, capturing the fact that a document containing a keyword appearing in few other documents is likely to be relevant to that keyword, in which case the previous formula can be rewritten

$$c\sum_{i=1}^{n} \log\left(\frac{N-n_i}{n_i}\right)$$

for some constant c. More sophisticated methods have been developed to better estimate the probabilities, such as the *Okapi weighting document score* (Robertson, Walker, & Beaulieu, 1999a) which defines the relevance of a document to a query as

$$\sum_{i=1}^{n} \log\left(\frac{N-n_i+0.5}{n_i+0.5}\right)\frac{(k_1+1)c_i}{(k_1(1-b)+b(l/\beta))+c_i} \times \frac{(k_3+1)d_i}{k_3+d_i},$$

where the notation is as above, with the addition of c_i to denote the number of occurrences of w_i in the document, d_i to denote the number of occurrences of w_i in the query, l to denote the number of bytes in the document, β to denote the average number of bytes in a document, and b, k_1, and k_3 to denote constants.

Query Classification

The development of effective methods of information retrieval from web resources requires a good understanding of users' needs and practice. In Markev (2007a), the following questions are identified as being especially relevant towards gaining such an understanding.

- What characterizes the queries that end users submit to online IR systems? What search features do people use? What features would enable them to improve on the retrievals they have in hand? What features are hardly ever used? What do end users do in response to the system's retrievals?

This chapter indicates that many of the basic features of information retrieval systems are poorly used. For instance, less than 15, 3, and 2% of queries make use of the *and*, *or*, and *not* boolean operators, respectively, and less than 15% of queries of enclosing quotes; the wrong syntax is often used, resulting in incorrect use of advanced search features in one third of the cases; less than 10% of queries take advantage of ►relevance feedback. Based on those findings, the second part (Markev, 2007b) of the article suggests *two dozen new research questions* for researchers in information retrieval, while noting that about 70% of users are satisfied with their search experience.

Evaluating search satisfaction has received lots of attention. In Fox, Karnawat, Mydland, Dumais, and White (2005), both explicit and implicit measures of satisfaction are collected. Explicit measures are obtained by prompting the user to evaluate a search result as satisfying, partially satisfying, or not satisfying, and similarly to evaluate satisfaction gained from a whole search session. Implicit measures are obtained by recording mouse and keyboard actions, time spent on a page, scrolling actions and durations, number of visits to a page, position of page in results list, number of queries submitted, number of results visited, etc. A Bayesian model can be used to infer the relationships between explicit and implicit measures of satisfaction. This chapter reports on two ►Bayesian networks that were built to predict satisfaction for individual page visits and satisfaction for entire search sessions – w.r.t. the feedback obtained from both kinds of prompts – with evidence that a combination of well chosen implicit satisfaction measures can be a good predictor of explicit satisfaction. Referring to the categorization of web queries in Broder (2002) as *user goals*, it is proposed in Lee, Liu, and Cho (2005) to build *click distributions* by sorting results to a query following the numbers of clicks they received from all users, and suggested that highly skewed distributions should correspond to navigational queries, while flat distributions should correspond to informational queries. The same kind of considerations are also applied to *anchor-link distributions*, the anchor-link distribution of a query being defined as the function that maps a URL to the number of times that URL is the destination of an anchor that has the same text as the query.

Finer techniques of query classification are proposed in Beitzel, Jensen, Lewis, Chowdhury, and Frieder (2007), where is a rule-based automatic classier is produced from *selectional preferences*. A query consisting of at least two keywords is split into a head x and a tail y, and then converted into a *forward* pair (x, u) and a *backward* pair (u, y), where u represents a category, that is, a generic term that refers to a list of semantically related words in a thesaurus. For instance, the query "interest rate" can (only) be split into $(\text{interest}, \text{rate})$ and converted to the forward pair $(\text{interest}, \text{personal finance})$ where "personal finance" denotes the list consisting of the terms "banks," "rates," "savings," etc.; so the first keyword – "interest" – provides context for the second one. Given a large query log, the *maximum likelihood estimate* (MLE) of $P(u/x)$, the probability that a query decomposed as (x, z) is such that z belongs to category u, is defined as the quotient between the number of queries in the log that have (x, u) as a forward pair and the number of queries in the log that can be decomposed as (x, z). This allows one to write a forward rule of the form "x Y classified as u with weight p," where p is the MLE of $P(u/x)$, provided that the *selectional preference strength* of x be above some given threshold. The rule can then be applied to incoming queries, such as "interest only loan" by matching a final or initial segment of the query – depending on whether forward or backward rules are under consideration – and suggest possible classifications; with the running example, "interest only loan" would then be classified as "personal finance with weight p" if a forward rule of the form "interest Y classified as personal finance with weight p" had been discovered. Such a classification can then be used to rewrite the query, or to send it to an appropriate database-backend if many domain-specific databases are available.

Cross References

►Bayesian Methods
►Classification

►Covariance Matrix
►Rule Learning
►Text Mining

Recommended Reading

Becker, J., & Kuropka, D. (2003). Topic-based vector space model. In W. Abramowicz & G. Klein (Eds.), *Proceedings of the sixth international conference on business information systems* (pp. 7–12). Colorado Springs, CO.

Beitzel, S. M., Jensen, E. C., Lewis, D. D., Chowdhury, A., & Frieder, O. (2007). Automatic classification of web queries using very large unlabeled query logs. *ACM Transactions on Information Systems, 25*(2), 9. ISSN: 1046-8188

Broder, A. (2002). A taxonomy of web search. *SIGIR Forum, 36*(2), 3–10.

Fox, S., Karnawat, K., Mydland, M., Dumais, S., & White, T. (2005). Evaluating implicit measures to improve web search. *ACM Transactions on Information Systems, 23*(2), 147–168.

Lee, U., Liu, Z., & Cho, J. (2005). Automatic identification of user goals in web search. In *WWW '05*: In *Proceedings of the 14th international conference on World Wide Web* (pp. 391–400). New York: ACM Press.

Markev, K. (2007a). Twenty-five years of end-user searching, part 1: Research findings. *Journal of the American Society for Information Science and Technology, 58*(8), 1071–1081.

Markev, K. (2007b). Twenty-five years of end-user searching, part 2: Future research directions. *Journal of the American Society for Information Science and Technology, 58*(8), 1123–1130.

Page, L., Brin, S., Motwani, R., & Winograd, T. (1999). *The pagerank citation ranking: Bringing order to the web*. Technical report. Stanford, CA: Stanford University Press.

Robertson, S. E., Walker, S., & Beaulieu, M. (1999a). Okapi at trec-7: Automatic ad hoc, filtering, VLC and filtering tracks. In E. Voorhees & D. Harman (Eds.), In *Proceedings of the seventh text retrieval conference* (pp. 253–264). Gaithersburg, MD.

Robertson, S. E., Walker, S., & Beaulieu, M. (1999b). On modeling of information retrieval concepts in vector spaces. *ACM Transactions on Database Systems, 12*(2), 299–321.

Self-Organizing Feature Maps

►Self-Organizing Maps

Self-Organizing Maps

Samuel Kaski
Helsinki University of Technology, Finland

Synonyms

Kohonen maps; Self-organizing feature maps; SOM

Definition

Self-organizing map (SOM), or Kohonen Map, is a computational data analysis method which produces nonlinear mappings of data to lower dimensions. Alternatively, the SOM can be viewed as a ►clustering algorithm which produces a set of clusters organized on a regular grid. The roots of SOM are in neural computation (see ►neural networks); it has been used as an abstract model for the formation of ordered maps of brain functions, such as sensory feature maps. Several variants have been proposed, ranging from dynamic models to Bayesian variants. The SOM has been used widely as an engineering tool for data analysis, process monitoring, and information visualization, in numerous application areas.

Motivation and Background

The SOM (Kohonen, 1982, 2001) was originally introduced in the context of modeling of how the spatial organization of brain functions forms. Formation of feature detectors selective to certain sensory inputs, such as orientation-selective visual neurons, had earlier been modeled by ►competitive learning in neural networks, and some models of how the feature detectors become spatially ordered had been published (von der Malsburg, 1973). The SOM introduced an *adaptation kernel* or *neighborhood function* that governs the adaptation in such networks; while in plain competitive learning only the winning neuron that best matches the inputs adapts, in SOM all neurons within a local neighborhood of the winner learn. The neighborhood is determined by the neighborhood function. The SOM is an algorithm for computing such ordered mappings.

While some of the motivation of the SOM comes from neural computation, its main uses have been as a practical data analysis method. The SOM can be viewed as a topographic vector quantizer, a nonlinear projection method, or a clustering method. In particular, it is a clustering-type algorithm that orders the clusters. Alternatively, it is a nonlinear projection-type algorithm that clusters, or more specifically quantizes, the data.

The SOM was very popular in the 1990s and still is; it is intuitively relatively easily understandable, yet hard to analyze thoroughly. It connects many research traditions and works well in practice. An impressive set of variants have been published over the years, of which probabilistic variants (e.g., Bishop, Svensén, & Williams (1998) and Heskes (2001)) are perhaps closest to the current mainstream machine learning. While there currently are excellent alternative choices for many of the specific tasks SOMs have been applied for over the years, even the basic SOM algorithm is still viable as a versatile engineering tool in data-analysis tasks.

Structure of Learning System

The SOM consists of a regular grid of nodes (Fig. 1). A *model* of data has been attached to each node. For vector-valued data $\mathbf{x} = [x_1, \ldots, x_d]^T$, the models are vectors in the same space; the model at the ith node is $\mathbf{m}_i = [m_{i1}, \ldots, m_{id}]$. The models define a mapping from the grid to the data space. The coordinates on the grid are uniquely determined by the index i of a node, and the model $\mathbf{m}_i$ gives the location in the data space. The whole grid becomes mapped into an "elastic net" in the data space. While being a mapping from the grid to the input space, the SOM defines a projection from the input space to the discrete grid locations as well; each data point is projected to the node having the closest model.

The original online SOM algorithm updates the model vectors toward the current input vector at time t,

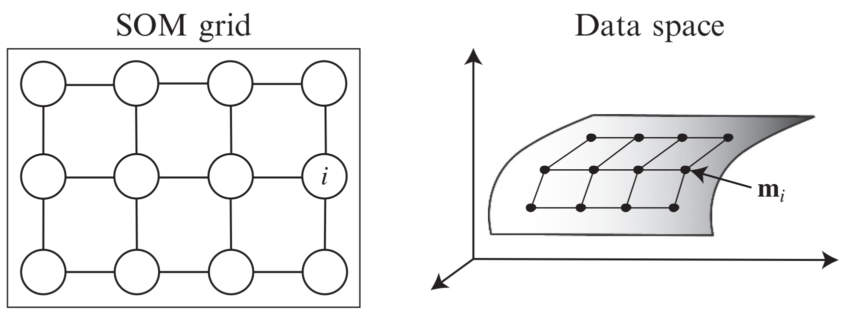
$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + h_{ci}(t)(\mathbf{x}(t) - \mathbf{m}_i(t)) \; .$$

Here c is the index of the unit having the closest model vector to $\mathbf{x}(t)$, and $h_{ci}(t)$ is the neighborhood function or adaptation kernel. The kernel is a decreasing function of the distance between the units i and c on the grid; it forces neighboring units to adapt toward similar input samples. The height and width of h are decreasing functions of time t. In an iteration over time and over the different inputs, the model vectors become ordered and specialize to represent different regions of the input space.

The online version of ►K-means clustering is a special case of the SOM learning rule, where only the closest model vector is adapted. That is, the neighborhood function is $h_{ci}(t) = \alpha(t)$ for $i = c$ and $h_{ci} = 0$ otherwise. Here $\alpha(t)$ is the adaptation coefficient, a decreasing scalar. In short, K-means and SOM use the prototypes in the same way, but in SOM the prototypes have an inherent order that stems from fixing them onto a grid and updating the prototypes to represent both the data mapped to themselves and to their neighbors.

A neural interpretation of the SOM adaptation process is that the nodes are feature detector neurons or processing modules that in a ►competitive learning process become specialized to represent different kinds of inputs. The neighborhood function is a plasticity kernel that forces neighboring neurons to adapt at the same time. The kernel transforms the discrete set of feature detectors into feature maps analogous to ordered brain maps of sensor inputs, and more generally to maps of more abstract properties of the input data.

A third interpretation of the SOM is as a vector quantizer. The task of a vector quantizer is to encode inputs with indexes of prototypes, often called codebook vectors, such that a distortion measure is minimized. If there is noise that may change the indexes, the



Self-Organizing Maps. Figure 1. A schematic diagram showing how the SOM grid of units (circles on the left, neighbors connected with lines) corresponds to an "elastic net" in the data space. The mapping from the grid locations, determined by the indices i, to the data space is given by the model vectors $\mathbf{m}_i$ attached to the units i

distribution of the noise should be used as the neighborhood function, and then the distortion becomes minimized by a variant of SOM (Luttrell, 1994). In summary, the SOM can be viewed as an algorithm for producing codebooks ordered on a grid.

While it has turned out to be hard to rigorously analyze the properties of the SOM algorithm (Fort, 2006), its fixed points may be informative. In a fixed point the models must fulfill

$$m_i = \frac{\sum_{\mathbf{x}} h_{c(\mathbf{x}),i}\mathbf{x}}{\sum_{\mathbf{x}} h_{c(\mathbf{x}),i}} ,$$

that is, each model vector is in the centroid of data projected to it and its neighbors. The definition of a *principal curve* (Hastie, Tibshirani, & Friedman, 2001), a nonlinear generalization of principal components (see ►principle components analysis), essentially is that the curve goes through the centroid of data projected to it. Hence, one interpretation of the SOM is a discretized, smoothed, nonlinear generalization of principal components. In short, SOMs aim to describe the variation in the data nonlinearly with their discrete grids.

Finally, a popular prototype-based classifier, ►learning vector quantization (LVQ) (Kohonen, 2001), can be loosely interpreted as a variant of SOMs, although it does not have the neighborhood function and hence, the prototypes do not have an order.

Programs and Data

The SOM has been implemented in several commercial packages and as freeware. Two examples, SOM_PAK written in C and Matlab SOM Toolbox (http://www.cis.hut.fi/research/software) came from Kohonen's group.

Applications

The SOM can be used as a nonlinear dimensionality reduction method, by projecting each data vector into the grid location having the closest model vector. An image of the grid can be used for *information visualization*. Since all grid locations are clusters, the SOM display actually visualizes an ordered set of clusters, or a quantized image of the principal manifold in data. More specifically, the SOM units can be thought of as subclusters, and data clusters may form larger areas on the SOM grid.

SOM-based visualizations can be used for illustrating the proximity relationships of data vectors, such as documents in the WEBSOM document maps (Kohonen et al., 2000), or monitoring the change of a system such as an industrial process or the utterances of a speaker, as a trajectory on the SOM display. More applications can be found in a collected bibliography (the latest one is Pöllä, Honkela, & Kohonen (in press)).

Cross References

►ART
►Competitive Learning
►Dimensionality Reduction
►Hebbian Learning
►K-means Clustering
►Learning Vector Quantization

Recommended Reading

Bishop, C. M., Svensén, M., & Williams, C. K. I. (1998). GTM: The generative topographic mapping. *Neural Computation, 10*, 215–234.
Fort, J. C. (2006). SOM's mathematics. *Neural Networks, 19*, 812–816.
Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The elements of statistical learning*. New York: Springer.
Heskes, T. (2001). Self-organizing maps, vector quantization, and mixture modeling. *IEEE Transactions on Neural Networks, 12*, 1299–1305.
Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics, 43*, 59–69.
Kohonen, T. (2001). *Self-organizing maps* (3rd ed.). Berlin: Springer.
Kohonen, T., Kaski, S., Lagus, K., Salojärvi, J., Honkela, J., Paatero, V., et al. (2000). Self organization of a massive document collection. *IEEE Transactions on Neural Networks, 11*, 574–585.
Luttrell, S. P. (1994). A Bayesian analysis of self-organizing maps. *Neural Computation, 6*, 767–794.
Pöllä, M., Honkela, T., & Kohonen, T. (2009). Bibliography of self-organizing map (SOM) papers: 2002-2005 addendum. Report TKK-ICS-R23, Helsinki University of Technology, Department of Information and Computer Science, Espoo, Finland.
von der Malsburg, C. (1973). Self-organization of orientation sensitive cells in the striate cortex. *Kybernetik, 14*, 85–100.

Semantic Mapping

►Text Visualization

Semi-Naive Bayesian Learning

Fei Zheng, Geoffrey I. Webb
Monash University,
Clayton, Melbourne,
Victoria, Australia

Definition

Semi-naive Bayesian learning refers to a field of ►Supervised Classification that seeks to enhance the classification and conditional probability estimation accuracy of ►naive Bayes by relaxing its attribute independence assumption.

Motivation and Background

The assumption underlying ►naive Bayes is that attributes are independent of each other, given the class. This is an unrealistic assumption for many applications. Violations of this assumption can render naive Bayes' classification suboptimal. There have been many attempts to improve the classification accuracy and probability estimation of naive Bayes by relaxing the attribute independence assumption while at the same time retaining much of its simplicity and efficiency.

Taxonomy of Semi-Naive Bayesian Techniques

Semi-naive Bayesian methods can be roughly subdivided into five high-level strategies for relaxing the independence assumption.

- The first strategy forms an attribute subset by deleting attributes to remove harmful interdependencies and applies conventional naive Bayes to this attribute subset.
- The second strategy modifies naive Bayes by adding explicit interdependencies between attributes.
- The third strategy accommodates violations of the attribute independence assumption by applying naive Bayes to a subset of training set. Note that the second and third strategies are not mutually exclusive.
- The fourth strategy performs adjustments to the output of naive Bayes without altering its direct operation.
- The fifth strategy introduces hidden variables to naive Bayes.

Methods That Apply Naive Bayes to a Subset of Attributes

Due to the attribute independence assumption, the accuracy of naive Bayes is often degraded by the presence of strongly correlated attributes. Irrelevant attributes may also degrade the accuracy of naive Bayes, in effect increasing variance without decreasing bias. Hence, it is useful to remove both strongly correlated and irrelevant attributes.

Backward sequential elimination (Kittler, 1986) is an effective wrapper technique to select an attribute subset and has been profitably applied to naive Bayes. It begins with the complete attribute set and iteratively removes successive attributes. On each iteration, naive Bayes is applied to every subset of attributes that can be formed by removing one further attribute. The attribute whose deletion most improves training set accuracy is then removed, and the process repeated. It terminates the process when subsequent attribute deletion does not improve training set accuracy. Conventional naive Bayes is then applied to the resulting attribute subset.

One extreme type of interdependencies between attributes results in a value of one being a generalization of a value of the other. For example, *Gender=female* is a generalization of *Pregnant=yes*. Subsumption resolution (SR) (Zheng & Webb, 2006) identifies at classification time pairs of attribute values such that one appears to subsume (be a generalization of) the other and delete the generalization. It uses the criterion $|T_{x_i}| = |T_{x_i,x_j}| \geq u$ to infer that attribute value x_j is a generalization of attribute value x_i, where $|T_{x_i}|$ is the number of training cases with value x_i, $|T_{x_i,x_j}|$ is the number of training cases with both values, and u is a user-specified minimum frequency. When SR is applied to naive Bayes, the resulting classifier acts as naive Bayes except that it deletes generalization attribute-values at classification time if a specialization is detected.

Methods That Alter Naive Bayes by Allowing Interdependencies between Attributes

Interdependencies between attributes can be addressed directly by allowing an attribute to depend on other non-class attributes. Sahami (1996) introduces the terminology of the z-dependence Bayesian classifier, in which each attribute depends upon the class and at most z other attributes. Figure 1 depicts methods in this group from the ►Bayesian Network perspective.

In Fig. 1a, each attribute depends on the class and at most one another attribute. ►Tree Augmented Naive Bayes (TAN) (Friedman, Geiger, & Goldszmidt, 1997) is a representative one-dependence classifier. It efficiently finds a directed spanning tree by maximizing the log-likelihood and employs this tree to perform classification. SuperParent TAN (Keogh & Pazzani, 1999) is an effective variant of TAN.

A SuperParent one-dependence classifier (Fig. 1b) is a special case of one-dependence classifiers, in which an attribute called the SuperParent (X_1 in this graph), is selected as the parent of all the other attributes. ►Averaged One-Dependence Estimators (AODE) (Webb, Boughton, & Wang, 2005) selects a restricted class of one-dependence classifiers and aggregates the predictions of all qualified classifiers within this class. Maximum a posteriori linear mixture of generative distributions (MAPLMG) (Cerquides & Mántaras, 2005) extends AODE by assigning a weight to each one-dependence classifier.

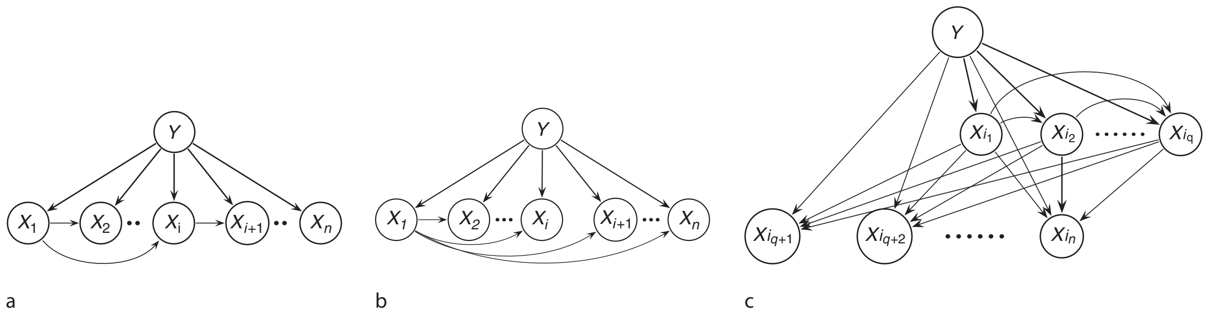
Two z-dependence classifiers ($z \geq 0$) are NBTree (Kohavi, 1996) and lazy Bayesian rules (LBR) (Zheng & Webb, 2000), both of which may add any number of non-class-parents for an attribute. In Fig. 1c, attributes in $\{X_{i_{q+1}}, \ldots, X_{i_n}\}$ depend on all the attributes in $\{X_{i_1}, \ldots, X_{i_q}\}$. The main difference between these two methods is that NBTree builds a single tree for all training instances while LBR generates a Bayesian rule for each test instance.

Methods That Apply Naive Bayes to a Subset of the Training Set

Another effective approach to accommodating violations of the conditional independence assumption is to apply naive Bayes to a subset of the training set, as it is possible that the assumption, although violated in the whole training set, may hold or approximately hold in a subset of the training set. NBTree and LBR use a local naive Bayes to classify an instance and can also be classified into this group. Locally weighted naive Bayes (LWNB) (Frank, Hall, & Pfahringer, 2003) applies naive Bayes to a neighborhood of the test instance, in which each instance is assigned a weight decreasing linearly with the Euclidean distance to the test instance. The number of instances in the subset is determined by a user-specified parameter. Only those instances whose weights are greater than zero are used for classification.

Methods That Calibrate Naive Bayes' Probability Estimates

Methods in this group make adjustments to the distortion in estimated posterior probabilities resulting from violations of independence assumption. Isotonic regression (IR) (Zadrozny & Elkan, 2002) is a nonparametric calibration method which produces a monotonically increasing transformation of the probability outcome



Semi-Naive Bayesian Learning. Figure 1. Bayesian Network. (a) one-dependence classifier, (b) SuperParent one-dependence classifier and (c) *z*-dependence classifier ($z \geq 0$)

of naive Bayes. It uses a pair-adjacent violators algorithm (Ayer, Brunk, Ewing, Reid, & Silverman, 1955) to perform calibration. To classify a test instance, IR first finds the interval in which the estimated posterior probability fits and predicts the isotonic regression estimate of this interval as the calibrated posterior probability. Adjusted probability naive Bayesian classification (Webb & Pazzani, 1998) makes adjustments to class probabilities, using a simple hill-climbing search to find adjustments that maximize the ►leave-one-out cross validation accuracy estimate. Starting with the conditional attribute-value frequency table generated by naive Bayes, iterative Bayes (Gama, 2003) iteratively updates the frequency table by cycling through all training instances.

Methods That Introduce Hidden Variables to Naive Bayes

Creating hidden variables or joining attributes is another effective approach to relaxing the attribute independence assumption. Backward sequential elimination and joining (BSEJ) (Pazzani, 1996) extends BSE by creating new Cartesian product attributes. It considers joining each pair of attributes and creates new Cartesian product attributes if the action improves leave-one-out cross validation accuracy. It deletes original attributes and also new Cartesian product attributes during a hill-climbing search. This process of joining or deleting is repeated until there is no further accuracy improvement. Hierarchical naive Bayes (Zhang, Nielsen, & Jensen, 2004) uses conditional mutual information as a criterion to create a hidden variable whose value set is initialized to the Cartesian product over all the value sets of its children. Values of a hidden variable are then collapsed by maximizing conditional log-likelihood via the ►minimum description length principle (Rissanen, 1978).

Selection Between Semi-Naive Bayesian Methods

No algorithm is universally optimal in terms of generalization accuracy. General recommendations for selection between semi-naive Bayesian methods is provided based on ►bias-variance tradeoff together with characteristics of the application to which they are applied.

Error can be decomposed into bias and variance (see ►bias variance decomposition). Bias measures how closely a learner is able to approximate the decision surfaces for a domain and variance measures the sensitivity of a learner to random variations in the training data. Unfortunately, we cannot, in general, minimize bias and variance simultaneously. There is a bias-variance tradeoff such that bias typically decreases when variance increases and vice versa. Data set size usually interacts with bias and variance and in turn affects error. Since differences between samples are expected to decrease with increasing sample size, differences between models formed from those samples are expected to decrease and hence variance is expected to decrease. Therefore, the bias proportion of error may be higher on large data sets than on small data sets and the variance proportion of error may be higher on small data sets than on large data sets. Consequently, low bias algorithms may have advantage in error on large data sets and low variance algorithms may have advantage in error on small data sets (Brain & Webb, 2002).

Zheng & Webb (2005) compare eight semi-naive Bayesian methods with naive Bayes. These methods are BSE, FSS, TAN, SP-TAN, AODE, NBTree, LBR, and BSEJ. NBTree, SP-TAN, and BSEJ have relatively high training time complexity, while LBR has high classification time complexity. BSEJ has very high space complexity. NBTree and BSEJ have very low bias and high variance. Naive Bayes and AODE have very low variance. AODE has a significant advantage in error over other semi-naive Bayesian algorithms tested, with the exceptions of LBR and SP-TAN. It achieves a lower error for more data sets than LBR and SP-TAN without SP-TAN's high training time complexity and LBR's high test time complexity. Subsequent researches (Cerquides & Mántaras, 2005; Zheng & Webb, 2006) show that MAPLMG and SR can in practice significantly improve both classification accuracy and the precision of conditional probability estimates of AODE. However, MAPLMG imposes very high training time overheads on AODE, while SR imposes no extra training time overheads and only modest test time overheads on AODE.

Within the prevailing computational complexity constraints, we suggest using the lowest bias semi-naive Bayesian method for large training data and lowest variance semi-naive Bayesian method for small training data. An appropriate tradeoff between bias and variance should be sought for intermediate size training data. For extremely small data, naive Bayes may be superior and for large data NBTree and BSEJ may be more appealing options if their computational complexity satisfies the computational constraints of the application context. AODE achieves very low variance, relatively low bias and low training time and space complexity. MAPLMG and SR further enhance AODE by substantially reducing bias and error and improving probability prediction with modest time complexity. Consequently, they may prove competitive over a considerable range of classification tasks. Furthermore, MAPLMG may excel if the primary consideration is attaining the highest possible classification accuracy and SR may have an advantage if one wishes efficient classification.

Cross References

►Bayesian Network
►Naive Bayes

Recommended Reading

Ayer, M., Brunk, H. D., Ewing, G. M., Reid, W. T., & Silverman, E. (1955). An empirical distribution function for sampling with incomplete information. *The Annals of Mathematical Statistics, 26*(4), 641–647.

Brain, D., & Webb, G. I. (2002). The need for low bias algorithms in classification learning from large data sets. In *Proceedings of the Sixteenth European Conference on Principles of Data Mining and Knowledge Discovery* (pp. 62–73). Berlin: Springer-Verlag.

Cerquides, J., & Mántaras, R. L. D. (2005). Robust Bayesian linear classifier ensembles. In *Proceedings of the Sixteenth European Conference on Machine Learning*, pp. 70–81.

Frank, E., Hall, M., & Pfahringer, B. (2003). Locally weighted naive Bayes. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*, Acapulco, Mexico (pp. 249–256). San Francisco, CA: Morgan Kaufmann.

Friedman, N., Geiger, D., & Goldszmidt, M. (1997). Bayesian network classifiers. *Machine Learning, 29*(2), 131–163.

Gama, J. (2003). Iterative Bayes. *Theoretical Computer Science, 292*(2), 417–430.

Keogh, E. J., & Pazzani, M. J. (1999). Learning augmented Bayesian classifiers: A comparison of distribution-based and classification-based approaches. In *Proceedings of the International Workshop on Artificial Intelligence and Statistics*, pp. 225–230.

Kittler, J., (1986). Feature selection and extraction. In T. Y. Young & K. S. Fu (Eds.), *Handbook of Pattern Recognition and Image Processing*. New York: Academic Press.

Kohavi, R. (1996). Scaling up the accuracy of naive-Bayes classifiers: A decisiontree hybrid. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pp. 202–207.

Pazzani, M. J. (1996). Constructive induction of Cartesian product attributes. In *ISIS: Information. Statistics and Induction in Science*, Melbourne, Australia, (pp. 66–77). Singapore: World Scientific.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica, 14*, 465–471.

Sahami, M. (1996). Learning limited dependence Bayesian classifiers. In *Proceedings of the Second International Conference on Knowledge Discovery in Databases* (pp. 334–338) Menlo Park: AAAI Press.

Webb, G. I., & Pazzani, M. J. (1998). Adjusted probability naive Bayesian induction. In *Proceedings of the Eleventh Australian Joint Conference on Artificial Intelligence*, Sydney, Australia (pp. 285–295). Berlin: Springer.

Webb, G. I., Boughton, J., & Wang, Z. (2005). Not so naive Bayes: Aggregating onedependence estimators. *Machine Learning, 58*(1), 5–24.

Zadrozny, B., & Elkan, C. (2002). Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the Eighth International Conference on Knowledge Discovery and Data Mining*, Edmonton, Alberta, Canada (pp. 694–699). New York: ACM Press.

Zhang, N. L., Nielsen, T. D., & Jensen, F. V. (2004). Latent variable discovery in classification models. *Artificial Intelligence in Medicine, 30*(3), 283–299.

Zheng, Z., & Webb, G. I. (2000). Lazy learning of Bayesian rules. *Machine Learning, 41*(1), 53–84.

Zheng, F., & Webb, G. I. (2005). A comparative study of semi-naive Bayes methods in classification learning. In *Proceedings of the Fourth Australasian Data Mining Conference*, Sydney, pp. 141–156.

Zheng, F., & Webb, G. I. (2006). Efficient lazy elimination for averaged-one dependence estimators. In *Proceedings of the Twenty-third International Conference on Machine Learning* (pp. 1113–1120). New York: ACM Press.

Semi-Supervised Learning

Xiaojin Zhu
University of Wisconsin-Madison,
Madison, WI, USA

Synonyms

Co-training; Learning from labeled and unlabeled data; Transductive learning

Definition

Semi-supervised learning uses both labeled and unlabeled data to perform an otherwise ►supervised learning or ►unsupervised learning task.

In the former case, there is a distinction between inductive semi-supervised learning and transductive learning. In inductive semi-supervised learning, the learner has both labeled training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^{l} \overset{iid}{\sim} p(\mathbf{x}, y)$ and unlabeled training data $\{\mathbf{x}_i\}_{i=l+1}^{l+u} \overset{iid}{\sim} p(\mathbf{x})$, and learns a predictor $f : \mathcal{X} \mapsto \mathcal{Y}, f \in \mathcal{F}$ where $\mathcal{F}$ is the hypothesis space. Here $\mathbf{x} \in \mathcal{X}$ is an input instance, $y \in \mathcal{Y}$ its target label (discrete for ▶classification or continuous for ▶regression), $p(\mathbf{x}, y)$ the unknown joint distribution and $p(\mathbf{x})$ its marginal, and typically $l \ll u$. The goal is to learn a predictor that predicts future test data better than the predictor learned from the labeled training data alone. In transductive learning, the setting is the same except that one is solely interested in the predictions on the unlabeled training data $\{\mathbf{x}_i\}_{i=l+1}^{l+u}$, without any intention to generalize to future test data.

In the latter case, an unsupervised learning task is enhanced by labeled data. For example, in semi-supervised clustering (a.k.a. ▶constrained clustering) one may have a few must-links (two instances must be in the same cluster) and cannot-links (two instances cannot be in the same cluster) in addition to the unlabeled instances to be clustered; in semi-supervised ▶dimensionality reduction one might have the target low-dimensional coordinates on a few instances.

This entry will focus on the former case of learning a predictor.

Motivation and Background

Semi-supervised learning is initially motivated by its practical value in learning faster, better, and cheaper. In many real world applications, it is relatively easy to acquire a large amount of unlabeled data $\{\mathbf{x}\}$. For example, documents can be crawled from the Web, images can be obtained from surveillance cameras, and speech can be collected from broadcast. However, their corresponding labels $\{y\}$ for the prediction task, such as sentiment orientation, intrusion detection, and phonetic transcript, often requires slow human annotation and expensive laboratory experiments. This labeling bottleneck results in a scarce of labeled data and a surplus of unlabeled data. Therefore, being able to utilize the surplus unlabeled data is desirable.

Recently, semi-supervised learning also finds applications in cognitive psychology as a computational model for human learning. In human categorization and concept forming, the environment provides unsupervised data (e.g., a child watching surrounding objects by herself) in addition to labeled data from a teacher (e.g., Dad points to an object and says "bird!"). There is evidence that human beings can combine labeled and unlabeled data to facilitate learning.

The history of semi-supervised learning goes back to at least the 1970s, when self-training, transduction, and Gaussian mixtures with the expectation-maximization (EM) algorithm first emerged. It enjoyed an explosion of interest since the 1990s, with the development of new algorithms like co-training and transductive support vector machines, new applications in natural language processing and computer vision, and new theoretical analyses. More discussions can be found in section 1.1.3 in Chapelle, Zien, and Schölkopf (2006).

Theory

Unlabeled data $\{\mathbf{x}_i\}_{i=l+1}^{l+u}$ by itself does not carry any information on the mapping $\mathcal{X} \mapsto \mathcal{Y}$. How can it help us learn a better predictor $f : \mathcal{X} \mapsto \mathcal{Y}$? Balcan and Blum pointed out in 2009 that the key lies in an implicit ordering of $f \in \mathcal{F}$ induced by the unlabeled data. Informally, if the implicit ordering happens to rank the target predictor f^* near the top, then one needs less labeled data to learn f^*. This idea will be formalized later on using PAC learning bounds. In other contexts, the implicit ordering is interpreted as a prior over $\mathcal{F}$ or as a regularizer.

A semi-supervised learning method must address two questions: what implicit ordering is induced by the unlabeled data, and how to algorithmically find a predictor near the top of this implicit ordering and fits the labeled data well. Many semi-supervised learning methods have been proposed, with different answers to these two questions (Abney, 2007; Chapelle et al., 2006; Seeger, 2001; Zhu & Goldberg, 2009). It is impossible to enumerate all methods in this entry. Instead, we present a few representative methods.

Generative Models

This semi-supervised learning method assumes the form of joint probability $p(\mathbf{x}, y \mid \theta) = p(y \mid \theta)p(\mathbf{x} \mid y, \theta)$. For example, the class prior distribution $p(y \mid \theta)$ can be a multinomial over $\mathcal{Y}$, while the class conditional distribution $p(\mathbf{x} \mid y, \theta)$ can be a multivariate Gaussian in $\mathcal{X}$ (Castelli & Cover, 1995; Nigam, McCallum, Thrun, & Mitchell, 2000). We use $\theta \in \Theta$ to denote the parameters of the joint probability. Each θ corresponds to a predictor f_θ via Bayes rule:

$$f_\theta(\mathbf{x}) \equiv \operatorname{argmax}_y p(y \mid \mathbf{x}, \theta) = \operatorname{argmax}_y \frac{p(\mathbf{x}, y \mid \theta)}{\sum_{y'} p(\mathbf{x}, y' \mid \theta)}.$$

Therefore, $\mathcal{F} = \{f_\theta : \theta \in \Theta\}$.

What is the implicit ordering of f_θ induced by unlabeled training data $\{\mathbf{x}_i\}_{i=l+1}^{l+u}$? It is the large to small ordering of log likelihood of θ on unlabeled data:

$$\log p(\{\mathbf{x}_i\}_{i=l+1}^{l+u} \mid \theta) = \sum_{i=l+1}^{l+u} \log \left(\sum_{y \in \mathcal{Y}} p(\mathbf{x}_i, y \mid \theta) \right).$$

The top ranked f_θ is the one whose θ (or rather the generative model with parameters θ) best fits the unlabeled data. Therefore, this method assumes that the form of the joint probability is correct for the task.

To identify the f_θ that both fits the labeled data well and ranks high, one maximizes the log likelihood of θ on both labeled and unlabeled data:

$$\operatorname{argmax}_\theta \log p(\{\mathbf{x}_i, y_i\}_{i=1}^{l} \mid \theta) + \lambda \log p(\{\mathbf{x}_i\}_{i=l+1}^{l+u} \mid \theta),$$

where λ is a balancing weight. This is a non-concave problem. A local maximum can be found with the EM algorithm, or other numerical optimization methods. (See also, ►generative learning.)

Semi-Supervised Support Vector Machines

This semi-supervised learning method assumes that the decision boundary $f(\mathbf{x}) = 0$ is situated in a low-density region (in terms of unlabeled data) between the two classes $y \in \{-1, 1\}$ (Joachims, 1999; Vapnik, 1998). Consider the following hat loss function on an unlabeled instance $\mathbf{x}$:

$$\max(1 - |f(\mathbf{x})|, 0),$$

which is positive when $-1 < f(\mathbf{x}) < 1$, and zero outside. The hat loss thus measures the violation in (unlabeled) large margin separation between f and $\mathbf{x}$. Averaging over all unlabeled training instances, it induces an implicit ordering from small to large over $f \in \mathcal{F}$:

$$\frac{1}{u} \sum_{i=l+1}^{l+u} \max(1 - |f(\mathbf{x})|, 0).$$

The top ranked f is one whose decision boundary avoids most unlabeled instances by a large margin.

To find the f that both fits the labeled data well and ranks high, one typically minimizes the following objective:

$$\operatorname{argmin}_f \frac{1}{l} \sum_{i=1}^{l} \max(1 - y_i f(\mathbf{x}_i), 0) + \lambda_1 \|f\|^2 + \lambda_2 \frac{1}{u} \sum_{i=l+1}^{l+u} \max(1 - |f(\mathbf{x})|, 0),$$

which is a combination of the objective for supervised support vector machines, and the average hat loss. Algorithmically, the optimization problem is difficult because the hat loss is non-convex. Existing solutions include semi-definite programming relaxation, deterministic annealing, continuation method, concave-convex procedure (CCCP), stochastic gradient descent, and Branch and Bound. (See also ►support vector machines.)

Graph-Based Models

This semi-supervised learning method assumes that there is a graph $G = \{V, E\}$ such that the vertices V are the labeled and unlabeled training instances, and the undirected edges E connect instances i, j with weight w_{ij} (Belkin, Niyogi, & Sindhwani, 2006; Blum & Chawla, 2001; Zhu, Ghahramani, & Lafferty, 2003). The graph is sometimes assumed to be a random instantiation of an underlying manifold structure that supports $p(\mathbf{x})$. Typically, w_{ij} reflects the proximity of $\mathbf{x}_i, \mathbf{x}_j$. For example, the Gaussian edge weight function defines $w_{ij} = \exp\left(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/\sigma^2\right)$. As another example, the kNN edge weight function defines $w_{ij} = 1$ if $\mathbf{x}_i$ is within the k nearest neighbors of $\mathbf{x}_j$ or vice versa, and $w_{ij} = 0$ otherwise. Other commonly used edge weight functions include ϵ-radius neighbors, b-matching, and combinations of the above.

Large w_{ij} implies a preference for the predictions $f(\mathbf{x}_i)$ and $f(\mathbf{x}_j)$ to be the same. This can be formalized by the graph energy of a function f:

$$\sum_{i,j=1}^{l+u} w_{ij} (f(\mathbf{x}_i) - f(\mathbf{x}_j))^2.$$

The graph energy induces an implicit ordering of $f \in \mathcal{F}$ from small to large. The top ranked function is the smoothest with respect to the graph (in fact, it is any constant function). The graph energy can be equivalently expressed using the so-called unnormalized graph Laplacian matrix. Variants including the normalized Laplacian and the powers of these matrices.

To find the f that both fits the labeled data well and ranks high (i.e., being smooth on the graph or manifold), one typically minimizes the following objective:

$$\operatorname{argmin}_f \frac{1}{l}\sum_{i=1}^{l} c(f(\mathbf{x}_i), y_i) + \lambda_1 \|f\|^2 + \lambda_2 \sum_{i,j=1}^{l+u} w_{ij}(f(\mathbf{x}_i) - f(\mathbf{x}_j))^2,$$

where $c(f(\mathbf{x}), y)$ is a convex loss function such as the hinge loss or the squared loss. This is a convex optimization problem with efficient solvers.

Co-training and Multiview Models

This semi-supervised learning method assumes that there are multiple, different learners trained on the same labeled data, and these learners agree on the unlabeled data. A classic algorithm is co-training (Blum & Mitchell, 1998). Take the example of web page classification, where each web page $\mathbf{x}$ is represented by two subsets of features, or "views" $\mathbf{x} = \langle \mathbf{x}^{(1)}, \mathbf{x}^{(2)} \rangle$. For instance, $\mathbf{x}^{(1)}$ can represent the words on the page itself, and $\mathbf{x}^{(2)}$ the words on the hyperlinks (on other web pages) pointing to this page. The co-training algorithm trains two predictors: $f^{(1)}$ on $\mathbf{x}^{(1)}$ (ignoring the $\mathbf{x}^{(2)}$ portion of the feature) and $f^{(2)}$ on $\mathbf{x}^{(2)}$, both initially from the labeled data. If $f^{(1)}$ confidently predicts the label of an unlabeled instance $\mathbf{x}$, then the instance-label pair $(\mathbf{x}, f^{(1)}(\mathbf{x}))$ is added to $f^{(2)}$'s labeled training data, and vice versa. Note this promotes $f^{(1)}$ and $f^{(2)}$ to predict the same on $\mathbf{x}$. This repeats so that each view teaches the other. Multiview models generalize co-training by utilizing more than two predictors, and relaxing the requirement of having separate views (Sindhwani, Niyogi, & Belkin, 2005). In either case, the final prediction is obtained from a (confidence weighted) average or vote among the predictors.

To define the implicit ordering on the hypothesis space, we need a slight extension. In general, let there be m predictors $f^{(1)}, \ldots, f^{(m)}$. Now let a hypothesis be an m-tuple of predictors $\langle f^{(1)}, \ldots, f^{(m)} \rangle$. The disagreement of a tuple on the unlabeled data can be defined as

$$\sum_{i=l+1}^{l+u} \sum_{u,v=1}^{m} c(f^{(u)}(\mathbf{x}_i), f^{(v)}(\mathbf{x}_i)),$$

where $c()$ is a loss function. Typical choices of $c()$ are the 0–1 loss for classification, and the squared loss for regression. Then the disagreement induces an implicit ordering on tuples from small to large.

It is important for these m predictors to be of diverse types, and have different ▶inductive biases. In general, each predictor $f^{(u)}, u = 1 \ldots m$ may be evaluated by its individual loss function $c^{(u)}$ and regularizer $\Omega^{(u)}$. To find a hypothesis (i.e., m predictors) that fits the labeled data well and ranks high, one can minimize the following objective:

$$\operatorname*{argmin}_{\langle f^{(1)},\ldots,f^{(m)}\rangle} \sum_{u=1}^{m} \left(\frac{1}{l}\sum_{i=1}^{l} c^{(u)}(f^{(u)}(\mathbf{x}_i), y_i) + \lambda_1 \Omega^{(u)}(f^{(u)}) \right) + \lambda_2 \sum_{i=l+1}^{l+u} \sum_{u,v=1}^{m} c(f^{(u)}(\mathbf{x}_i), f^{(v)}(\mathbf{x}_i)).$$

Multiview learning typically optimizes this objective directly. When the loss functions and regularizers are convex, numerical solution is relatively easy to obtain. In the special cases when the loss functions are the squared loss, and the regularizers are squared ℓ_2 norms, there is a closed form solution. On the other hand, the co-training algorithm, as presented earlier, optimizes the objective indirectly with the iterative procedure. One advantage of co-training is that the algorithm is a wrapper method, in that it can use any "blackbox" learners $f^{(1)}$ and $f^{(2)}$ without the need to modify the learners.

A PAC Bound for Semi-Supervised Learning

Previously, we presented several semi-supervised learning methods, each induces an implicit ordering on the hypothesis space using the unlabeled training data, and each attempts to find a hypothesis that fit the labeled training data well as well as rank high in that implicit ordering. We now present a theoretical justification on why this is a good idea. In particular, we present a uniform convergence bound by Balcan and Blum

(Theorem 11 in Balcan and Blum (2009)). Alternative theoretical analyses on semi-supervised learning can be found by following the recommended reading.

First, we introduce some notations. Consider the 0–1 loss for classification. Let $c^* : \mathcal{X} \mapsto \{0,1\}$ be the unknown target function, which may not be in $\mathcal{F}$. Let $\text{err}(f) = E_{\mathbf{x}\sim p}[f(\mathbf{x}) \neq c^*(\mathbf{x})]$ be the true error rate of a hypothesis f, and $\widehat{\text{err}}(f) = \frac{1}{l}\sum_{i=1}^{l} f(\mathbf{x}_i) \neq c^*(\mathbf{x}_i)$ be the empirical error rate of f on the labeled training sample. To characterize the implicit ordering, we defined an "unlabeled error rate" $\text{err}_{\text{unl}}(f) = 1 - E_{\mathbf{x}\sim p}[\chi(f,\mathbf{x})]$, where the *compatibility function* $\chi : \mathcal{F} \times \mathcal{X} \mapsto [0,1]$ measures how "compatible" f is to an unlabeled instance $\mathbf{x}$. As an example, in semi-supervised support vector machines, if $\mathbf{x}$ is far away from the decision boundary produced by f, then $\chi(f,\mathbf{x})$ is large; but if $\mathbf{x}$ is close to the decision boundary, $\chi(f,\mathbf{x})$ is small. In this example, a large $\text{err}_{\text{unl}}(f)$ then means that the decision boundary of f cuts through dense unlabeled data regions, and thus f is undesirable for semi-supervised learning. In contrast, a small $\text{err}_{\text{unl}}(f)$ means that the decision boundary of f lies in a low density gap, which is more desirable. In theory, the implicit ordering on $f \in \mathcal{F}$ is to sort $\text{err}_{\text{unl}}(f)$ from small to large. In practice, we use the empirical unlabeled error rate $\widehat{\text{err}}_{\text{unl}}(f) = 1 - \frac{1}{u}\sum_{i=l+1}^{l+u} \chi(f,\mathbf{x}_i)$.

Our goal is to show that if an $f \in \mathcal{F}$ "fits the labeled data well and ranks high," then f is almost as good as the best hypothesis in $\mathcal{F}$. Let $t \in [0,1]$. We first consider the best hypothesis f_t^* in the subset of $\mathcal{F}$ that consists of hypotheses whose unlabeled error rate is no worse than t: $f_t^* = \text{argmin}_{f' \in \mathcal{F}, \text{err}_{\text{unl}}(f') \leq t} \text{err}(f')$. Obviously, $t = 1$ gives the best hypothesis in the whole $\mathcal{F}$. However, the nature of the guarantee has the form $\text{err}(f) \leq \text{err}(f_t^*) + \text{EstimationError}(t) + c$, where the EstimationError term increases with t. Thus, with $t = 1$ the bound can be loose. On the other hand, if t is close to 0, $\text{EstimationError}(t)$ is small, but $\text{err}(f_t^*)$ can be much worse than $\text{err}(f_{t=1}^*)$. The bound will account for the optimal t.

We introduce a few more definitions. Let $\mathcal{F}(f) = \{f' \in \mathcal{F} : \widehat{err}_{unl}(f') \leq \widehat{err}_{unl}(f)\}$ be the subset of $\mathcal{F}$ with empirical error no worse than that of f. As a complexity measure, let $[\mathcal{F}(f)]$ be the number of different partitions of the first l unlabeled instances $\mathbf{x}_{l+1} \ldots \mathbf{x}_{2l}$, using $f \in \mathcal{F}(f)$. Finally, let $\hat{\epsilon}(f) = \sqrt{\frac{24}{l}\log(8[\mathcal{F}(f)])}$. Then we have the following agnostic bound (meaning that c^* may not be in $\mathcal{F}$, and $\widehat{\text{err}}_{\text{unl}}(f)$ may not be zero for any $f \in \mathcal{F}$):

Theorem 1 *Given l labeled instances and sufficient unlabeled instances, with probability at least $1 - \delta$, the function*

$$f = argmin_{f' \in \mathcal{F}} \widehat{\text{err}}(f') + \hat{\epsilon}(f')$$

satisfies the guarantee that

$$\text{err}(f) \leq \min_t(\text{err}(f_t^*) + \hat{\epsilon}(f_t^*)) + 5\sqrt{\frac{log(8/\delta)}{l}}.$$

If a function f fits the labeled data well, it has a small $\widehat{\text{err}}(f)$. If it ranks high, then $\mathcal{F}(f)$ will be a small set, consequently $\hat{\epsilon}(f)$ is small. The argmin operator identifies the best such function during training. The bound account for the minimum of all possible t tradeoffs. Therefore, we see that the "lucky" case is when the implicit ordering is good such that $f_{t=1}^*$, the best hypothesis in $\mathcal{F}$, is near the top of the ranking. This is when semi-supervised learning is expected to perform well. Balcan and Blum also give results addressing the key issue of how much *unlabeled* data is needed for $\widehat{\text{err}}_{\text{unl}}(f)$ and $\text{err}_{\text{unl}}(f)$ to be close for all $f \in \mathcal{F}$.

Applications

Because the type of semi-supervised learning discussed in this entry has the same goal of creating a predictor as supervised learning, it is applicable to essentially any problems where supervised learning can be applied. For example, semi-supervised learning has been applied to natural language processing (word sense disambiguation (Yarowsky, 1995), document categorization, named entity classification, sentiment analysis, machine translation), computer vision (object recognition, image segmentation), bioinformatics (protein function prediction), and cognitive psychology. Follow the recommended reading for individual papers.

Future Directions

There are several directions to further enhance the value semi-supervised learning. First, we need guarantees that it will outperform supervised learning. Currently, the practitioner has to manually choose a particular semi-supervised learning method, and often manually set learning parameters. Sometimes, a bad choice that does not match the task (e.g., modeling each class with a Gaussian when the data does not have this distribution) can make semi-supervised learning worse than supervised learning. Second, we need methods that benefit from unlabeled when *l*, the size of labeled data, is large. It has been widely observed that the gain over supervised learning is the largest when *l* is small, but diminishes as *l* increases. Third, we need good ways to combine semi-supervised learning and ►active learning. In natural learning systems such as humans, we routinely observe unlabeled input, which often naturally leads to questions. And finally, we need methods that can efficiently process massive unlabeled data, especially in an ►online learning setting.

Cross References

►Active Learning
►Classification
►Constrained Clustering
►Dimensionality Reduction
►Online Learning
►Regression
►Supervised Learning
►Unsupervised Learning

Recommended Reading

Abney, S. (2007). *Semisupervised learning for computational linguistics*. Florida: Chapman & Hall/CRC.

Balcan, M.-F., & Blum, A. (2009). A discriminative model for semi-supervised learning. *Journal of the ACM*.

Belkin, M., Niyogi, P., & Sindhwani, V. (2006). Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research, 7*, 2399–2434.

Blum, A., & Chawla, S. (2001). Learning from labeled and unlabeled data using graph mincuts. In *Proceedings of the 18th international conference on machine learning* (pp. 19–26). San Francisco: Morgan Kaufmann.

Blum, A., & Mitchell, T. (1998). Combining labeled and unlabeled data with co-training. In *COLT: Proceedings of the workshop on computational learning theory* (pp. 92–100). New York: ACM.

Castelli, V., & Cover, T. (1995). The exponential value of labeled samples. *Pattern Recognition Letters, 16*(1), 105–111.

Chapelle, O., Zien, A., & Schölkopf, B., (Eds.) (2006). *Semi-supervised learning*. Cambridge, MA MIT Press.

Joachims, T. (1999). Transductive inference for text classification using support vector machines. In *Proceedings of the 16th international conference on machine learning* (pp. 200–209). San Francisco: Morgan Kaufmann.

Nigam, K., McCallum, A. K., Thrun, S., & Mitchell, T. (2000). Text classification from labeled and unlabeled documents using EM. *Machine Learning, 39*(2/3), 103–134.

Seeger, M. (2001). *Learning with labeled and unlabeled data*. Technical report. University of Edinburgh, Edinburgh.

Sindhwani, V., Niyogi, P., & Belkin, M. (2005). A co-regularized approach to semi-supervised learning with multiple views. In *Proceedings of the 22nd ICML workshop on learning with multiple views*.

Vapnik, V. (1998). *Statistical learning theory*. New York: Wiley.

Yarowsky, D. (1995). Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd annual meeting of the association for computational linguistics* (pp. 189–196).

Zhu, X., Ghahramani, Z., & Lafferty, J. (2003). Semi-supervised learning using Gaussian fields and harmonic functions. In *The 20th international conference on machine learning (ICML)*.

Zhu, X., & Goldberg, A. B. (2009). Synthesis lectures on artificial intelligence and machine learning. In *Introduction to semi-supervised learning*. Morgan & Claypool.

Semi-Supervised Text Processing

Ion Muslea
Language Weaver, Inc.,
Marina del Rey, CA, USA

Synonyms

Learning from labeled and unlabeled data; Transductive learning

Definition

In contrast to supervised and unsupervised learners, which use solely labeled or unlabeled examples, respectively, semi-supervised learning systems exploit both labeled and unlabeled examples. In a typical semi-supervised framework, the system takes as input a (small) training set of labeled examples and a (larger) working set of unlabeled examples; the learner's performance is evaluated on a test set that consists of unlabeled examples. Transductive learning is a particular case of semi-supervised learning in which the working set and the test set are identical.

Semi-supervised learners use the unlabeled examples to improve the performance of the system that could be learned solely from labeled data. Such learners typically exploit – directly or indirectly – the distribution of the available unlabeled examples. Text processing is an ideal application domain for semi-supervised learning because the abundance of text documents available on the Web makes it impossible for humans to label them all. We focus here on two related types of text processing tasks that were heavily studied in the semi-supervised framework: text classification and text ▶Clustering.

Motivation and Background

In most applications of machine learning, collecting large amounts of labeled examples is an expensive, tedious, and error-prone process. In contrast, one may often have cheap or even free access to large amounts of unlabeled examples. For example, for text classification, which is the task of classifying text documents into categories such as politics, sports, entertainment, etc., one can easily crawl the Web and download billions of Web pages; however, manually labeling all these documents according to the taxonomy of interest is an extremely expensive task.

The key idea in semi-supervised learning is to complement a small amount of labeled data by a large number of unlabeled examples. Under certain conditions, the unlabeled examples can be mined for knowledge that will allow the semi-supervised learner to build a system that performs better than one learned solely from the labeled data. More precisely, semi-supervised learners assume that the learning model matches the structure of the application domain. If this is the case, the information extracted from the unlabeled data can be used to *guide* the search towards the optimal solution (e.g., by modifying or re-ranking the learned hypotheses); otherwise, the unlabeled examples may *hurt* rather than help the learning process (Cozman, Cohen, & Cirelo, 2003).

For the sake of concision and clarity, we have had to make several compromises in terms of the algorithms and the applications presented here. Given the vastness of the field of text processing, we have decided to focus only on the two related tasks of text classification and text clustering. They are the most studied text processing applications within the field of machine learning; furthermore, virtually all the main types of semi-supervised algorithms were applied to these two tasks. This decision has two main consequences. First, we do not consider many other text processing tasks, such as information extraction, natural language parsing, or base noun–phrase identification; for these we refer the interested reader to Muslea, Minton, and Knoblock (2006). Second, we discuss and cite approaches that were applied to text classification or clustering there is however, alone an excellent survey by Zhu (2005) covering seminal work on semi-supervised learning that was not applied to text processing.

Structure of the Learning System

Generative Models

The early work on semi-supervised text categorization (Nigam, McCallum, Thrun, & Mitchell, 2000) was based primarily on generative models (see ▶generative learning). Such approaches make two major assumptions: (1) the data is generated by a mixture model, and (2) there is a correspondence between the components of the mixture and the classes of the application domain. Intuitively, if these assumptions hold, the unlabeled examples become instrumental in identifying the mixture's components, while the labeled examples can be used to label each individual component.

The iterative approach proposed by Nigam et al. (2000) is based on ▶The EM Algorithm and works as follows. First, the labeled examples are used to learn an initial classifier, which is used to probabilistically label all unlabeled data; then the newly labeled examples are added to the training set. Finally, a new classifier is learned from all the data, and the entire process is repeated till convergence is reached (or, alternatively, till the number of iterations is fixed).

Nigam et al. (2000) noticed that, in practice, the two above-mentioned assumptions about the generative model may not hold; in order to deal with this problem, the authors propose two extensions of their basic approach. First, they allow each class to be generated by multiple mixture components. Second, they introduce a weighting factor that adjusts the contribution of the unlabeled examples; this factor is tuned during the learning process so that the influence of the unlabeled examples correlates with the degree in which the data distribution is consistent with the mixture model.

The same general framework can also be applied to the related task of text clustering. In the clustering framework, the learner is not concerned with the actual label of an example; instead, it tries to find a partitioning of the examples in clusters that are similar *respect to* a predefined objective function. For example, Seeded-KMeans (Basu, Banerjee, & Mooney, 2002) is a semi-supervised text clustering algorithm that uses the few available labeled examples to seed the search for the data clusters. In order to optimize the target objective function, Seeded-KMeans uses an EM algorithm on a mixture of Gaussians.

Discriminative Approaches

►Support vector machines (SVMs) (Joachims, 1999) are particularly well suited for text classification because of their ability to deal with high-dimensional input spaces (each word in the corpus is a feature) and sparse feature-value vectors (any given document contains only a small fraction of the corpus vocabulary). SVMs are called maximum margin classifiers because they minimize the empirical classification error by maximizing the geometric margin between the domain's positive and negative examples. Intuitively, this is equivalent to finding a discriminative decision boundary that avoids the high-density regions in the instance space.

Transductive SVMs (Joachims, 1999) are designed to find an optimal decision boundary for a particular test set. More precisely, they have access to both the (labeled) training set and the unlabeled test set. Transductive SVMs work by finding a labeling of the test examples that maximizes the margin over all the examples in the training and the test set. This transductive approach has shown significant improvements over the traditional inductive SVMs, especially if the size of the training set is small.

In contrast to transductive SVMs, semi-supervised SVMs (S3VM) work in a true semi-supervised setting in which the test set is not available to the learner. A major difficulty in the S3VM framework is the fact that the resulting optimization problem is not convex, thus being sensitive to the issue of (non-optimal) local minima. CS3VMs (Chapelle, Chi, & Zien, 2006) alleviate this problem by using a global optimization technique called continuation. On binary classification tasks CS3VMs compare favorably against other S3VM approaches, but applying it on multiclass domains is still an open problem.

Multiview Approaches

Multiview learners are a class of algorithms for domains in which the features can be partitioned in disjoint subsets (views), each of which is sufficient to learn the target concept. For example, when classifying Web pages, one can use either the words that appear in the documents or those that appear in the hyper-links pointing to them. Co-training (Blum & Mitchell, 1998) is a semi-supervised, multiview learner that, intuitively, works by bootstrapping the views from each other. First, it uses the labeled examples to learn a classifier in each view. Then it applies the learned classifiers to the unlabeled data and detects the examples on which each view makes the most confident prediction; these examples are labeled by the respective classifiers and added to the (labeled) training set of the other view. The entire process is repeated for a number of iterations.

Multiview learners rely on two main assumptions, namely that the views are compatible and uncorrelated. The former requires that each example is identically labeled by the target concept in each view; the latter means that given an example's label, its description in each view are independent. In practice, both these assumptions are likely to be violated; in order to deal with the first issue, one can use the adaptive view validation algorithm (Muslea, Minton, & Knoblock, 2002b), which predicts whether the views are sufficiently compatible for multiview learning.

With respect to view correlation Muslea, Minton, and Knoblock (2002a) have shown that by interleaving active and semi-supervised learning, multiview approaches become robust the view correlation. A similar idea was previously used in the generative, single-view framework: McCallum and Nigam (1998) have shown that by allowing the algorithm to (smartly) choose which examples to include in the training set, one can significantly improve over the performance of both supervised and semi-supervised learners that used randomly chosen training sets.

The main limitation of multiview learning is the requirement that the user identifies at least two suitable views. In order to cope with this problem, researchers have proposed algorithms that work in a way similar to co-training, but exploit multiple ►inductive biases instead of multiple views. For example, tri-training (Zhou & Li, 2005) uses all domain features to train three supervised classifiers (e.g., a decision tree, a neural network, and a Naive Bayes classifier). These classifiers are then applied to each unlabeled example; if two of them agree on the example's label, they label it accordingly and add it to the third classifier's training set. A degenerate case is represented by *self-training*, which uses a single classifier that repeatedly goes through the unlabeled data and adds to its own training set, the examples on which its predictions are the most confident.

Graph-Based Approaches

The work on graph-based, semi-supervised text learning is based on the idea of representing the labeled and unlabeled examples as vertices in a graph. The edges of this graph are weighted by the pair-wise similarity between the corresponding examples, thus offering a flexible way to incorporate prior domain knowledge. With the learning task encoded in this manner, the problem to be solved becomes one of graph theory, namely finding a partitioning of the graph that agrees with the labeled examples. A major challenge for the graph-based approaches is to find a balanced partitioning of the graph (e.g., in a degenerate scenario, one can propose an unbalanced, undesirable partition in which, except for the negative examples in the training set, all other examples are labeled as positive).

One possible approach to cope with the issue on unbalanced partitions is to use randomized min-cuts (Blum, Lafferty, Rwebangira, & Reddy, 2004). The algorithm starts with the original graph and repeatedly adds random noise to the weights of the edges. Then, for each modified graph, it finds a partitioning by using minimum cuts. Finally, the results from the various runs aggregated in order to create probabilistic labels for the unlabeled examples. This approach has the additional benefit of offering a measure of the confidence in each particular prediction.

The SGT algorithm (Joachims, 2003) uses spectral methods to perform the graph partitioning. SGT can be seen as a transductive version of the k nearest-neighbor classifier; furthermore Joachims (2003) also show that co-training emerges as a special case of SGT. In contrast to transductive SVMs and co-training, SGT does not require additional heuristics for avoiding unbalanced graph partitionings (e.g., in the original co-training algorithm, the examples that are added to the training set after each iteration must respect the domain-dependent ratio of negative-to-positive examples).

LapSVM (Sindhwani, Niyogi, & Belkin, 2005) is a graph-based kernel method that uses a weighted combination a regularizer learned solely from labeled data and a graph Laplacian obtained from both the labeled and unlabeled examples. This approach allows LapSVM to perform a principled search for a decision boundary that is both consistent with the labeled examples and reflects the underlying geometry of all available data points.

Approaches that Exploit Background Knowledge

WHIRL-BG (Zelikovitz & Hirsh, 2000) is an algorithm for classifying short text fragments. It uses an information integration approach that combines three different information sources: the training set, which consists of the labeled examples; the test set that WHIRL-BG must label; and a secondary corpus that consists longer, related documents that are not labeled. Intuitively, WHIRL-BG exploits the secondary corpus as background knowledge that allows the system to link a test example to the most similar labeled training example. In other words, instead of trying to measure directly a (unreliable) similarity between two short strings (i.e., a test and a training example), the system searches for a background document that may include (a large fraction of) both strings.

HMRF-KMEANS (Basu, Bilenko, & Mooney, 2004) unifies the two main approaches to semi-supervised text clustering: the constraint-based one and the adaptive distance one. The former exploits user-provided background knowledge to find an appropriate partitioning of the data; for HMRF-KMEANS, the domain knowledge consists of must-link or cannot-link constraints, which specify whether two examples should or should not have the same label, respectively. The later uses a small number of labeled examples to learn a domain-specific distance measure that is appropriate for the clustering task at hand. HMRF-KMEANS can use any Bregman divergence to measure the clustering distortion, thus supporting a wide variety of learnable distances.

HMRF-KMEANS exploits the labeled examples in three main ways. First, it uses the neighborhoods induced from the constraints to initialize the cluster centroids. Second, when assigning examples to clusters, the algorithm tries to simultaneously minimize both the similarity to the cluster's centroid and the number of violated constraints. Last but not least, during the clustering process, HMRF-KMEANS iteratively re-estimates the distance measure so that it takes into account both the background knowledge and the data variance.

Recommended Reading

Basu, S., Banerjee, A., & Mooney, R. (2002). Semi-supervised clustering by seeding. In *Proceedings of the international conference on machine learning* (pp. 19–26). Sydney, Australia.

Basu, S., Bilenko, M., & Mooney, R. (2004). A probabilistic framework for semi-supervised clustering. In *Proceedings of the ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 59–68). Seattle, WA.

Blum, A., Lafferty, J., Rwebangira, M. R., & Reddy, R. (2004). Semi-supervised learning using randomized mincuts. In *Proceedings of the twenty-first international conference on machine learning* (p. 13).

Blum, A., & Mitchell, T. (1998). Combining labeled and unlabeled data with co-training. In *Proceedings of the 1988 conference on computational learning theory* (pp. 92–100).

Chapelle, O., Chi, M., & Zien, A. (2006). A continuation method for semi-supervised SVMs. In *Proceedings of the 23rd international conference on machine learning* (pp. 185–192). New York: ACM Press.

Cozman, F., Cohen, I., & Cirelo, M. (2003). Semi-supervised learning of mixture models. In *Proceedings of the international conference on machine learning* (pp. 99–106). Washington, DC.

Joachims, T. (1999). Transductive inference for text classification using support vector machines. In *Proceedings of the 16th international conference on machine learning (ICML-99)* (pp. 200–209). San Francisco: Morgan Kaufmann.

Joachims, T. (2003). Transductive learning via spectral graph partitioning. In *Proceedings of the international conference on machine learning*.

McCallum, A., & Nigam, K. (1998). Employing EM in pool-based active learning for text classification. In *Proceedings of the 15th international conference on machine learning* (pp. 359–367).

Muslea, I., Minton, S., & Knoblock, C. (2002a). Active + semi-supervised learning = robust multi-view learning. In *The 19th international conference on machine learning (ICML-2002)* (pp. 435–442). Sydney, Australia.

Muslea, I., Minton, S., & Knoblock, C. (2002b). Adaptive view validation: A first step towards automatic view detection. In *The 19th international conference on machine learning (ICML-2002)* (pp. 443–450). Sydney, Australia.

Muslea, I., Minton, S., & Knoblock, C. (2006). Active learning with multiple views. *Journal of Artificial Intelligence Research, 27*, 203–233.

Nigam, K., McCallum, A. K., Thrun, S., & Mitchell, T. M. (2000). Text classification from labeled and unlabeled documents using EM. *Machine Learning, 39*(2/3), 103–134.

Sindhwani, V., Niyogi, P., & Belkin, M. (2005). Beyond the point cloud: From transductive to semi-supervised learning. In *Proceedings of the 22nd international conference on machine learning* (pp. 824–831). Bonn, Germany.

Zelikovitz, S., & Hirsh, H. (2000). Improving short text classification using unlabeled background knowledge. In *Proceedings of the 17th international conference on machine learning* (pp. 1183–1190).

Zhou, Z.-H., & Li, M. (2005). Tri-training: Exploiting unlabeled data using three classifiers. *IEEE Transactions on Knowledge and Data Engineering, 17*(11), 1529–1541.

Zhu, X. (2005). *Semi-supervised learning literature survey*. Technical report 1530, Department of Computer Sciences, University of Wisconsin, Madison.

Sensitivity

Synonyms

Recall; True positive rate

Sensitivity is the fraction of positive examples predicted correctly by a model. See ►Sensitivity and Specificity, ►Recall and Precision.

Sensitivity and Specificity

Kai Ming Ting
Monash University, Gippsland Campus Churchill, VIC, Australia

Definition

Sensitivity and specificity are two measures used together in some domains to measure the predictive performance of a classification model or a diagnostic test. For example, to measure the effectiveness of a diagnostic test in the medical domain, sensitivity measures the fraction of people with disease (i.e., positive examples) who have a positive test result; and specificity measures the fraction of people without disease (i.e., negative examples) who have a negative test result. They are defined with reference to a special case of the ►confusion matrix, with two classes, one designated

Sensitivity and Specificity. Table 1 The outcomes of classification into positive and negative classes

		Assigned Class	
		Positive	Negative
Actual Class	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

the *positive* class, and the other the *negative* class, as indicated in Table 1.

Sensitivity is sometimes also called *true positive rate*. Specificity is sometimes also called *true negative rate*. They are defined as follows:

Sensitivity = TP/(TP + FN)

Specificity = TP/(TN + FP)

Instead of two measures, they are sometimes combined to provide a single measure of predictive performance as follows:

Sensitivity × Specificity

= TP * TN/[(TP + FN) * (TN + FP)]

Note that sensitivity is equivalent to ►recall.

Cross References

►Confusion Matrix

Sequence Data

►Sequential Data

Sequential Data

Synonyms

Sequence data

Sequential Data refers to any *data* that contain elements that are ordered into sequences. Examples include ►time series, DNA sequences (see ►biomedical informatics) and sequences of user actions. Techniques for learning from sequential data include ►Markov models, ►Conditional Random Fields and ►time series techniques.

Sequential Inductive Transfer

►Cumulative Learning

Sequential Prediction

►Online Learning

Set

►Class

Shannon's Information

If a message announces an event E_1 of probability $P(E_1)$ its information content is $-\log_2 P(E_1)$. This is also its length in bits.

Shattering Coefficient

Synonyms

Growth function

Definition

The shattering coefficient $S_{\mathcal{F}}(n)$ is a function that measures the size of a function class $\mathcal{F}$ when its functions $f : \mathcal{X} \to \mathbb{R}$ are restricted to sets of points $\mathbf{x} = (x_1, \ldots, x_n) \in \mathcal{X}^n$ of size n. Specifically, for each $n \in \mathbb{N}$ the shattering coefficient is the maximum size of the set of vectors $\mathcal{F}_{\mathbf{x}} = \{(f(x_1), \ldots, f(x_n)) : f \in \mathcal{F}\} \subset \mathbb{R}^n$ that can be realized for some choice of $\mathbf{x} \in \mathcal{X}^n$. That is,

$$S_{\mathcal{F}}(n) = \sup_{\mathbf{x} \in \mathcal{X}^n} |\mathcal{F}_{\mathbf{x}}|.$$

The shattering coefficient of a hypothesis class $\mathcal{H}$ is used in ►generalization bounds as an analogue to the class's size in the finite case.

Similarity Measures

Michail Vlachos
IBM Zürich Research Laboratory, Rüschlikon, Switzerland

Synonyms

Distance; Distance metrics; Distance functions; Distance measures

Definition

The term similarity measure refers to a function that is used for comparing objects of any type. The objects can be data structures, database records, or even multimedia objects (audio, video, etc.). Therefore, the input of a similarity measure is two objects and the output is, in general, a number between 0 and 1; "zero" meaning that the objects are completely dissimilar and "one" signifying that the two objects are identical. Similarity is related to distance, which is the inverse of similarity. That is, a similarity of 1 implies a distance of 0 between two objects.

Motivation and Background

Similarity measures are typically used for quantifying the affinity between objects in search operations, where the user presents an object (query) and requests other objects "similar" to the given query. Therefore, a similarity measure is a mathematical abstraction for comparing objects, assigning a single number that indicates the affinity between the said pair of objects. The results of the search are typically presented to the user in the order suggested by the returned similarity value. Objects with higher similarity value are presented first to the user because they are deemed to be more relevant to the query posed by the user. For example, when searching for specific keywords on an Internet search engine, Internet pages that are more relevant/similar to the posed query are presented first. The selection of the proper similarity function is a important parameter in many applications, including ►instance-based learning, ►clustering, and ►anomaly detection.

Most similarity measures attempt to model (imitate) the human notion of similarity between objects. If a similarity function resembles very closely the similarity ranking between objects as returned by a human, then it is considered successful. This is where the difficulty also lies, because in general similarity is something that is very subjective.

Consider the case where a user poses a keyword query 'crane' at a search engine, while searching for images. The returned results would contain images with machineries, birds or even origami creations. This is because when the similarity measure used is solely based on textual information then, then all such images are indeed proper answers to the query. If one was interested also in the semantics of an image, then perhaps additional features such as texture, color or shape could have been utilized. Therefore, for defining an effective similarity measure, one has to first extract the proper object features and then evaluate the similarity using an appropriate distance function.

Classes of Similarity Functions

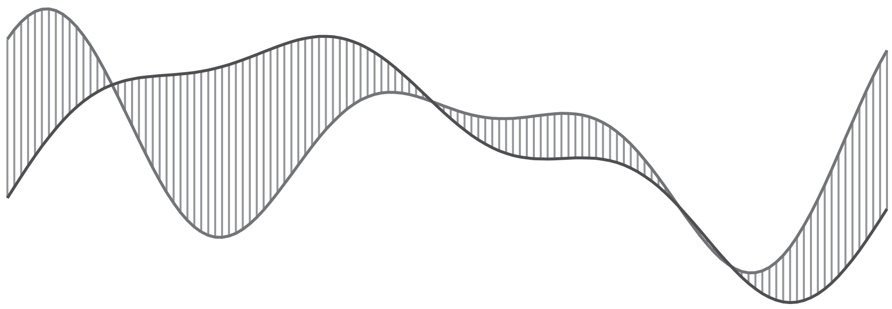
There are two major classes of similarity functions: metric functions and non-metric functions. In order for a function d to be a metric it has to satisfy all the following three properties for any objects X, Y, Z:

1. $d(X,Y) = 0$ iff $X = Y$ (identity axiom)
2. $d(X,Y) = d(Y,X)$ (symmetry axiom)
3. $d(X,Y) + d(Y,Z) \geq d(X,Z)$ (triangle inequality)

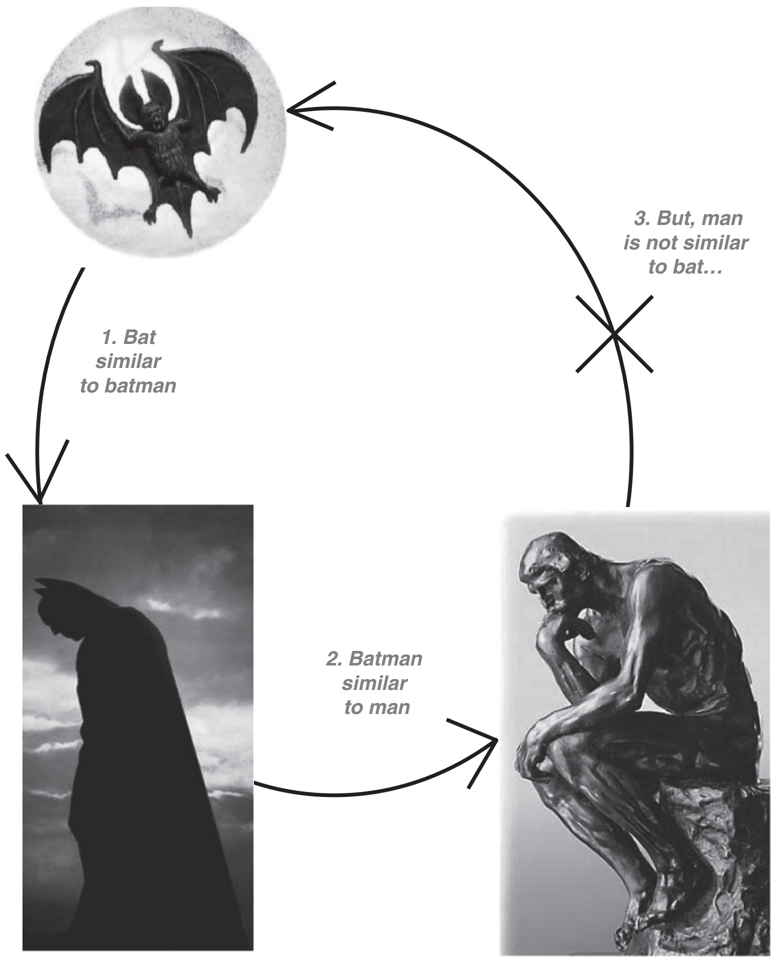
Metric similarity functions are very widely used in search operations because of their support of the triangle inequality. The triangle inequality can help prune a lot of the search space, by eliminating objects from examination that are guaranteed to be distant to the given query (Agrawal et al., 1993; Zezula et al., 2005). The most frequently used metric similarity function is the Euclidean distance. For two objects X and Y that are characterized by set of n features $X = (x_1, x_2, \ldots, x_n)$ and similarly $Y = (y_1, y_2, \ldots, y_n)$ the Euclidean distance is defined as

$$D = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

If we represent the objects X and Y as an ordered sequence of their features, we can visualize the



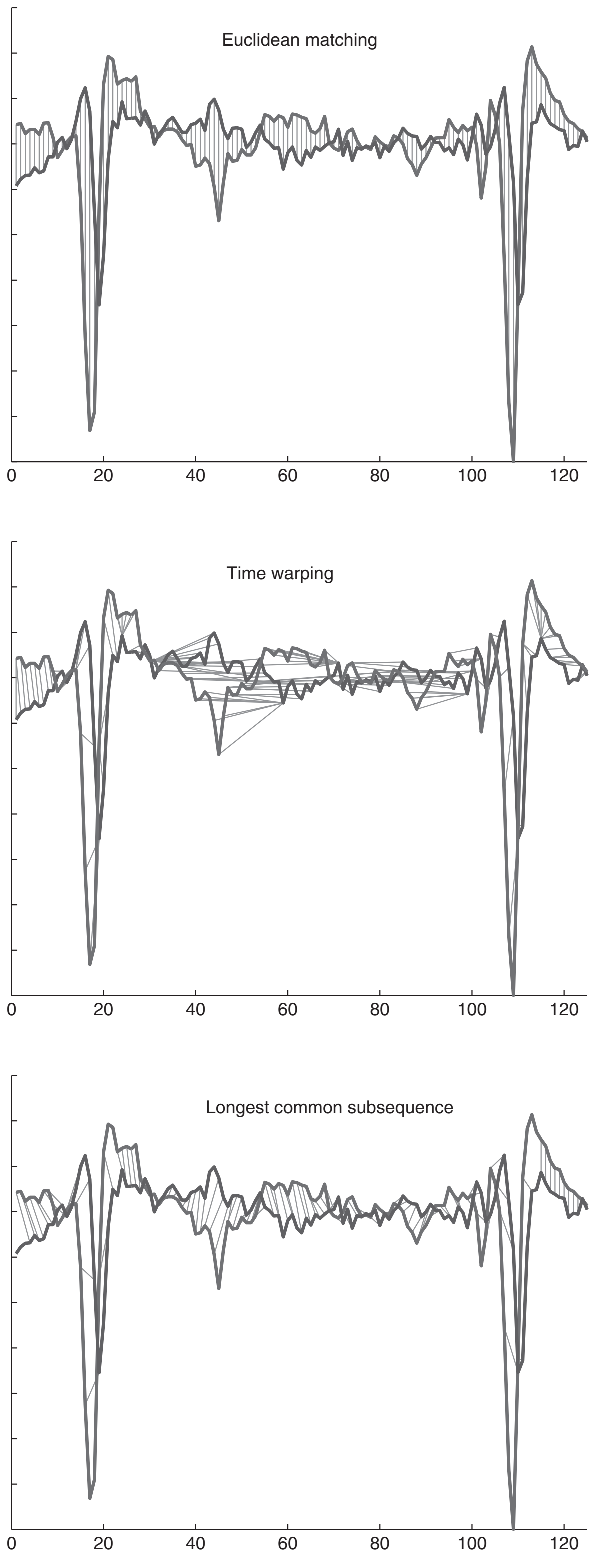
Similarity Measures. Figure 1. Mapping achieved by the Euclidean distance between time-series data



Similarity Measures. Figure 2. Nonmetric similarity that disobeys the triangle inequality

linear mapping achieved by the Euclidean distance in Fig. 1.

Non-metric similarity measures resemble more closely the human notion of similarity by allowing more flexible matching between the examined objects, for example, by allowing non-linear mappings or even by accommodating occlusion of points or features. The human visual system is in general considered nonmetric. Non-metric functions typically disobey the triangle inequality. We can see an example of this below in Fig. 2.



Similarity Measures. Figure 3. Comparison of Euclidean, warping, and longest common subsequence measures

Widely used non-metric similarity functions are the Warping distance and the Longest Common Subsequence (LCSS). The Warping distance (also known as dynamic time warping – DTW) has been very extensively used in the past in voice recognition tasks, due to its ability to perform compression or decompression of the features, allowing flexible non-linear mappings. In Fig. 3 we visually depict the outcome of the previously mentioned measures for ►time-series data. The Euclidean distance performs a rigid linear mapping of points, the DTW can perform nonlinear one-to-many mappings, and the LCSS constructs a one-to-one nonlinear mapping.

Recently, similarity metrics based on information theory, and in specific, on Kolmogorov complexity have been presented (Keogh et al., 2004; Li et al., 2004) and can also be considered as *compression-based* measures. A very simple and easily implementable version of a compression based distance is

$$d_c(X, Y) = \frac{C(XY)}{C(X) + C(Y)}$$

where $C(X)$ is the compressed size (bytes) of X given a certain compression algorithm. The distance will be close to 1, if X and Y are dissimilar and less than 1 when X and Y are related. Therefore, we exploit the fact that if X and Y are "similar" they should compress equally well (approximately same amount of bytes) either when considered separately or together, because the compression dictionaries will be similar when the two objects are related.

In summary, the choice of a similarity metric is highly dependent on the application at hand. The practitioner should also closely consider on which object features the similarity measure will be applied. Ultimately, the combination of both feature selection and similarity metric will define the quality of a search process.

Cross References

►Dimensionality Reduction
►Feature Selection

Recommended Readings

Agrawal, R., Faloutsos, C., & Swami, A. (1993). Efficient similarity search in sequence databases. In *Proceedings of foundations of data organization and algorithms (FODO)*, (pp. 69–84). Chicago, Illinois, USA.

Keogh, E., Lonardi, S., & Ratanamahatana, A. (2004). Towards parameter-free data mining. *Proceedings of International Conference on Knowledge Discovery and Data Mining (SIGKDD)* (pp. 206–215). Seattle, Washington, USA.

Li, M., Chen, X., Li, X., Ma, B., & Vitanyi, P. M. B. (2004). The similarity metric. *IEEE Transactions on Information Theory, 50*(12), 3250–3264.

Zezula, P., Amato, G., Dohnal, V., & Batko, M. (2005). *Similarity search: the metric approach.* Advances in Database Systems, Springer.

Simple Bayes

►Naïve Bayes

Simple Recurrent Network

Risto Miikkulainen
The University of Texas at Austin, Austin, TX, USA

Synonyms

Elman network; Feedforward recurrent network

Definition

The simple recurrent network is a specific version of the ►Backpropagation neural network that makes it possible to process of sequential input and output (Elman, 1990). It is typically a three-layer network where a copy of the hidden layer activations is saved and used (in addition to the actual input) as input to the hidden layer in the next time step. The previous hidden layer is fully connected to the hidden layer. Because the network has no recurrent connections per se (only a copy of the activation values), the entire network (including the weights from the previous hidden layer to the hidden layer) can be trained with the backpropagation algorithm as usual. It can be trained to read a sequence of inputs into a target output pattern, to generate a sequence of outputs from a given input pattern, or to map an input sequence to an output sequence (as in predicting the next input). Simple recurrent networks have been particularly useful in ►time series prediction, as well as in modeling cognitive processes, such as language understanding and production.

Recommended Reading

Elman, J. L. (1990). Finding structure in time. *Cognitive Science, 14*, 179–211.

SMT

►Statistical Machine Translation

Solution Concept

A criterion specifying which locations in the search space are solutions and which are not. In designing a coevolutionary algorithm, it is important to consider whether the solution concept implemented by the algorithm (i.e., the set of individuals to which it can converge) corresponds with the intended solution concept.

Solving Semantic Ambiguity

►Word Sense Disambiguation

SOM

►Self-Organizing Maps

SORT

►Class

Spam Detection

►Text Mining for Spam Filtering

Specialization

Specialization is the converse of ►generalization. Thus, if h_1 is a generalization of h_2 then h_2 is a specialization of h_1.

Cross References

►Generalization
►Induction
►Learning as Search
►Subsumption
►Logic of Generality

Specificity

Synonyms

True negative rate

Specificity is the fraction of negative examples predicted correctly by a model. See ►Sensitivity and Specificity.

Spectral Clustering

►Graph Clustering

Speedup Learning

Alan Fern
Science, Oregon State University,
Corvallis, OR, USA

Definition

Speedup learning is a branch of machine learning that studies learning mechanisms for speeding up problem solvers based on problem-solving experience. The input to a speedup learner typically consists of observations of prior problem-solving experience, which may include traces of the problem solver's operations and/or solutions to solve the problems. The output is knowledge that the problem solver can exploit to find solutions more quickly than before learning without seriously effecting the solution quality. The most distinctive feature of speedup learning, compared with most branches of machine learning, is that the learned knowledge does not provide the problem solver with the ability to solve new problem instances. Rather, the learned knowledge is intended solely to facilitate faster solution times compared to the solver without the knowledge.

Motivation and Background

Much of the work in computer science and especially artificial intelligence aims at developing practically-efficient problem solvers for combinatorially hard problem classes such as automated planning, logical and probabilistic reasoning, game playing, constraint satisfaction, and combinatorial optimization. While it is often straightforward to develop optimal problem solvers for these problems using brute-force, exponential-time search procedures, it is generally much more difficult to develop solvers that are efficient across a wide range of problem instances. The main motivation behind speedup learning is to create adaptive problem solvers that can learn patterns from problem solving experience that can be exploited for efficiency gains. Such adaptive solvers have the potential to significantly outperform traditional static solvers by specializing their behavior to the characteristics of a single problem instance or to an entire class of related problem instances. The exact form of knowledge and learning mechanism is tightly tied to the problem class and the problem-solver architecture.

Most branches of machine learning, such as ►supervised classification, aim to learn fundamentally new problem solving capabilities that are not easily programmed by hand even when ignoring efficiency issues – for example, learning to recognize hand-written digits. Speedup learning is distinct in that it is typically applied in situations where hand-coding an optimal, but inefficient, problem solver is straightforward – for example, solving satisfiability problems. Rather, learning is aimed exclusively at finding solutions in a more practical time frame.

Work in speedup learning grew out of various subfields of artificial intelligence, and more generally computer science. An early example, from automated planning involved learning knowledge for speeding up the original STRIPS planner Fikes, Hart, and Nilsson (1972) via the learning of triangle tables or macros that could later be exploited by the problem solver. Throughout the 1980s and early 1990s, there was a great deal of additional work on speedup learning in the area of automated planning as overviewed in Minton (1993) and Zimmerman and Kambhampati (2003).

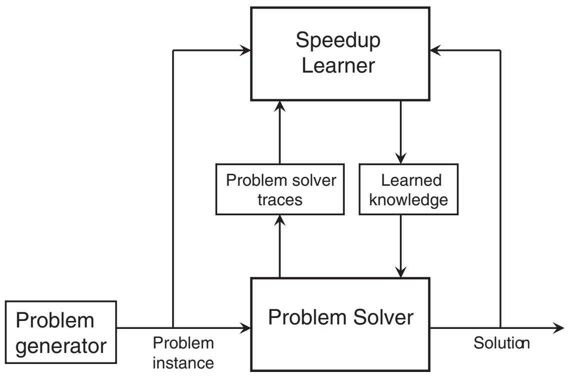
Another major source of speedup learning research has originated from the areas of AI search and constraint satisfaction. Many of the ►intelligent backtracking mechanisms from these areas, which are critical to perform, can be viewed as speedup learning techniques Kambhampati (1998) where knowledge is learned, while solving a problem instance that better informs later search decisions. Such methods have also come out of the area of logic programming Kumar and Lin (1988), where search efficiency plays a central role.

In addition, various branches of AI have developed speedup-learning approaches based on learning improved heuristic evaluation functions. Samuel's checker player Samuel (1959) was one such early examples, where learned evaluation functions allowed for the performance of deep game tree search to be approximated by shallower, a less expensive search.

Structure of Learning System

Figure 1 shows a generic diagram of a speedup learning system. The main components are the problem solver and the speedup learner. The role of the problem solver is to receive problem instances from a problem generator and to produce solutions for those instances. For example, problem solvers might include constraint-satisfaction engines, automated planners, or A^* search. The role of the speedup learner is to produce knowledge that the problem solver can use to improve its solution time. The input to the speedup learner, which is analyzed in order to produce the knowledge, can include one or more of the following data sources: (1) the input problem instances, (2) traces of the problem solver's decisions while solving the input problems, and (3) solutions to solved problems.

Clearly there is a large space of possible speedup learning systems that result from different problem solvers, forms of learned knowledge, learning methods, and intended mode of applicability. Some of the main dimensions are described in the following section along which speedup learning approaches can be characterized. Examples of typical learners that span this space are provided, noting that the examples are far from an exhaustive list.



Speedup Learning. Figure 1. Schematic diagram of a speedup learning system. The problem solver receives problem instances from a problem generator and produces solutions. The speedup learner can observe the input problem instances, traces of the problem solver while solving the problem instances, and sometimes also the solutions to previously solved problem instances. The speedup learner outputs knowledge that can be used by the problem solver to speedup its solution time either on the current problem instance (intra-problem speedup) and/or future related instances (inter-problem speedup)

Dimensions of Speedup Learning

Intra-Problem versus Inter-Problem Speedup. Intra-problem speedup learning is when knowledge is learned during the solution of the current problem instance and is only applicable to speeding up the solution of the current instance. After a solution is found, the knowledge is discarded as it is not applicable for the future instances. Inter-problem speedup learning is when the learned knowledge is applicable not only to the problem(s) it was learned on but also to new problems to be encountered in the future. In this sense, the learned knowledge can be viewed as a generalized knowledge about how to find solutions more quickly for an entire class of problems.

Typically in the inter-problem learning, the problem generator produces instances that are related in some way, and, thus, share common structure that can be learned from the earlier instances and exploited when solving the later instances. Rather intra-problem speedup learners treat each problem instance as completely distinct from the rest. Also note that inter-problem learners have the potential to benefit from the analysis of solutions to previous problem instances. Rather, intra-problem learners are unable to use this source of information, since, once the current problem is solved, no further learning is warranted.

Types of Learned Knowledge. Most problem solvers can be viewed as search procedures, which is the view that will be taken when characterizing the various forms of learned knowledge in speedup learning. Four types of commonly used knowledge are listed below, noting that this is far from an exhaustive list. First, *pruning constraints* are the sets of constraints on search nodes that signal when certain branch of the search space can be safely pruned. Second, *macro operators* (macros) are sequences of search operators that are typically useful when executed in order. Problem solvers can often utilize macros in order to decrease the effective solution depth of the search space by treating macros as additional search operators. It is important that the decrease in effective depth is enough to compensate for the increase in number of operators, which increases the search complexity. Third, *search-control rules* are the sets of rules that typically test the current problem solving state and suggest problem-solving actions such as rejecting, selecting, or preferring a particular search operator. In the extreme case, learned search control rules can completely remove the need for search. Fourth, *heuristic evaluation functions* are used to measure the quality of a particular search node. Learning-improved heuristics can result in better directed search behavior.

Deductive versus Inductive Learning. ▶Deductive learning refers to a learning process for which the learned knowledge can be deductively proven to be correct. For example, in the case of learned pruning constraints, a deductive learning mechanism would provide a guarantee that the pruning was sound in the sense that the optimality of the problem solver would be unaffected. ▶Inductive learning mechanisms rather are statistical in nature and typically do not produce knowledge with associated deductive guarantees. Rather, inductive methods focus on finding statistical regularities that are typically useful, though perhaps not correct in all cases. For example, an inductive learner may discover patterns that are strongly correlated to pruning opportunities, though these patterns may have a small probability of leading to unsound pruning.

In cases where one must guarantee a sound and complete problem solver, deductive learning approaches are always applicable, though their utility depends on the particular application. In certain cases, inductively-learned knowledge can also be utilized in a way that does not effect the correctness of the problem solver. For example, inductively learned search-control rules that assert preferences, rather than prune nodes from the search, do not lead to incompleteness. Traditionally, the primary disadvantage of deductive learning, compared with inductive learning, is that the inductive methods typically produce knowledge that generalizes to a wider range of situations than deductive methods. In addition, deductive learning methods are often more costly in terms of learning time as they rely on expensive deductive reasoning mechanisms. Naturally, a number of speedup learning systems exist that utilize a combination of inductive and deductive learning techniques.

Examples of Intra-Problem Speedup Learning

Much of the speedup learning work arising from research in AI search and constraint satisfaction falls into the intra-problem paradigm. The most common forms of learning are deductive and are based on computing explanations of "search failures" that occur during the solution of a particular problem. Here a search failure typically corresponds to a point where the problem solver must backtrack. By computing and forming such failure explanations the problem solver is typically able to avoid similar types of failures in the future by detecting that a search path will lead to failure without fully exploring that path. ▶Nogood learning is a very successful, and commonly used, example of the general failure-explanation approach Schiex and Verfaillie (1994). Nogoods are combinations of variable values that lead to search failures. By computing and recording nogoods, it is possible to immediately prune search states that consider those value combinations. There are many variations of nogood learning, with different techniques utilizing different approaches to analyzing search failures to extract general nogoods.

Another example of the failure-explanation approach, which is commonly utilized in satisfiability solvers, is ▶clause learning. The idea is similar to nogood learning. When a failure occurs during the systematic search, a proof of the failure is constructed and analyzed to extract implied constraints, or clauses, that the solution must satisfy. These learned clauses are then added to the set of clauses of the original satisfiability problem and in later search trigger early pruning when they, or their consequences, are violated. Efficient implementations of this idea have lead to huge gains in satisfiability solvers. In addition, it has been shown theoretically that clause learning can improve solution times by an exponential factor Beame and Sabharwal (2004).

Inductive techniques for learning heuristic evaluation functions have also been investigated in the intra-problem speedup paradigm. Here we discuss just two such approaches, where in both cases the key idea is to observe the problem solver and extract training examples that can be used to learn an accurate evaluation function. A particularly successful example of this approach is the STAGE system Boyan and Moore (1998) for solving combinatorial optimization problems such as traveling salesman and circuit layout. The problem solving architecture used by STAGE is based on repeated random restarts of a fast hill-climbing local optimizer, which when given an initial configuration of the combinatorial object, performs a greedy search to a local minimum configuration. The speedup learning mechanism for STAGE is to learn an approximate function that maps initial configurations to the performance of the local optimizer when started at that configuration. Note that on each restart of the problem solver the learning component gets a training example that can be used to improve the function. The problem solver uses the learned function in order to select promising configurations from which to restart, rather than choosing randomly. In particular, STAGE attempts to restart from a configuration that optimizes the learned function, which is the predicted best starting point for the hill-climber. This overall approach has shown impressive performance gains in a number of combinatorial optimization domains.

As a second example of inductive learning of heuristics in the intra-problem paradigm, there has been work within the more traditional problem solving paradigm of best-first search Sarkar, Chakrabarti, and Ghose (1998). Here the speedup learner observes the sequence of search nodes traversed by the problem solver. For any pair of nodes observed to be on the same search path, the learner creates a training example in an attempt to train a heuristic to better predict the distance between those two nodes. Ideally, this updated heuristic function better reflects the distance from nodes in the search queue to the goal node of the current problem instance, and, hence, result in improved search performance.

Examples of Inter-Problem Speedup Learning

Much of the work on inter-problem speedup learning came out of AI planning research, where researchers have long studied learning approaches for speeding up planners. speedup in planning is focused in this chapter, noting that similar ideas have also been pursued in other research areas such as constraint satisfaction. For a collection and survey of work on speedup in planning see Minton (1993) and Zimmerman and Kambhampati (2003). Typically in this work, one is interested in learning knowledge for an entire planning domain, which is a collection of problems that share the same set of actions. The Blocksworld is a classic example of such a planning domain. After experiencing and solving a number of problems from a target domain, such as the Blocksworld, the learned knowledge is then used to speed up performance on new problems from the same domain.

There have been a number of deductive learning approaches to speedup learning in planning, which are traditionally cited as ►explanation-based learning (EBL) approaches Minton et al. (1989). EBL for AI planning is strongly related to the failure-explanation approaches developed for CSPs as characterized nicely by Kambhampati (1998). There are two main differences between the inter-problem EBL work in planning and the intra-problem EBL approaches for CSPs. First, EBL approaches in planning produce more general explanations that are applicable not only in the problem in which they were learned, but also new problems. This is often made possible by introducing variables in the place of specific objects into the explanations derived from a particular problem. This allows the explanations to apply to contexts in new problems that share similar structure but involve different objects. The second difference is that inter-problem EBL approaches in planning often produce explanations of successes and not just of failures. These positive explanations are not possible in the context of intra-problem speedup since the intra-problem learner is only interested in solving a single problem.

Despite the relatively large effort invested in inter-problem EBL research, the best approaches typically did not consistently lead to significant gains, and even hurt performance in many cases. A primary way that EBL can hurt performance is by learning too many explanations, which results in the problem solver spending too much time simply evaluating the explanations at the cost of reducing the number of search nodes considered. This problem is commonly referred to as the EBL utility problem Minton (1988) as it is difficult to determine which explanations have high enough utility to be worth keeping.

In addition to EBL, there has also been work on inductive mechanisms for acquiring search-control rules to speedup AI planners. Typically, statistical learning mechanisms are used to find common patterns that can distinguish between good and bad search decisions. As one example, Huang et al. learn action-rejection and selection rules based on the solutions to planning problems from a common domain Huang, Selman, and Kautz (2000). The learned rules were then added as constraints to the constraint satisfaction engine, which served to guide the solver to solution plans more quickly. Another approach, which has been studied at a theoretical and empirical level, is to learn heuristic functions to guide a bounded search process Xu, Fern (2009), in particular, bread-first beam search. Results in a number of planning domains demonstrate significant improvements over planners that do not incorporate a learning component. One other class of approach is based on attempting to learn knowledge that removes the need for a problem solver altogether. In particular, to learn a reactive policy for quickly selecting actions in any given state of the environment. Such policies can be learned via statistical techniques by simply trying to learn an efficient function that maps planning states to the actions selected by the planner. Despite its simplicity, this approach has demonstrated considerable success Khardon (1999) and has also been characterized at a theoretical level Tadepalli and Natarajan (1996).

Cross References

►Explanation-Based Learning

Recommended Reading

Beame, P., Kautz, H., & Sabharwal, A. (2004). Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, *22*, 319–351.

Boyan, J. A., & Moore, A. W. (1998). Learning evaluation functions for global optimization and boolean satisfiability. In *National conference on artificial intelligence* (pp. 3–10). Mlenio Park, CA: AAAI Press.

Fikes, R., Hart, P., & Nilsson, N. (1972). Learning and executing generalized robot plans. *Artificial Intelligence*, *3*(1–3), 251–288.

Huang, Y.-C., Selman, B., & Kautz, H. (2000). Learning declarative control rules for constraint-based planning. In *International conference on machine learning* (pp. 415–422). San Francisco: Morgan Kaufmann.

Kambhampati, S. (1998). On the relations between intelligent backtracking and failure-driven explanation-based learning in constraint satisfaction and planning. *Artificial Intelligence*, *105*(1-2), 161–208.

Khardon, R. (1999). Learning action strategies for planning domains. *Artificial Intelligence*, *113*(1-2), 125–148.

Kumar, V., & Lin, Y. (1988). A data-dependency based intelligent backtracking scheme for prolog. *The Journal of Logic Programming*, *5*(2), 165–181.

Minton, S. (1988). Quantitative results concerning the utility of explanation-based learning. In *National conference on artificial intelligence* (pp. 564–569). St. Paul, MN: Morgan Kaufmann.

Minton, S. (Ed.) (1993). *Machine learning methods for planning*. San Francisco: Morgan Kaufmann.

Minton, S., Carbonell, J., Knoblock, C. A., Kuokka, D. R., Etzioni, O., & Gil, Y. (1989). Explanation-based learning: A problem solving perspective. *Artificial Intelligence*, *40*, 63–118.

Samuel, A. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, *3*(3), 211–229.

Sarkar, S., Chakrabarti, P., & Ghose, S. (1998). Learning whiles solving problems in best first search. *IEEE Transactions on Systems, Man, and Cybernetics–Part A: Systems and Humans*, *28*(4), 553–541.

Schiex, T., & Verfaillie, G. (1994). Nogood recording for static and dynamic constraint satisfaction problems. *International Journal on Artificial Intelligence Tools*, *3*(2), 187–207.

Tadepalli, P., & Natarajan, B. (1996). A formal framework for speedup learning from problems and solutions. *Journal of Artificial Intelligence Research*, *4*, 445–475.

Zimmerman, T., & Kambhampati, S. (2003). Learning-assisted automated planning: Looking back, taking stock, going forward. *AI Magazine*, *24*(2), 73–96.

Speedup Learning For Planning

►Explanation-Based Learning for Planning

Spike-Timing-Dependent Plasticity

A biological form of Hebbian learning where the change of synaptic weights depends on the exact timing of presynaptic and postsynaptic action potentials.

Cross References

►Biological Learning: Synaptic Plasticity
►Hebb Rule
►Spike Timing Dependent Plasticity

Sponsored Search

►Text Mining for Advertising

Squared Error

►Error Squared

Squared Error Loss

►Mean Squared Error

Stacked Generalization

Synonyms

Stacking

Definition

Stacking is an ►ensemble learning technique. A set of models are constructed from bootstrap samples of a dataset, then their outputs on a hold-out dataset are used as *input* to a "meta"-model. The set of base models are called *level*-0, and the meta-model *level*-1. The task of the level-1 model is to combine the set of outputs so as to correctly classify the target, thereby correcting any mistakes made by the level-0 models.

Recommended Reading

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks* 5(2), 241–259.

Stacking

►Stacked Generalization

Starting Clause

►Bottom Clause

State

In a ►*Markov decision process*, *states* represent the possible system configurations facing the decision-maker at each *decision epoch*. They must contain all variable information relevant to the decision-making process.

Statistical Learning

►Inductive Learning

Statistical Machine Translation

Miles Osborne
University of Edinburgh, Edinburgh, UK

Synonyms

SMT

Definition

Statistical machine translation (SMT) deals with automatically mapping sentences in one human language (for example, French) into another human language (such as English). The first language is called the *source* and the second language is called the *target*. This process can be thought of as a stochastic process. There are many SMT variants, depending upon how translation is modeled. Some approaches are in terms of a string-to-string mapping, some use trees-to-strings, and some use tree-to-tree models. All share in common the central idea that translation is automatic, with models estimated from parallel corpora (source-target pairs) and also from monolingual corpora (examples of target sentences).

Motivation and Background

Machine Translation has widespread commercial, military, and political applications. For example, increasingly, the Web is accessed by non-English speakers reading non-English pages. The ability to find relevant information clearly should not be bounded by our language-speaking capabilities. Furthermore, we may not have sufficient linguists in some language of interest to cope with the sheer volume of documents that we would like translated. Enter automatic translation. Machine translation poses a number of interesting machine learning challenges: data sets are typically very large, as are the associated models; the training material used is often noisy and plagued with sparse statistics; the search space of possible translations is sufficiently large that exhaustive search is not possible. Advances in machine learning, such as maximum-margin methods, frequently appear in translation research. SMT systems are now sufficiently mature that they can be deployed in production systems. A good example of this is Google's online Arabic-English translation, which is based upon SMT techniques.

Structure of the Learning System

Modeling

Formally, translation can be described as finding the most likely target sentence e^* for some source sentence f:

$$e^* = \text{argmax}_e P(f \mid e)P(e)$$

(e conventionally stands for English and f for French, but any language pairs can be substituted.)

This approach has three major aspects:

- A translation model ($P(f \mid e)$), which specifies the set of possible translations for some target sentence. The translation model also assigns probabilities to these translations, representing their relative correctness.
- A ▸language model ($P(e)$), which models the fluency of the proposed target sentence. This assigns a distribution over strings, with higher probabilities being assigned to sentences which are more representative of natural language. Language models are usually smoothed n-gram models, typically conditioning on two (or more) previous words when predicting the probability of the current word.
- A search process (the argmax operation), which is concerned with navigating through the space of possible target translations. This is called *decoding*. Decoding for SMT is NP-hard, so most approaches use a beam search.

This is called the *Source-Channel* approach to translation (Brown, Pietra, Pietra, & Mercer, 1994). Most modern SMT systems instead use a ▸log-linear model, as it is more flexible and allows for various aspects of translation to be balanced together (Och & Ney, 2001):

$$e^* = \text{argmax}_e \left(\sum_i f_i(e,f)\lambda_i \right)$$

Here, feature functions $f_i(e,f)$ capture some aspect of translation and each feature function has an associated weight λ_i. When we have the two feature functions $P(f \mid e)$ and $P(e)$, we have the Source-Channel model. The weights are scaling factors (balancing the contributions that each feature function makes) and are optimized with respect to some ▸loss function which evaluates translation quality. Frequently, this is in terms of the *BLEU* evaluation metric Papineni, Roukos, Ward, & Zhu (2001). Typically, the error surface is nonconvex and the loss function is nondifferentiable, so search techniques which do not use first-order derivatives must be employed. It is worth noting that machine translation evaluation is a complex problem and that methods such as BLEU are not without criticism.

SMT systems usually decompose entire sentences into a sequence of strings called *phrases* (Koehn, Och, & Marcu, 2003). The modeling task then becomes one of determining how to break a source sentence into a sequence of contiguous phrases and how to specify which source phrase should be associated with each target phrase. Figure 1 shows an example English-French sentence pair. Figure 2 shows that sentence pair decomposed into phrase-pairs. Phrase-based systems

Those people have grown up, lived and worked for many years in a farming district.
Ces gens ont grandi, vécu et oeuvré des dizaines d'années dans le domain agricole.

Statistical Machine Translation. Figure 1. A sentence pair

Ces gens ont	Those people have
gens ont grandi	people have grown up
ont grandi ,	have grown up ,
grandi , vécu	grown up , lived
, vécu et	, lived and
vécu et oeuvré	lived and worked
et oeuvré des dizaines d' oeuvré	and worked many
oeuvré des dizaines d' années dizaines	worked many years
des dizaines d' années dans	many years in
années dans le	years in a
le domaine agricole	a farming districtle
domaine agricole .	farming district .

Statistical Machine Translation. Figure 2. Example phrase pairs

represented an advance over previous word-based models, since phrase-based translation can capture local (within a phrase) word order. Furthermore, phrase-based translation approaches need to make fewer decisions than word-based models. This means there are fewer errors to make.

A major aspect of any SMT approach is dealing with phrasal *reordering*. Typically, the translation of each source phrase need not follow the same temporal order in the target sentence. Simple approaches model the absolute distance a target phrase can "move" from the originating target phrase. More sophisticated reordering models condition this movement upon the aspects of the phrase pair.

Our description of SMT is in terms of a string-to-string model. There are numerous other SMT approaches, for example those which use notions of syntax (Chiang, 2005). These models are now showing promising results, but are significantly more complex to describe.

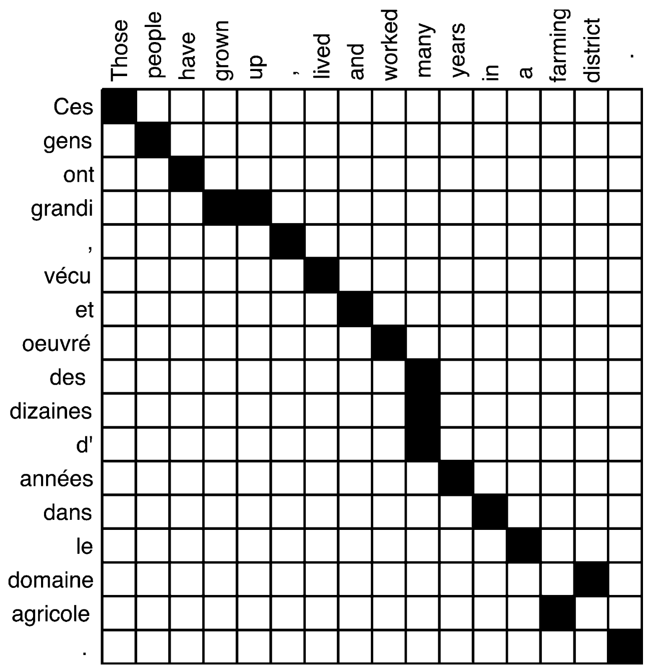
Estimation

The translation model of a SMT system is estimated using *parallel corpora*. Because the search space is so large and that parallel corpora is not aligned at the word level, the estimation process is based upon a large-scale application of Expectation-Maximization, along with heuristics. This consists of the following steps:

- Determine how each source word translates to zero or more target words. The IBM models are used for this task, which are based upon the Expectation-Maximization algorithm for parameter estimation (Brown et al., 1994).
- Repeat this process, but instead determine how each target word translates to zero or more source words.
- Harmonize the previous two steps, creating a set of *word alignments* for each sentence pair. This process is designed to use the two directions as alternative views on how words should be translated. Figure 3 shows the sentence pair aligned at the word level.
- Heuristically, determine which sequence of source words translates to a sequence of target words. This produces a set of *phrase-pairs*: a snippet of text in the source sentence and the associated snippet of text in the target sentence.
- Relative frequency estimators can then be used to characterize how each source phrase translates to a given target phrase.

Parallel corpora varies in size tremendously; for language pairs such as Arabic to English, we have on the order of ten million sentence pairs. Most other language pairs (for example, Finnish to Irish) will have far smaller parallel corpora available. Parallel corpora exists for all European languages and for many other pairs, such as Mandarin to English.

The language model is instead estimated from monolingual corpora, typically using relative frequency estimates, which are then smoothed. For languages such as English, typically billion (and more) words are used. Deploying such large models can pose significant engineering challenges. This is because the language model can easily be so large that it will not fit into the memory



Statistical Machine Translation. Figure 3. The sentence pair in Fig. 1 aligned at the word-level

of conventional machines. Also, the language model can be queried millions of times when translating sentences, which precludes storing it on disk.

Programs and Data

All of the code and data necessary to begin work on SMT is available either as public source, or for a small payment (in the case of corpora from the LDC):

- The standard software to estimate word-based translation models is Giza++:
  http://www.fjoch.com/GIZA++.html
- Converting word-based to phrase-based models and decoding can be achieved using the Moses decoder and associated sets of scripts:
  http://www.statmt.org/jhuws/?n=Moses.HomePage
- Translation performance can be evaluated using BLEU:
  http://www.nist.gov/speech/tests/mt/resources/scoring.htm
- The SRILM is the standard toolkit for building and using language models:
  http://www.speech.sri.com/projects/srilm/
- Europarl is a set of parallel corpora, dealing with European languages:
  http://www.statmt.org/europarl/
- The Linguistics Data Consortium (LDC) maintains corpora of various kinds, including large volumes of monolingual data which can be used to train language models:
  http://www.ldc.upenn.edu/

Recommended Reading

Brown, P. F., Pietra, S. D., Pietra, V. J. D., & Mercer, R. L. (1994). The mathematic of statistical machine translation: Parameter estimation. *Computational Linguistics*, *19*(2), 263–311.

Chiang, D. (2005, June). A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd annual meeting of the association for computational linguistics (ACL'05)* (pp. 263–270). Ann Arbor, MI: Association for Computational Linguistics.

Koehn, P., Och, F. J., & Marcu, D. (2003). Statistical phrase-based translation. In *NAACL '03: Proceedings of the 2003 conference of the north american chapter of the association for computational linguistics on human language technology* (pp. 48–54). Morristown, NJ: Association for Computational Linguistics.

Och, F. J., & Ney, H. (2001). Discriminative training and maximum entropy models for statistical machine translation. In *ACL '02: Proceedings of the 40th annual meeting on association for computational linguistics* (pp. 295–302). Morristown, NJ: Association for Computational Linguistics.

Papineni, K., Roukos, S., Ward, T., & Zhu, W. -J. (2001). Bleu: A method for automatic evaluation of machine translation. In *ACL '02: Proceedings of the 40th annual meeting on association for computational linguistics* (pp. 311–318). Morristown, NJ: Association for Computational Linguistics.

Statistical Natural Language Processing

►Maximum Entropy Models for Natural Language Processing

Statistical Physics Of Learning

►Phase Transitions in Machine Learning

Statistical Relational Learning

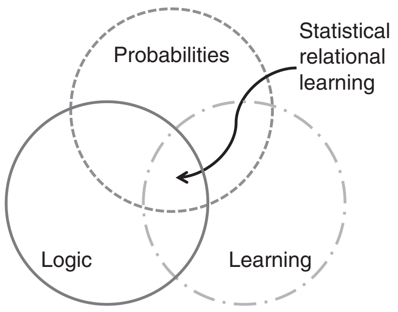
Luc De Raedt[1], Kristian Kersting[2]
[1]Katholieke Universiteit Leuven,
Heverlee, Belgium
[2]Knowledge Discovery, Fraunhofer IAIS,
Sankt Augustin, Germany

Definition

Statistical relational learning a.k.a. probabilistic inductive logic programming deals with machine learning and data mining in relational domains where observations may be missing, partially observed, or noisy. In doing so, it addresses one of the central questions of artificial intelligence – the integration of probabilistic reasoning with machine learning and first-order and relational representations – and deals with all related aspects such as reasoning, parameter estimation, and structure learning.

Motivation and Background

One of the central questions of artificial intelligence is concerned with combining expressive knowledge representation formalisms such as relational and first-order logic with principled probabilistic and statistical approaches to inference and learning. While traditionally relational and logical representations, probabilistic and statistical reasoning, and machine learning have been studied independently of one another, statistical relational learning investigates them jointly, cf. Fig. 1. A major driving force is the explosive growth in the amount of heterogeneous data that is being collected in the business and scientific world in domains such as bioinformatics, transportation systems, communication networks, social network analysis, citation analysis, and robotics. Characteristic for these domains is that they provide *uncertain* information about varying numbers of entities and relationships among the entities, that is, about *relational* domains. Traditional machine learning approaches are able to cope either with uncertainty or with relational representations but typically not with both.



Statistical Relational Learning. Figure 1. Statistical relational learning a.k.a. probabilistic inductive logic programming combines probability, logic, and learning

Many formalisms and representations have been developed in statistical relational learning. For instance, Eisele (1994) has introduced a probabilistic variant of comprehensive unification formalism (CUF). In a similar manner, Muggleton (1996) and Cussens (1999) have upgraded stochastic grammars toward *stochastic logic programs*. Sato (1995) has introduced *probabilistic distributional semantics* for logic programs. Taskar, Abbeel, and Koller (2002) have upgraded Markov networks toward *relational Markov networks*, and Richardson and Domingos (2006) toward *Markov logic networks*. Neville and Jensen (2004) have extended dependency networks toward *relational dependency networks*. Another research stream has investigated logical and relational extensions of Bayesian networks. It includes Poole's *independent choice logic* (Poole, 1993), Ngo and Haddawy's *probabilistic logic programs* (Ngo & Haddawy, 1997), Jäger's *relational Bayesian networks* (Jager, 1997), Koller, Getoor, and Pfeffer's *probabilistic relational models* (Getoor, 2001; Pfeffer 2000), and Kersting and De Raedt's *Bayesian logic programs* (Kersting & De Raedt, 2007).

The benefits of employing logical abstraction and relations within statistical learning are manyfold:

1. Relations among entities allow one to use information about one entity to help reach conclusions about other, related entities.
2. Variables, that is, placeholders for entities allow one to make abstraction of specific entities.
3. Unification allows one to share information among entities. Thus, instead of learning regularities for each single entity independently, statistical relational learning aims at finding general regularities among groups of entities.
4. The learned knowledge is often declarative and compact, which makes it easier for people to understand and to validate.
5. In many applications, there is a rich background theory available, which can efficiently and elegantly be represented as a set of general regularities. This is important because background knowledge may improve the quality of learning as it focuses the learning on the relevant patterns, that is, it restricts the search space.
6. When learning a model from data, relational and logical abstraction allow one to reuse experience in that *learning about one entity improves the prediction for other entities*; and this may even generalize to objects that have never been observed before.

Thus, relational and logical abstraction make statistical learning more robust and efficient. This has proven to be beneficial in many fascinating real-world applications in citation analysis, web mining, natural language processing, robotics, bio- and chemo-informatics, electronic games, and activity recognition.

Theory

Whereas most of the existing works on statistical relational learning have started from a statistical and probabilistic learning perspective and extended probabilistic formalisms with relational aspects, statistical relational learning can elegantly be introduced by starting from ►inductive logic programming (De Raedt, 2008; Muggleton & De Raedt, 1994), which is often also called *multi-relational data mining* (MRDM) (Džeroski & Lavrač, 2001). Inductive logic programming is a research field at the intersection of machine learning and logic programming. It forms a formal framework and has introduced practical algorithms for inductively learning relational descriptions (in the form of logic programs) from examples and background knowledge. So, the only difference to statistical relational learning is that it does not explicitly deal with uncertainty.

Essentially, there are only two changes to apply to inductive logic programming approaches in order to arrive at statistical relational learning:

1. ►clauses (i.e., logical formulae that can be interpreted as rules; cf. below) are annotated with probabilistic information such as conditional probabilities; and
2. the ►covers relation (which states the conditions under which a hypothesis considers an example as positive) becomes probabilistic.

A probabilistic covers relation softens the hard covers relation employed in traditional inductive logic programming and is defined as the probability of an example given the hypothesis and the background theory.

Definition 1 (Probabilistic Covers Relation). *A probabilistic covers relation takes as arguments an example e, a hypothesis H and possibly the background theory B, and returns the probability value* $\mathbf{P}(e \mid H, B)$ *between* 0 *and* 1 *of the example e given H and B, that is, covers*$(e, H, B) = \mathbf{P}(e \mid H, B)$.

It specifies the likelihood of the example given the hypothesis and the background theory. Different choices of the probabilistic covers relation lead to different statistical relational learning approaches; this is akin to the learning settings in inductive logic programming.

Statistical Relational Languages

There is a multitude of different languages and formalisms for statistical relational learning. For an overview of these languages we refer to (Getoor & Taskar, 2007) and (De Raedt, Frasconi, Kersting, & Muggleton, 2008). Here, we choose two formalisms that are representatives of the two main streams in statistical relational learning. First, we discuss Markov logic (Richardson & Domingos, 2006), which upgrades Markov network toward first-order logic, and second, we discuss ProbLog (De Raedt, Kimmig, & Toivonen, 2007), which is a probabilistic Prolog based on Sato's distribution semantics (Sato, 1995). While Markov logic is a typical example of knowledge-based model construction, ProbLog is a probabilistic programming language.

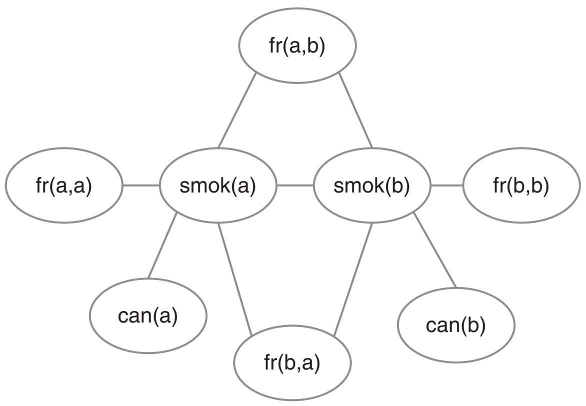
Case Study: Markov Logic Networks Markov logic combines first-order logic with ▶Markov networks. The idea is to view logical formulae as soft constraints on the set of possible worlds, that is, on the ▶interpretations (an interpretation is a set of facts). If an interpretation does not satisfy a logical formula, it becomes less probable, but not necessarily impossible as in traditional logic. Hence, the more formulae an interpretation satisfies, the more likely it becomes. In a Markov logic network, this is realized by associating a weight to each formula that reflects how strong the constraint is. More precisely, a Markov logic network consists of a set of weighted clauses $H = \{c_1, \ldots, c_m\}$. (Markov logic networks, in principle, also allow one to use arbitrary logical formulae, not just clauses. However, for reasons of simplicity, we only employ clauses and make some further simplifications.) The weights w_i of the clauses then specify the strength of the clausal constraint.

Example 1 *Consider the following example (adapted from Richardson and Domingos (2006)). Friends & Smokers is a small Markov logic network that computes the probability of a person having lung cancer on the basis of her friends smoking. This can be encoded using the following weighted clauses:*

1.5: cancer(P) ← smoking(P)
1.1: smoking(X) ← friends(X,Y), smoking(Y)
1.1: smoking(Y) ← friends(X,Y), smoking(X)

The first clause states the soft constraint that smoking causes cancer. So, interpretations in which persons that smoke have cancer are more likely than those where they do not (under the assumptions that other properties remain constant). The second and third clauses state that friends of smokers are typically also smokers.

A Markov logic network together with a Herbrand domain (in the form of a set of constants $\{d_1, \ldots, d_k\}$) then induces a grounded Markov network, which defines a probability distribution over the possible Herbrand interpretations.



Statistical Relational Learning. Figure 2. The Markov network for the constants ann and bob. Adapted from Richardson and Domingos (2006)

The nodes, that is, the random variables in the grounded network, are the atoms in the Herbrand base, that is, the facts of the form $p(d'_1, \ldots, d'_n)$ where p is a predicate or relation and the d'_i are constants. Furthermore, for every ground instance $c_i\theta$ of a clause c_i in H, there will be an edge between any pair of atoms $a\theta, b\theta$ that occurs in $c_i\theta$. The Markov network obtained for the constants anna and bob is shown in Fig. 2. To obtain a probability distribution over the Herbrand interpretations, we still need to define the potentials. The probability distribution over interpretations I is

$$\mathbf{P}(I) = \frac{1}{Z} \prod_{c:clause} \mathbf{f}_c(I) \qquad (1)$$

where the f_c are defined as

$$f_c(I) = e^{n_c(I) w_c} \qquad (2)$$

and $n_c(I)$ denotes the number of substitutions θ for which $c\theta$ is satisfied by I, and Z is a normalization constant. The definition of a potential as an exponential function of a weighted feature of a clique is common in Markov networks; cf. ▶graphical models. The reason is that the resulting probability distribution is easier to manipulate.

Note that for different (Herbrand) domains, different Markov networks will be produced. Therefore, one can view Markov logic networks as a kind of template for generating Markov networks, and, hence, Markov logic is based on knowledge-based model construction. Notice also that Markov logic networks define a probability distribution over interpretations, and nicely separate the qualitative from the quantitative component.

Case Study: ProbLog Many formalisms do not explicitly encode a set of conditional independency assumptions, as in Bayesian or Markov networks, but rather extend a (logic) programming language with probabilistic choices. Stochastic logic programs (Cussens, 2001; Muggleton, 1996) directly upgrade stochastic context-free grammars toward definite clause logic, whereas PRISM (Sato, 1995), probabilistic Horn abduction (PHA) (Poole, 1993), and the more recent independent choice logic (ICL) (Poole, 1997) specify probabilities on facts from which further knowledge can be deduced. As a simple representative of this stream of work, we introduce the probabilistic Prolog called ProbLog (De Raedt et al., 2007).

The key idea underlying Problog is that some facts f for *probabilistic* predicates are annotated with a probability value. This value indicates the degree of belief, that is the probability, that any ground instance $f\theta$ of f is true. It is also assumed that the $f\theta$ are marginally independent. The probabilistic facts are then augmented with a set of definite clauses defining further predicates (which should be disjoint from the probabilistic ones). An example adapted from De Raedt et al. (2007) is given below.

Example 2 *Consider the facts*

```
0.9: edge(a,c) ←
0.7: edge(c,b) ←
0.6: edge(d,c) ←
0.9: edge(d,b) ←
```

which specify that with probability 0.9 *there is an edge from* a *to* c. *Consider also the following (simplified) definition of* path/2.

```
path(X,Y)edge(X,Y) ←
path(X,Y)edge(X,Z), path(Z,Y) ←
```

One can now define a probability distribution on (ground) proofs as follows. The probability of a ground proof is the product of the probabilities of the (ground) clauses (here, facts) used in the proof. For instance, the only proof for the goal ← path(a,b) employs the facts edge(a,c) and edge(c,b); these facts are marginally independent, and hence the probability of the proof is 0.9×0.7. The probabilistic facts used in a single proof are sometimes called an *explanation*.

It is now tempting to define the probability of a ground atom as the sum of the probabilities of the proofs for that atom. However, this does not work without additional restrictions, as shown in the following example.

Example 3 *The fact* path(d,b) *has two explanations:*

1. {edge(d,c), edge(c,b)} *with probability* $0.6 \times 0.7 = 0.42$, *and*
2. {edge(d,b)} *with probability* 0.9.

Summing the probabilities of these explanations gives a value of 1.32, which is clearly impossible.

The reason for this problem is that the different explanations are not mutually exclusive, and therefore their probabilities may not be summed. The probability $P(\text{path(d,b)} = true)$ is, however, equal to the probability that *a* proof succeeds, that is,

$$P(\text{path(d,b)} = true) = P[(\text{e(d,c)} \wedge \text{e(c,b)}) \vee \text{e(d,b)}]$$

which shows that computing the probability of a derived ground fact reduces to computing the probability of a boolean formula in disjunctive normal form (DNF), where all random variables are marginally independent of one another. Computing the probability of such formulae is an NP-hard problem, the *disjoint-sum* problem. Using the *inclusion-exclusion* principle from set theory, one can compute the probability as

$$\begin{aligned} P(\text{path(d,b)} = true) &= P[(\text{e(d,c)} \wedge \text{e(c,b)}) \vee \text{e(d,b)}] \\ &= P(\text{e(d,c)} \wedge \text{e(c,b)}) \\ &\quad + P(\text{e(d,b)}) \\ &\quad - P((\text{e(d,c)} \wedge \text{e(c,b)}) \\ &\quad \wedge \text{e(d,b)}) \\ &= 0.6 \times 0.7 + 0.9 - 0.6 \times 0.7 \\ &\quad \times 0.9 = 0.942 \end{aligned}$$

S

There exist more effective ways to compute the probability of such DNF formulae (De Raedt et al., 2007), where binary decision diagrams are employed to represent the DNF formulae.

The above example shows how the probability of a specific fact is defined and can be computed. The distribution at the level of individual facts (or goals) can easily be generalized to a possible world semantics, specifying a probability distribution on interpretations. It is formalized in the *distribution semantics* of Sato (1995), which is defined by starting from the set of all probabilistic ground facts F for the given program. For simplicity, we shall assume that this set is finite, though Sato's results also hold for the infinite case. The distribution semantics then starts from a probability distribution $P_F(S)$ defined on subsets $S \subseteq F$:

$$P_F(S) = \prod_{f \in S} P(f) \prod_{f \notin S} (1 - P(f)). \quad (3)$$

Each subset S is now interpreted as a set of logical facts and combined with the definite clause program R that specifies the logical part of the probabilistic logic program. Any such combination $S \cup R$ possesses a unique least Herbrand model $M(S \cup R)$, which corresponds to a possible world. The probability of such a possible world is then the sum of the probabilities of the subsets S yielding that possible world, that is,

$$P_W(M) = \sum_{S \subseteq F : M(S \cup R) = M} P_F(S) \quad (4)$$

For instance, in the path example, there are 16 possible worlds, which can be obtained from the 16 different truth assignments to the facts, and whose probabilities can be computed using Eq. (4). As for graphical models, the probability of any logical formula can be computed from a possible world semantics (specified here by P_W).

Because computing the probability of a fact or goal under the distribution semantics is hard, systems such as Prism (Sato, 1995) and PHA (Poole, 1993) impose additional restrictions that can be used to improve the efficiency of the inference procedure. The key assumption is that the explanations for a goal are *mutually exclusive*, which overcomes the disjoint-sum problem. If the different explanations of a goal do not overlap, then its probability is simply the sum of the probabilities of its explanations. This directly follows from the inclusion-exclusion formulae as under the exclusive-explanation assumption the conjunctions (or intersections) are empty.

Learning

Essentially, any statistical relational approach can be viewed as lifting a traditional inductive logic programming setting by associating probabilistic information to clauses and by replacing the deterministic coverage relation by a probabilistic one. In contrast to traditional graphical models such as Bayesian networks or Markov networks, however, we can also employ "counterexamples" for learning. Consider a simple kinship domain. Assume `rex` is a male person. Consequently, he cannot be the `daughter` of any other person, say `ann`. Thus, `daughter(rex,ann)` can be listed as a negative example although we will never observe it. "Counterexamples" conflict with the usual view on learning examples in statistical learning.

In statistical learning, we seek to find that hypothesis H^*, which is most likely given the learning examples:

$$H^* = \arg\max_H P(H|E) = \arg\max_H \frac{P(E|H) \cdot P(F)}{P(E)}$$
$$\text{with} \quad P(E) > 0 .$$

Thus, examples E in traditional statistical learning are always observable, that is, $P(E) > 0$. However, in statistical relational learning, as in inductive logic programming, we may also employ "counterexamples" such as `daughter(rex,ann)`, which have probability "0," and that actually never can be observed.

Definition 2 (**SRL Problem**). **Given** *a set* $E = E_p \cup E_i$ *of positive and negative examples* E_p *and* E_i *(with* $E_p \cap E_i = \emptyset$*) over some example language* $\mathcal{L}_E$*, a probabilistic covers relation covers*$(e, H, B) = P(e \mid H, B)$*, a probabilistic logical language* $\mathcal{L}_H$ *for hypotheses, and a background theory B,* ***find*** *a hypothesis* H^* *in* $\mathcal{L}_H$ *such that* $H^* = \arg\max_H score(E, H, B)$ *and the following constraints hold:* $\forall\ e_p \in E_p : covers(e_p, H^*, B) > 0$ *and* $\forall\ e_i \in E_i : covers(e_i, H^*, B) = 0$*. The score is some objective function, usually involving the probabilistic covers relation of the observed examples such as the observed likelihood* $\prod_{e_p \in E_p} covers(e_p, H^*, B)$ *or some penalized variant thereof.*

This learning setting unifies inductive logic programming and statistical learning in the following sense: using a deterministic covers relation (either 1 or 0), it yields the classical inductive logic programming learning problem; sticking to propositional logic and learning from *positive* examples, that is, $P(E) > 0$, only yields traditional statistical learning.

To come up with algorithms solving the SRL problem, say for density estimation, one typically distinguishes two subtasks because $H = (L, \lambda)$ is essentially a logical theory L annotated with probabilistic parameters λ:

1. *Parameter estimation* where it is assumed that the underlying logic program L is fixed, and the learning task consists of estimating the parameters λ that maximize the likelihood.
2. *Structure learning* where both L and λ have to be learned from the data.

In the following paragraphs, we will sketch the basic parameter estimation and structure learning techniques, and illustrate them for each setting.

Parameter Estimation The problem of parameter estimation is concerned with estimating the values of the parameters λ of a fixed probabilistic program $H = (L, \lambda)$ that best explains the examples E. So, λ is a set of parameters and can be represented as a vector. As already indicated above, to measure the extent to which a model fits the data, one usually employs the likelihood of the data, that is, $P(E \mid L, \lambda)$, though other scores or variants could be used as well.

When all examples are fully observable, maximum likelihood reduces to frequency counting. In the presence of missing data, however, the maximum likelihood estimate typically cannot be written in closed form. It is a numerical optimization problem, and all known algorithms involve nonlinear optimization. The most commonly adopted technique for probabilistic logic learning is the expectation-maximization (EM) algorithm (Dempster, Laird, Rubin, 1977; McLachlan & Krishnan, 1997). EM is based on the observation that learning would be easy (i.e., correspond to frequency counting), if the values of all the random variables would be known. Therefore, it estimates these values, maximizes the likelihood based on the estimates, and then iterates. More specifically, EM assumes that the parameters have been initialized (e.g., at random) and then iteratively performs the following two steps until convergence:

(E-Step) On the basis of the observed data and the present parameters of the model, it computes a distribution over all possible completions of each partially observed data case.

(M-Step) Treating each completion as a fully observed data case weighted by its probability, it computes the improved parameter values using (weighted) frequency counting.

The frequencies over the completions are called the *expected counts*. Examples for parameter estimation of probabilistic relational models can be found in (Getoor & Taskar, 2007) and (De Raedt, Frasconi, Kersting, & Muggleton, 2008).

Structure Learning The problem is now to learn both the structure L and the parameters λ of the probabilistic program $H = (L, \lambda)$ from data. Often, further information is given as well. As in inductive logic programming, the additional knowledge can take various different forms, including a ▶*language bias* that imposes restrictions on the syntax of L, and an *initial hypothesis* (L, λ) from which the learning process can start.

Nearly all (score-based) approaches to structure learning perform a heuristic search through the space of possible hypotheses. Typically, hill-climbing or beam-search is applied until the hypothesis satisfies the logical constraints and the $score(H, E)$ is no longer improving. The steps in the search-space are typically made using refinement operators, which make small, syntactic modifications to the (underlying) logic program.

At this point, it is interesting to observe that the logical constraints often require that the positive examples are covered in the logical sense. For instance, when learning ProbLog programs from entailment, the observed example clauses must be entailed by the logic program. Thus, for a probabilistic program $H = (L_H, \lambda_H)$ and a background theory $B = (L_B, \lambda_B)$ it holds that $\forall e_p \in E_p : P(e|H, B) > 0$ if and only if $covers(e, L_H, L_B) = 1$, where L_H (respectively L_B) is the underlying logic program (logical background theory) and *covers*(e, L_H, L_B) is the purely logical *covers* relation, which is either 0 or 1.

Applications

Applications of statistical relational learning can be found in many areas such as web search and mining, text mining, bioinformatics, natural language processing, robotics, and social network analysis, among others. Due to space restrictions, we will only name a few of these exciting applications.

For instance, Getoor, Taskar, & Koller (2001) have used statistical relational models to estimate the result size of complex database queries. Segal et al. have employed probabilistic relational models to cluster gene expression data (Segal, Taskar, Gasch, Friedman, & Koller, 2001) and to discover cellular processes from gene expression data (Segal, Battle, & Koller, 2003). Getoor et al. have used probabilistic relational models to understand tuberculosis epidemiology (Getoor, Rhee, Koller, & Small, 2004). McGovern et al. (2003) have estimated probabilistic relational trees to discover publication patterns in high-energy physics. Probabilistic relational trees have also been used to learn to rank brokers with respect to the probability that they would commit a serious violation of securities regulations in the near future (Neville et al., 2005). Anguelov et al. (2005) have used relational Markov networks for segmentation of 3D scan data. Markov networks have also been used to compactly represent object maps and to estimate trajectories of people (Limketkai, Liao, & Fox, 2005). Kersting et al. have employed relational hidden Markov models for protein fold recognition (Kersting, De Raedt, & Raiko, 2006). Poon and Domingos (2008) have shown how to use Markov logic to perform joint unsupervised coreference resolution. Xu et al. have used nonparametric relational models for analyzing social networks (Xu, Tresp, Rettinger, & Kersting, 2010). Kersting and Xu (2009) have used relational Gaussian processes for learning to rank search results. Recently, Poon and Domingos (2009) have shown how to perform unsupervised semantic parsing using Markov logic networks.

Future Directions

We have provided an overview of the new and exciting area of statistical relational learning. It combines principles of probabilistic reasoning, logical representation, and statistical learning into a coherent whole. The techniques of probabilistic logic learning were analyzed starting from an inductive logic programming perspective by lifting the coverage relation to a probabilistic one and annotating the logical formulae. Different choices of the probabilistic coverage relation lead to different representational formalisms, two of which were introduced.

Statistical relational learning is an active area of research within the machine learning and the artificial intelligence community. First, there is the issue of *efficient inference* and learning. Most current inference algorithms for statistical relational models require explicit state enumeration, which is often impractical: the number of states grows very quickly with the number of domain objects and relations. *Lifted* inference algorithms seek to avoid explicit state enumeration and directly work at the level of groups of atoms, eliminating all the instantiations of a set of atoms in a single step, in some cases independently of the number of these instantiations. Despite various approaches to lifted inference (de Salvo Braz, Amir, & Roth, 2005; Jaimovich, Meshi, & Friedman, 2007; Kersting, Ahmadi, & Natarajan, 2009; Kisynski & Poole, 2009; Milch, Zettlemoyer, Kersting, Haimes, & Kaelbling , 2008; Poole, 2003; Sen, Deshpande, & Getoor, 2008; Singla & Domingos, 2008), it largely remains a challenging problem. For what concerns learning, advanced principles of both statistical learning and logical and relational learning can be employed for learning the parameters and the structure of probabilistic logics such as statistical *predicate invention* (Kok & Domingos, 2007) and *boosting* (Gutmann & Kersting, 2006). Recently, people started to investigate *learning from weighted examples* (see e.g., Chen, Muggleton, & Santos, 2008) and to link statistical relational learning to support vector machines (see e.g., Passerini, Frasconi, & De Raedt, 2006). Second, there is the issue of *closed-world versus open-world* assumption that is, do we know how many objects there are (see e.g., Milch et al., 2005). Third, there is interest in dealing with *continuous values* within statistical relational learning (see e.g., Chu, Sindhwani, Ghahramani, & Keerthi, 2006; Silva, Chu, & Ghahramani, 2007; Wang & Domingos, 2008; Xu, Kersting, & Tresp, 2009). This is mainly motivated by the fact that most real-world applications actually contain continuous values. *Nonparametric Bayesian* approaches to statistical relational learning have also been developed (see e.g., Kemp, Tenenbaum, Griffiths, Yamada, & Ueda, 2006; Xu, Tresp, Yu, & Kriegel, 2006; Yu & Chu, 2007; Yu, Chu, Yu, Tresp, & Xu, 2006), to overcome the typically strong parametric assumptions underlying current statistical relational learning. People have also started to investigate *relational variants of classical statistical learning tasks* such as matrix factorizations (see e.g., Singh & Gordon, 2008). Finally, while statistical relational learning approaches have been used successfully in a number of applications, they do not yet cope with the *dynamic environments* in an effective way.

Cross References

►Multi-Relational Data Mining
►Relational Learning

Recommended Reading

In addition to the references embedded in the text above, we also recommend De Raedt et al. (2008), Getoor & Taskar (2007), De Raedt (2008) and the SRL tutorials at major artificial intelligence and machine learning conferences.

Anguelov, D., Taskar, B., Chatalbashev, V., Koller, D., Gupta, D., Heitz, G., et al. (2005). Discriminative learning of Markov random fields for segmentation of 3D scan data. In C. Schmid, S. Soatto, & C. Tomasi (Eds.), *IEEE Computer Society international conference on computer vision and pattern recognition (CVPR-05)*, San Diego, CA, USA (Vol. 2, pp. 169–176).

Chen, J., Muggleton, S., & Santos, J. (2008). Learning probabilistic logic models from probabilistic examples. *Machine Learning, 73*(1), 55–85.

Chu, W., Sindhwani, V., Ghahramani, Z., & Keerthi, S. (2006). Relational learning with Gaussian processes. In *Advances in Neural information processing systems 19* (NIPS-2006). Cambridge, MA: MIT Press.

Cussens, J. (1999). Loglinear models for first-order probabilistic reasoning. In K. Blackmond Laskey & H. Prade (Eds.), *Proceedings of the fifteenth annual conference on uncertainty in artificial intelligence (UAI-99)* (pp. 126–133), Stockholm, Sweden. San Francisco: Morgan Kaufmann.

Cussens, J. (2001). Parameter estimation in stochastic logic programs. *Machine Learning Journal, 44*(3), 245–271.

De Raedt, L. (2008). *Logical and relational learning*. Springer.

De Raedt, L., Frasconi, P., Kersting, K., & Muggleton, S. (Eds.). (2008). *Probabilistic inductive logic programming. Lecture notes in computer science* (Vol. 4911). Berlin/Heidelberg: Springer.

De Raedt, L., Kimmig, A., & Toivonen, H. (2007). Problog: A probabilistic Prolog and its application in link discovery. In M. Veloso (Ed.), *Proceedings of the 20th international joint conference on artificial intelligence* (pp. 2462–2467). Hyderabad, India.

de Salvo Braz, R., Amir, E., & Roth, D. (2005). Lifted first order probabilistic inference. In *Proceedings of the 19th international joint conference on artificial intelligence (IJCAI-05)* (pp. 1319–1325). Edinburgh, Scotland.

Dempster, A., Laird, N., & Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B, 39*, 1–39.

Džeroski, S., & Lavrač, N. (Eds.). (2001). *Relational data mining*. Berlin: Springer.

Eisele, A. (1994). Towards probabilistic extensions of contraint-based grammars. In J. Dörne (Ed.), *Computational aspects of constraint-based linguistics description-II*. Stuttgart: Institute for Computational Linguistics (IMS-CL). DYNA-2 deliverable R1.2.B.

Getoor, L. (2001). *Learning statistical models from relational data*. PhD thesis, Stanford University, USA.

Getoor, L., Rhee, J., Koller, D., & Small, P. (2004). Understanding tuberculosis epidemiology using probabilistic relational models. *Journal of Artificial Intelligence in Medicine, 30*, 233–256.

Getoor, L., & Taskar, B. (Eds.). (2007). *Introduction to statistical relational learning*. Cambridge, MA, USA: The MIT Press.

Getoor, L., Taskar, B., & Koller, D. (2001). Using probabilistic models for selectivity estimation. In *Proceedings of ACM SIGMOD international conference on management of data* (pp. 461–472). Santa Barbara, CA, USA: ACM Press.

Gutmann, B., & Kersting, K. (2006). TildeCRF: Conditional random fields for logical sequences. In J. Fuernkranz, T. Scheffer, & M. Spiliopoulou (Eds.), *Proceedings of the 17th European conference on machine learning (ECML-2006)*, Berlin, Germany (pp. 174–185).

Jäger, M. (1997). Relational Bayesian networks. In K. Laskey & H. Prade (Eds.), *Proceedings of the thirteenth conference on uncertainty in artificial intelligence (UAI-97)*, Stockholm, Sweden (pp. 266–273). San Franciso, CA, USA: Morgan Kaufmann.

Jaimovich, A., Meshi, O., & Friedman, N. (2007). Template-based inference in symmetric relational Markov random fields. In *Proceedings of the conference on uncertainty in artificial intelligence (UAI-07)* (pp. 191–199).

Kemp, C., Tenenbaum, J., Griffiths, T., Yamada, T., & Ueda, N. (2006). Learning systems of concepts with an infinite relational model. In *Proceedings of 21st AAAI*.

Kersting, K., Ahmadi, B., & Natarajan, S. (2009). Counting belief propagation. In *Proceedings of the 25th conference on uncertainty in artificial intelligence (UAI-09)*. Montreal, Canada.

Kersting, K., & De Raedt, L. (2007). Bayesian logic programming: Theory and tool. In L. Getoor & B. Taskar (Eds.), *An introduction to statistical relational learning* (pp. 291–321). Cambridge, MA, USA: MIT Press.

Kersting, K., De Raedt, L., & Raiko, T. (2006). Logial Hidden Markov Models. *Journal of Artificial Intelligence Research (JAIR), 25*, 425–456.

Kersting, K., & Xu, Z. (2009). Learning preferences with hidden common cause relations. In *Proceedings of the European conference on machine learning and principles and practice of knowledge discovery in databases (ECML PKDD 09). LNAI*, Bled, Slovenia, Springer.

Kisynski, J., & Poole, D. (2009). Lifted aggregation in directed first-order probabilistic models. In C. Boutilier (Ed.), *Proceedings of the international joint conference on artificial intelligence (IJCAI-09)*. Pasadena, CA, USA.

Kok, S., & Domingos, P. (2007). Statistical predicate invention. In *Proceedings of the twenty-fourth international conference on machine learning (ICML-07)*, Corvallis, OR, USA (pp. 433–440). ACM Press.

Limketkai, B., Liao, L., & Fox, D. (2005). Relational object maps for mobile robots. In F. Giunchiglia & L. P. Kaelbling (Eds.), *Proceedings of the nineteenth international joint conference on artificial intelligence (IJCAI-05)*, Edinburgh, Scotland (pp. 1471–1476). AAAI Press.

McGovern, A., Friedland, L., Hay, M., Gallagher, B., Fast, A., Neville, J., et al. (2003). Exploiting relational structure to understand publication patterns in high-energy physics. *SIGKDD Explorations, 5*(2), 165–173.

McLachlan, G., & Krishnan, T. (1997). *The EM algorithm and extensions*. New York: Wiley.

Milch, B., Marthi, B., Russell, S., Sontag, D., Ong, D., & Kolobov, A. (2005). BLOG: Probabilistic models with unknown objects. In F. Giunchiglia, L. P. Kaelbling (Eds.), *Proceedings of the nineteenth international joint conference on artificial intelligence (IJCAI-05)*, Edinburgh, Scotland (pp. 1352–1359). Edinburgh, Scotland: AAAI Press.

Milch, B., Zettlemoyer, L., Kersting, K., Haimes, M., & Pack Kaelbling, L. (2008). Lifted probabilistic inference with counting formulas. In *Proceedings of the 23rd AAAI conference on artificial intelligence (AAAI-08)*.

Muggleton, S. (1996). Stochastic logic programs. In L. De Raedt (Ed.), *Advances in inductive logic programming* (pp. 254–264). Amsterdam: IOS Press.

Muggleton, S., & De Raedt, L. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming, 19*(20), 629–679.

Neville, J., & Jensen, D. (2004). Dependency networks for relational data. In R. Rastogi, K. Morik, M. Bramer, & X. Wu (Eds.), *Proceedings of the fourth IEEE international conference on data mining (ICDM-04)*, Brighton, UK (pp. 170–177). IEEE Computer Society Press.

Neville, J., Simsek, Ö., Jensen, D., Komoroske, J., Palmer, K., & Goldberg, H. (2005). Using relational knowledge discovery to prevent securities fraud. In *Proceedings of the 11th ACM SIGKDD international conference on knowledge discovery and data mining*. Chicago, IL, USA: ACM Press.

Ngo, L., & Haddawy, P. (1997). Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science, 171*, 147–177.

Passerini, A., Frasconi, P., & De Raedt, L. (2006). Kernels on prolog proof trees: Statistical learning in the ILP setting. *Journal of Machine Learning Research, 7*, 307–342.

Pfeffer, A. (2000). *Probabilistic reasoning for complex systems*. PhD thesis, Computer Science Department, Stanford University.

Poole, D. (1993). Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence Journal, 64*, 81–129.

Poole, D. (1997). The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence, 94*(1–2), 7–56.

Poole, D. (2003). First-order probabilistic inference. In G. Gottlob & T. Walsh (Eds.), *Proceedings of the eighteenth international joint conference on artificial intelligence (IJCAI-03)*, Acapulco, Mexico (pp. 985–991). San Francisco: Morgan Kaufmann.

Poon, H., & Domingos, P. (2008). Joint unsupervised coreference resolution with markov logic. In: *Proceedings of the 2008 conference on empirical methods in natural language processing (EMNLP)*, Honolulu, HI, USA.

Poon, H., & Domingos, P. (2009). Unsupervised semantic parsing. In *Proceedings of the 2009 conference on empirical methods in natural language processing (EMNLP)*, Singapore.

Richardson, M., & Domingos, P. (2006). Markov logic networks. *Machine Learning, 62*, 107–136.

Sato, T. (1995). A statistical learning method for logic programs with distribution semantics. In L. Sterling (Ed.), *Proceedings of the twelfth international conference on logic programming (ICLP-95)*, Tokyo, Japan (pp. 715–729). MIT Press.

Segal, E., Battle, A., & Koller, D. (2003). Decomposing gene expression into cellular processes. In *Proceedings of Pacific symposium on biocomputing (PSB)* (pp. 89–100). World Scientific.

Segal, E., Taskar, B., Gasch, A., Friedman, N., & Koller, D. (2001). Rich probabilistic models for gene expression. *Bioinformatics, 17*(Suppl. 1), S243–252 (Proceedings of ISMB 2001).

Sen, P., Deshpande, A., & Getoor, L. (2008). Exploiting shared correlations in probabilistic databases. In *Proceedings of the international conference on very large data bases (VLDB-08)*. Auckland, New Zealand.

Silva, R., Chu, W., & Ghahramani, Z. (2007). Hidden common cause relations in relational learning. In *Advances in Neural information processing systems 20* (NIPS-2007). Cambridge, MA: MIT Press.

Singh, A., & Gordon, G. (2008). Relational learning via collective matrix factorization. In *Proceedings of 14th international conference on knowledge discovery and data mining*. Las Vegas, US.

Singla, P., & Domingos, P. (2008). Lifted first-order belief propagation. In *Proceedings of the 23rd AAAI conference on artificial intelligence (AAAI-08)*, Chicago, IL, USA (pp. 1094–1099).

Taskar, B., Abbeel, P., & Koller, D. (2002). Discriminative probabilistic models for relational data. In A. Darwiche & N. Friedman (Eds.), *Proceedings of the eighteenth conference on uncertainty in artificial intelligence (UAI-02)*, Edmonton, Alberta, Canada (pp. 485–492).

Wang, J., & Domingos, P. (2008). Hybrid markov logic networks. In *Proceedings of the 23rd AAAI conference on artificial intelligence (AAAI-08)*, Chicago, IL, USA (pp. 1106–1111).

Xu, Z., Kersting, K., & Tresp, V. (2009). Multi-relational learning with Gaussian processes. In C. Boutilier (Ed.) *Proceedings of the international joint conference on artificial intelligence (IJCAI-09)*. Pasadena, CA.

Xu, Z., Tresp, V., Rettinger, A., & Kersting, K. (2010). Social network mining with nonparametric relational models. In *Advances in social network mining and analysis*. Lecture Notes in Computer Science (Vol. 5498), Berlin/Heidelberg: Springer.

Xu, Z., Tresp, V., Yu, K., & Kriegel, H. P. (2006). Infinite hidden relational models. In *Proceedings of 22nd UAI*.

Yu, K., & Chu, W. (2007). Gaussian process models for link analysis and transfer learning. In *Advances in Neural information processing systems 20* (NIPS-2007). Cambridge, MA: MIT Press.

Yu, K., Chu, W., Yu, S., Tresp, V., & Xu, Z. (2006). Stochastic relational models for discriminative link prediction. In *Advances in Neural information processing systems 19* (NIPS-2006). Cambridge, MA: MIT Press.

Stochastic Finite Learning

Thomas Zeugmann
Hokkaido University,
Sapparo, Japan

Motivation and Background

Assume that we are given a concept class $\mathcal{C}$ and should design a learner for it. Next, suppose we already know or could prove $\mathcal{C}$ not to be learnable in the model of ►PAC Learning. But it can be shown that $\mathcal{C}$ is learnable within Gold's (1967) model of ►Inductive Inference or learning in the limit. Thus, we can design a learner behaving as follows. When fed any of the data sequences allowed in this model, it converges in the limit to a hypothesis correctly describing the target concept. Nothing more is known. Let M be any fixed learner. If $(d_n)_{n \geq 0}$ is any data sequence, then the *stage of convergence* is the least integer m such that $M(d_m) = M(d_n)$ for all $n \geq m$, provided such an n exists (and infinite, otherwise). In general, it is undecidable whether or not the learner has already reached the stage of convergence, but even if it is decidable for a particular concept class, it may be practically infeasible to do so. This *uncertainty* may not be tolerable in many applications.

When we tried to overcome this uncertainty, the idea of stochastic finite learning emerged. Clearly, in general nothing can be done, since in Gold's (1967) model the learner has to learn from any data sequence. So for every concept that needs more than one datum to converge, one can easily construct a sequence, where the first datum is repeated very often and where therefore, the learner does not find the right hypothesis within the given bound. However, such data sequences seem unnatural. Therefore, we looked at data sequences that are generated with respect to some probability distribution taken from a prespecified class of probability distributions and computed the expected *total learning time*, i.e., the expected time until the learner reaches the stage of convergence (cf. Erlebach, Rossmanith, Stadtherr, Steger, & Zeugmann, 2001; Zeugmann, 2006). Clearly, one is then also interested in knowing how often the expected total learning time is exceeded. In general, Markov's inequality can be applied to obtain the relevant tail bounds. However, if the learner is known to be rearrangement independent and conservative, then we always get *exponentially* shrinking tail bounds (cf. Rossmanith & Zeugmann, 2001). A learner is said to be *rearrangement independent*, if its output depends exclusively on the range and length of its input (but not the order) (cf., e.g., Lange & Zeugmann, 1996 and the references therein). Furthermore, a learner is *conservative*, if it exclusively performs mind changes that can be justified by an inconsistency of the abandoned hypothesis with the data received so far (see Angluin, 1980b for a formal definition).

Combining these ideas results in stochastic finite learning. A stochastic finite learner is successively fed data about the target concept. Note that these data are generated randomly with respect to one of the probability distributions from the class of underlying probability distributions. Additionally, the learner takes a confidence parameter δ as input. But in contrast to learning in the limit, the learner itself decides how many examples it wants to read. Then it computes a hypothesis, outputs it and stops. The hypothesis output is correct for the target with a probability at least $1 - \delta$.

The description given above explains how it works, but not why it does. Intuitively, the stochastic finite learner simulates the limit learner until an upper bound for twice the expected total number of examples needed until convergence has been met. Assuming this to be true, by Markov's inequality, the limit learner has now converged with a probability $1/2$. All what is left is to decrease the probability of failure. This can be done by using again Markov's inequality, i.e., increasing the sample complexity by a factor of $1/\delta$ results in a confidence of $1 - \delta$ for having reached the stage of convergence.

Note that the stochastic finite learner has to calculate an upper bound for the stage of convergence. This is precisely the point where we need the parameterization of the class $\mathcal{D}$ of underlying probability distributions. Then, a bit of *prior knowledge* must be provided in the form of suitable upper and/or lower bounds for the parameters involved. A more serious difficulty is to incorporate the unknown target concept into this estimate. This step depends on the concrete learning problem on hand and requires some extra effort.

It should also be noted that our approach may be beneficial even in case that the considered concept class is PAC learnable.

Definition

Let $\mathcal{D}$ be a set of probability distributions on the learning domain, let $\mathcal{C}$ be a concept class, $\mathcal{H}$ a hypothesis space for $\mathcal{C}$, and $\delta \in (0,1)$. $(\mathcal{C},\mathcal{D})$ is said to be *stochastically finitely learnable with δ-confidence* with respect to $\mathcal{H}$ iff there is a learner M that for every $c \in \mathcal{C}$ and every $D \in \mathcal{D}$ performs as follows. Given any random data sequence θ for c generated according to D, M stops after having seen a finite number of examples and outputs a single hypothesis $h \in \mathcal{H}$. With a probability at least $1 - \delta$ (with respect to distribution D), h has to be correct, i.e., $c = h$.

If stochastic finite learning can be achieved with δ-confidence for every $\delta > 0$ then we say that $(\mathcal{C},\mathcal{D})$ can be learned stochastically finite *with high confidence*.

Detail

Note that there are subtle differences between our model and PAC learning. By its definition, stochastic finite learning is not completely distribution independent. A bit of *additional knowledge* concerning the underlying probability distributions is required. Thus, from that perspective, stochastic finite learning is weaker than the PAC model. On the other hand, we do *not* measure the quality of the hypothesis with respect to the underlying probability distribution. Instead, we require the hypothesis computed to be exactly correct with high probability. Note that the exact identification with high confidence has been considered within the PAC paradigm, too (cf., e.g., Saly, Goldman, & Schapire, 1993). Conversely, we also can easily relax the requirement to learn *probably exactly correct*, but whenever possible we shall not do it.

Furthermore, in the uniform PAC model as introduced in Valiant (1984), the sample complexity depends exclusively on the VC dimension of the target concept class and the error and confidence parameters ε and δ, respectively. This model has been generalized by allowing the sample size to depend on the concept complexity, too (cf., e.g., Blumer, Ehrenfeucht, Haussler, & Warmuth, 1989; Haussler, Kearns, Littlestone, & Warmuth, 1991). Provided no upper bound for the concept complexity of the target concept is given, such PAC learners decide themselves how many examples they wish to read (cf. Haussler et al., 1991). This feature is also adopted to our setting of stochastic finite learning. However, all variants of efficient ►PAC Learning, we are aware of, require that all hypotheses from the relevant hypothesis space are uniformly polynomially evaluable. Though this requirement may be necessary in some cases to achieve (efficient) stochastic finite learning, it is not necessary in general as we shall see below.

In the following, we provide two sample applications of Stochastic Finite Learning. We always choose the concept class $\mathcal{C}$ itself as hypothesis space.

Learning Monomials

Let $X_n = \{0,1\}^n$ be the learning domain, let $\mathcal{L}_n = \{x_1, \bar{x}_1, x_2, \bar{x}_2 \ldots, x_n, \bar{x}_n\}$ (set of literals), and consider the class $\mathcal{C}_n$ of all concepts describable by a conjunction of literals. As usual, we refer to any conjunction of literals as a *monomial*. A monomial m describes a concept $c \subseteq X_n$ in the obvious way: The concept contains exactly those binary vectors for which the monomial evaluates to 1.

The basic ingredient to the stochastic finite learner is Haussler's (1987) Wholist algorithm, and thus the main emphasis is on the resulting complexity. The Wholist algorithm can also be used to achieve ►PAC Learning of the class $\mathcal{C}_n$ and the resulting sample complexity is $O(1/\varepsilon \cdot (n + \ln(1/\delta)))$ for all $\varepsilon, \delta \in (0,1]$. Since the Wholist algorithm learns from positive examples only, it is meaningful to study the learnability of $\mathcal{C}_n$ from positive examples only. So, the stage of convergence is *not* decidable.

Since the Wholist algorithm immediately converges for the empty concept, we exclude it from our considerations. That is, we consider concepts $c \in \mathcal{C}_n$ described by a monomial $m = \bigwedge_{j=1}^{\#(m)} \ell_{i_j}$ such that $k = k(m) = n - \#(m) > 0$. A literal not contained in m is said to be irrelevant. Bit i is said to be irrelevant for monomial m if neither x_i nor $\bar{x}_i$ appear in m. There are 2^k positive examples for c. For the sake of presentation, we assume these examples to be *binomially distributed* with parameter p. So, in a random positive example, all entries corresponding to irrelevant bits are selected independently of each other. With some probability p this will be a 1, and with probability $1-p$, this will be a 0. Only distributions where $0 < p < 1$ are considered, since otherwise exact identification is impossible. Now, one can show that the expected number of examples needed by the Wholist algorithm until convergence is bounded by $\lceil \log_\psi k(m) \rceil + \tau + 2$, where $\psi := \min\{\frac{1}{1-p}, \frac{1}{p}\}$ and $\tau := \max\{\frac{p}{1-p}, \frac{1-p}{p}\}$.

Let CON denote a random variable for the stage of convergence. Since the Wholist algorithm is rearrangement independent and conservative, we can conclude (cf. Rossmanith & Zeugmann, 2001)

$$\Pr(CON > 2t \cdot \mathrm{E}[CON]) \leq 2^{-t} \text{for all natural numbers } t \geq 1.$$

Finally, in order to obtain a stochastic finite learner, we reasonably assume that the *prior knowledge* is provided by parameters p_{low} and p_{up} such that $p_{\text{low}} \leq p \leq p_{\text{up}}$ for the true parameter p. Binomial distributions fulfilling this requirement are called $(p_{\text{low}}, p_{\text{up}})$*-admissible distributions*. Let $\mathcal{D}_n[p_{\text{low}}, p_{\text{up}}]$ denote the set of such distributions on X_n. Then one can show

Let $0 < p_{\text{low}} \leq p_{\text{up}} < 1$ *and* $\psi := \min\left\{\frac{1}{1-p_{\text{low}}}, \frac{1}{p_{\text{up}}}\right\}$. *Then* $(\mathcal{C}_n, \mathcal{D}_n[p_{\text{low}}, p_{\text{up}}])$ *is stochastically finitely learnable with high confidence from positive examples. To achieve* δ*-confidence no more than* $O\left(\log_2 1/\delta \cdot \log_\psi n\right)$, *many examples are necessary.*

Therefore, we have achieved an exponential improvement on the number of examples needed for learning (compared to the PAC bound displayed above), and, in addition, our stochastic finite learner exactly identifies the target. Note that this result is due to Reischuk and Zeugmann, however, we refer the reader to Zeugmann (2006) for the relevant proofs.

The results obtained for learnability from positive examples only can be extended mutatis mutandis to the case when the learner is fed positive and negative examples (cf. Zeugmann, 2006 for details).

Learning Pattern Languages

The pattern languages have been introduced by Angluin (1980a) and can be informally defined as follows. Let $\Sigma = \{0, 1, \ldots\}$ be any finite alphabet containing at least two elements. Let $X = \{x_0, x_1, \ldots\}$ be a countably infinite set of variables such that $\Sigma \cap X = \varnothing$. *Patterns* are nonempty strings over $\Sigma \cup X$, e.g., 01, $0x_0111$, $1x_0x_00x_1x_2x_0$ are patterns. The length of a string $s \in \Sigma^*$ and of a pattern π is denoted by $|s|$ and $|\pi|$, respectively. A pattern π is in *canonical form* provided that if k is the number of different variables in π then the variables occurring in π are precisely $x_0, \ldots, x_{k-1}$. Moreover, for every j with $0 \leq j < k-1$, the leftmost occurrence of x_j in π is left to the leftmost occurrence of x_{j+1}. The examples given above are patterns in canonical form.

If k is the number of different variables in π then we refer to π as to a *k-variable pattern*. For example, $x0xx$ is a one-variable pattern, and $x_010x_1x_0$ is a two-variable pattern. If π is a pattern, then the language generated by π is the set of all strings that can be obtained from π by substituting a nonnull element $s_i \in \Sigma^*$ for each occurrence of the variable symbol x_i in π, for all $i \geq 0$. We use $L(\pi)$ to denote the language generated by pattern π. So, 1011, 1001010 belong to $L(x0xx)$ (by substituting 1 and 10 for x, respectively) and 010110 is an element of $L(x_010x_1x_0)$ (by substituting 0 for x_0 and 11 for x_1). Note that even the class of all one-variable patterns has infinite ►VC Dimension (cf. Mitchell, Scheffer, Sharma, & Stephan, 1999).

Reischuk and Zeugmann (2000) designed a stochastic finite learner for the class of all one-variable pattern languages that runs in time $O(|\pi| \log(1/\delta))$ for all meaningful distributions and learns from positive data only. That is, all data fed to the learner belong to the target pattern language. Furthermore, by meaningful distribution essentially the following is meant. The expected length of an example should be finite, and the distribution should allow to learn the target pattern. This is then expressed by fixing some suitable parameters. It should be noted that the algorithm is highly practical, and a modification of it also works for the case that *empty* substitutions are allowed.

For the class of all pattern languages, one can also provide a stochastic finite learner, identifying the whole class from positive data. In order to arrive at a suitable class of distributions, essentially three requirements are made. The first one is the same as in the one-variable case, i.e., the expected length $\mathrm{E}[\Lambda]$ of a generated string should be finite. Second, the class of distributions is restricted to regular product distributions, i.e., for all variables the substitutions are identically distributed. Third, two parameters α and β are introduced. The parameter α is the probability that a string of length 1 is substituted and β is the conditional probability that two random strings that get substituted into π are identical under the condition that both have length 1. These two parameters ensure that the target pattern language is learnable at all. The stochastic finite learner is then using as *a* priori knowledge a lower bound α^* for α and an upper bound β^* for β. The basic ingredient to this stochastic finite learner is Lange and Wiehagen's (1991) pattern language learning algorithm. Rossmanith and Zeugmann's (2001) stochastic finite learner for the pattern languages runs in time $O\left((1/\alpha_*^k)E[\Lambda]\log_{1/\beta_*}(k)\log_2(1/\delta)\right)$, where k is the number of different variables in the target pattern. So, with increasing k it becomes impractical.

Note that the two stochastic finite learners for the pattern languages can compute the expected stage of convergence, since the first string seen provides an upper bound for the length of the target pattern.

For further information, we refer the reader to Zeugmann (2006) and the references therein. More research is needed to explore the potential of stochastic finite learning. Such investigations should extend the learnable classes, study the incorporation of noise, and explore further possible classes of meaningful probability distributions.

Cross References

►Inductive Inference
►PAC Learning

Recommended Reading

Angluin, D. (1980a). Finding patterns common to a set of strings. *Journal of Computer and System Sciences, 21*(1), 46–62.

Angluin, D. (1980b). Inductive inference of formal languages from positive data. *Information Control, 45*(2), 117–135.

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. K. (1989). Learnability and the Vapnik–Chervonenkis dimension. *Journal of the ACM, 36*(4), 929–965.

Erlebach, T., Rossmanith, P., Stadtherr, H., Steger, A., & Zeugmann, T. (2001). Learning one-variable pattern languages very efficiently on average, in parallel, and by asking queries. *Theoretical Computer Science, 261*(1), 119–156.

Gold, E. M. (1967). Language identification in the limit. *Information and Control, 10*(5), 447–474.

Haussler, D. (1987). Bias, version spaces and Valiant's learning framework. In P. Langley (Ed.), *Proceedings of the fourth international workshop on machine learning* (pp. 324–336). San Mateo, CA: Morgan Kaufmann.

Haussler, D., Kearns, M., Littlestone, N., & Warmuth, M. K. (1991). Equivalence of models for polynomial learnability. *Information and Computation, 95*(2), 129–161.

Lange, S., & Wiehagen, R. (1991). Polynomial-time inference of arbitrary pattern languages. *New Generation Computing, 8*(4), 361–370.

Lange, S., & Zeugmann, T. (1996). Set-driven and rearrangement-independent learning of recursive languages. *Mathematical Systems Theory, 29*(6), 599–634.

Mitchell, A., Scheffer, T., Sharma, A., & Stephan, F. (1999). The VC-dimension of sub- classes of pattern languages. In O. Watanabe & T. Yokomori (Eds.), *Algorithmic learning theory, tenth international conference, ALT"99, Tokyo, Japan, December 1999, Proceedings, Lecture notes in artificial intelligence* (Vol. 1720, pp. 93–105). Springer.

Reischuk, R., & Zeugmann, T. (2000). An average-case optimal one-variable pattern language learner. *Journal of Computer and System Sciences, 60*(2), 302–335.

Rossmanith, P., & Zeugmann, T. (2001). Stochastic finite learning of the pattern languages. *Machine Learning, 44*(1/2), 67–91.

Saly, A., Goldman, M. J. K., & Schapire, R. E. (1993). Exact identification of circuits using fixed points of amplification functions. *SIAM Journal of Computing, 22*(4), 705–726.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM, 27*(11), 1134–1142.

Zeugmann, T. (1998). Lange and Wiehagen's pattern language learning algorithm: An average-case analysis with respect to its total learning time. *Annals of Mathematics and Artificial Intelligence, 23*, 117–145.

Zeugmann, T. (2006). From learning in the limit to stochastic finite learning. *Theoretical Computer Science, 364*(1), 77–97. Special issue for ALT 2003.

Stratified Cross Validation

Stratified Cross Validation is a form of ►cross validation in which the class distribution is kept as close as possible to being the same across all folds.

Stream Mining

A subfield of knowledge discovery called *stream mining* addresses the issue of rapidly changing data. The idea is to be able to deal with the stream of incoming data quickly enough to be able to simultaneously update the corresponding models (e.g., ontologies), as the amount of data is too large to be stored: new evidence from the incoming data is incorporated into the model without storing the data. For instance, modeling ontology changes and evolution over time using text mining methods (►Text Mining for Semantic Web). The underlying methods are based on the machine learning methods of ►Online Learning, where the model is built from the initially available data and updated regularly as more data become available.

Examples of data streams include computer network traffic, phone conversations, ATM transactions, web searches, and sensor data.

Cross References

►Clustering Data Streams
►Online Learning

String kernel

A *string kernel* is a function from any of various families of kernel functions (see ►kernel methods) that operate over strings and sequences. The most typical example is as follows. Suppose that we are dealing with strings over a finite alphabet Σ. Given a string $a = a_1a_2\ldots a_n \in \Sigma^*$, we say that a substring $p = p_1p_2\ldots p_k$ *occurs* in a on positions $i_1\ i_2\ldots i_k$ iff $1 \le i_1 < i_2 < \ldots < i_k \le n$ and $a_{i_j} = p_j$ for all $j = 1,\ldots,k$. We define the *weight* of this occurrence as $\lambda^{i_k - i_1 - k + 1}$, where $\lambda \in [0,1]$ is a constant chosen in advance; in other words, an occurrence weighs less if characters of p are separated by other characters. Let $\phi_p(a)$ be the sum of the weights of all occurrences of p in a, and let $\phi(a) = (\phi_p(a))_{p\in\Sigma^*}$ be an infinite-dimensional feature vector consisting of $\phi_p(a)$ for all possible substrings $p \in \Sigma^*$. It turns out that the dot product of two such infinite-length vectors, $K(a,a') = \phi(a)^T\phi(a')$, can be computed in time polynomial in the length of a and a', e.g., using dynamic programming. The function K defined in this way can be used as a kernel with various kernel methods. See also ►feature construction in text mining.

String Matching Algorithm

A string matching algorithm returns parts of text matching a given pattern, such as a *regular expression*. Such algorithms have countless applications, from file editing to bioinformatics. Many algorithms compute deterministic finite automata, which can be expensive to build, but are usually efficient to use; they include the *Knuth–Morris–Pratt* algorithm and the *Boyer–Moore* algorithm, that build the automaton in time $O(m)$ and $O(m+s)$, respectively, where m is the length of the pattern and s the size of the alphabet, and match a text of length n in time $O(n)$ in the worst case.

Structural Credit Assignment

►Credit Assignment

Structural Risk Minimization

Xinhua Zhang
Australian National University, NICTA London Circuit,
Canberra, Australia

Definition

The goal of learning is usually to find a model which delivers good generalization performance over an underlying distribution of the data. Consider an input space $\mathcal{X}$ and output space $\mathcal{Y}$. Assume the pairs $(X \times Y) \in \mathcal{X}\times\mathcal{Y}$ are random variables whose (unknown) joint distribution is P_{XY}. It is our goal to find a predictor $f : \mathcal{X} \mapsto \mathcal{Y}$ which minimizes the expected risk:

$$P(f(X) \ne Y) = \mathbb{E}_{(X,Y)\sim P_{XY}}\left[\delta(f(X) \ne Y)\right],$$

where $\delta(z) = 1$ if z is true, and 0 otherwise.

In practice we only have n pairs of training examples (X_i, Y_i) drawn identically and independently from P_{XY}. Based on these samples, the ►empirical risk can be defined as

$$\frac{1}{n}\sum_{i=1}^{n}\delta(f(X_i) \ne Y_i).$$

Choosing a function f by minimizing the empirical risk often leads to ►overfitting. To alleviate this problem, the idea of structural risk minimization (SRM) is to employ an infinite sequence of models $\mathcal{F}_1, \mathcal{F}_2, \ldots$ with increasing capacity. Here each $\mathcal{F}_i$ is a set of functions, e.g., polynomials with degree 3. We minimize the empirical risk in each model with a penalty for the capacity of the model:

$$f_n := \underset{f\in\mathcal{F}_i, i\in\mathbb{N}}{\arg\min}\ \frac{1}{n}\sum_{j=1}^{n}\delta(f(X_j) \ne Y_j) + \text{capacity}(\mathcal{F}_i, n),$$

where capacity$(\mathcal{F}_i, n)$ quantifies the complexity of model $\mathcal{F}_i$ in the context of the given training set. For example, it equals 2 when $\mathcal{F}_i$ is the set of polynomials with degree 2. In other words, when trying to reduce the risk on the training set, we prefer a predictor from a simple model.

Note the penalty is measured on the model $\mathcal{F}_i$, *not* the predictor f. This is different from the regularization framework, e.g., support vector machines, which penalizes the complexity of the *classifier*.

More details about SRM can be found in Vapnik (1998).

Recommended Reading

Vapnik, V. (1998). *Statistical learning theory*. New York: Wiley.

Structure

►Topology of a Neural Network

Structured Data Clustering

►Graph Clustering

Structured Induction

Michael Bain
University of New South Wales,
Sydney, Australia

Definition

Structured induction is a method of applying machine learning in which a model for a task is learned using a representation where some of the components are themselves the outputs of learned models for specified sub-tasks. The idea was inspired by structured programming (Dahl, Dijkstra and Hoare, 1972), in which a complex task is solved by repeated decomposition into simpler sub-tasks that can be easily analyzed and implemented. The approach was first developed by Alen Shapiro (1987) in the context of constructing expert systems by ►decision tree learning, but in principle it could be applied using other learning methods.

Motivation and Background

Structured induction is designed to solve complex learning tasks for which it is difficult a priori to obtain a set of attributes or features in which it is possible to represent an accurate approximation of the target hypothesis reasonably concisely. In Shapiro's approach, a hierarchy of ►decision trees is learned, where in each tree of the hierarchy the attributes can have values that are outputs computed by a lower-level ►decision tree. Shapiro showed in computer chess applications that structured induction could learn accurate models, while significantly reducing their complexity. Structured induction was first commercialized in the 1980s by a number of companies providing expert systems solutions and has since seen many applications (Razzak, Hassan and Pettipher, 1984).

A key assumption is that human expertise is available to define the task structure. Several approaches have been proposed to address the problem of learning this structure (under headings such as ►constructive induction, ►representation change, ►feature construction, and ►predicate invention) although to date, none have received wide acceptance.

The identification of knowledge acquisition as the "bottleneck" in knowledge engineering by Feigenbaum (1977) sparked considerable interest in symbolic machine-learning methods as a potential solution. Early work on ►decision tree induction around this time was often driven by problems from computer chess, a challenging domain by the standards of the time due to relatively large data sets and the complexity of the target hypotheses. In a landmark paper on his ID3 ►decision tree learning algorithm, Quinlan (1983) reported experiments on learning to classify positions in a four-piece chess endgame as winnable (or not) within a certain number of moves ("lost N-ply"). A set of attributes was defined as *inadequate* for a classification task if two objects belonging to different classes had identical values for each attribute. He concluded that "almost all the effort (for a non chess-player, at least) must be devoted to finding attributes that are adequate for the classification problem being tackled".

The problem is that the effort of developing the set of attributes becomes disproportionate to the time taken to do the induction. Quinlan (1983) reported durations of three weeks and two man-months, respectively, to define an adequate set of attributes for the "lost

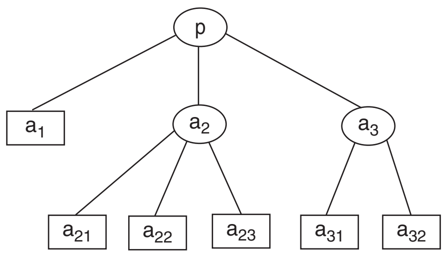
2-ply" and "lost 3-ply" experiments. In contrast, the implementation of ID3 used in that work induced the ▶decision trees in 3 s and 34 s, respectively. It is worth noting that the more complex problem of "lost 4-ply" was abandoned due to the difficulty of developing an adequate set of attributes.

Although Quinlan's experiments with ID3 produced exact ▶classifiers for his chess problems, the resulting ▶decision trees were too large to be comprehensible to domain experts. This is a serious drawback when machine learning is used with the goal of installing learned ▶rules in an expert system, since the system cannot provide understandable explanations. Shapiro and Niblett (1982) proposed structured induction as a solution to this problem, and the method was developed in Shapiro's PhD thesis (Shapiro, 1987) motivated by expert systems development.

Structure of Learning System

Structured induction is essentially a two-stage process, comprising a top-down decomposition of the problem into a solution structure, followed by a bottom-up series of ▶classifier learning steps, one for each of the subproblems. A knowledge engineer and a domain expert are required to collaborate at each stage, with the latter acting as a source of examples. The use of machine learning to avoid the knowledge acquisition bottleneck is based on the finding that although domain experts find it difficult to express general and accurate ▶rules for a problem, they are usually able to generate tutorial examples in an attribute-value formalism from which ▶rules can be generalized automatically. The key insight of structured induction is that the task of specifying an attribute and its value set can be treated as a subproblem of the learning task, and solved in the same way.

The approach can be illustrated by a simple example using the structure shown in Fig. 1. Suppose the task is to learn a model for some concept **p**. Suppose further that the domain expert proposes three attributes $\mathbf{a}_1$, $\mathbf{a}_2$, and $\mathbf{a}_3$ as adequate for the classification of **p**. Now the domain expert consults with the knowledge engineer and it is decided that while attribute $\mathbf{a}_1$ is directly implementable, the other two are not. An attribute that is directly implementable by a knowledge engineer is referred to as *primitive* for the domain.



Structured Induction. Figure 1. A schematic diagram of a model learned by structured induction (after Shapiro, 1987). Concepts to be learned are shown in ovals, and primitive attributes in boxes. The top-level concept p is defined in terms of the primitive attribute $\mathbf{a}_1$ and two sub-concepts $\mathbf{a}_2$ and $\mathbf{a}_3$. Each of the two sub-concepts are further decomposed into sets of primitive attributes, $\mathbf{a}_{21\ldots3}$ and $\mathbf{a}_{31\ldots2}$

The other attributes become sub-concepts $\mathbf{a}_2$ and $\mathbf{a}_3$, and each in turn is addressed by the domain expert. In this case, three attributes are proposed as most relevant to the solution of $\mathbf{a}_2$, and two for $\mathbf{a}_3$. Since all of these attributes are found to be primitive, the top-down problem decomposition stage is therefore complete.

The domain expert, having proposed a set of primitive attributes for a sub-concept, say $\mathbf{a}_3$, is now required to provide a set of classified examples defined in terms of values for attributes $\mathbf{a}_{31}$ and $\mathbf{a}_{32}$. Given these examples, the knowledge engineer will run a learning algorithm to induce a ▶classifier such as a ▶decision tree. The domain expert will then inspect the ▶classifier and can optionally refine it by supplying further examples, until they are satisfied that it completely and correctly defines the sub-concept $\mathbf{a}_3$. This process is repeated in a bottom-up fashion for all sub-concepts. At every level of the hierarchy, once all sub-concepts have been defined, they are now directly executable ▶classifiers and can be treated in the same way as primitive attributes and used for learning. The structured induction solution is complete once an acceptable ▶classifier has been learned for the top-level concept, **p** in this example.

Structured Versus Unstructured Induction

On two chess end-game domains, Shapiro (1987) showed that structured induction could generate more compact trees from fewer examples compared with an unstructured approach. To quantify this improvement, Shapiro made an analysis based on Michie's finite message information theory (Michie, 1982). This showed that on one of the domains, the information gain contributed by the structured induction approach over learning unstructured trees from the same set of examples was 84%. Essentially, this is because the structure devised by the domain expert in collaboration with the knowledge engineer provides a context for each of the induction tasks required. Since within this context only a subset of the complete attribute set is used to specify a sub-concept, it suffices to obtain only sufficient examples to learn a model for that sub-concept. However, without the benefit of such contextual restrictions the task of learning a complete solution can require considerably more examples. Shapiro's analysis is an attempt to quantify the relative increase in information per example in structured versus unstructured induction.

Related Work

While induction can bypass the knowledge acquisition bottleneck, in structured induction the process of acquiring the structure in collaboration with a domain expert can become a new bottleneck. In an attempt to avoid this, a number of researchers have attempted to develop methods whereby the structure, as well as the sub-concept models can be learned automatically.

Muggleton (1987) introduced ►inverse resolution as an approach to learning structured ►rule sets in a system called Duce. As part of this process, a domain expert is required to provide names for new sub-concepts or *predicates* that are proposed by the learning algorithm. A domain expert is also required to confirm learned ►rules. Both these roles are similar to those required of the expert in constructive induction, but the key difference is that the learning algorithm is now the source of both the structure and the ►rules. Duce was applied to one of the chess end-game domains used in Shapiro's study (Shapiro, 1987) and found a solution that was less compact, but still accepted as comprehensible by a chess expert.

The Duce system searches for commonly occurring subsets of attribute-value pairs within ►rules, and uses these to construct new sub-concept definitions. Many approaches have been developed using related methods to learn new sub-concepts in the context of ►decision tree or ►rule learning; some examples include Pagallo and Haussler (1990), Zheng (1995), and Zhang and Honavar (2003). Gaines (1996) proposed EDAGs (exception directed acyclic graphs) as a generalization of both ►decision trees and ►rules with exceptions, and reported EDAG representations of chess end-game ►classifiers that were more comprehensible than either ►rules or ►decision trees. Zupan, Bohanec, Demsar, and Bratko (1999) developed a system named HINT designed to learn a model represented as a concept hierarchy based on methods of function decomposition. Inverse resolution as used in Duce has been generalized to first-order logic representations in the field of inductive logic programming. In this framework, the construction of new intermediate concepts is referred to as ►*predicate invention*, but to date this remains a largely open problem. More recently, much of the interest in ►representation change has focused on approaches like support vector machines, where the so-called kernel trick enables the use of *implicit* ►feature construction (Shawe-Taylor and Cristianini, 2004).

Cross References

►Classifier
►Constructive Induction
►Decision Tree
►Feature Construction
►Predicate Invention
►Rule Learning

Recommended Reading

Dahl, O. J., Dijkstra, E. W., & Hoare, C. A. R. (Eds.). (1972). *Structured programming*. London: Academic Press.

Feigenbaum, E. A. (1977). The art of artificial intelligence: Themes and case studies of knowledge engineering. In R. Reddy (Ed.), *Proceedings of the fifth international conference on artificial intelligence* (IJCAI77) (pp. 1014–1029). Los Altos, CA: William Kaufmann.

Gaines, B. (1996). Transforming rules and trees into comprehensible knowledge structures. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, & R. Uthurusamy (Eds.), *Advances in knowledge discovery and data mining* (pp. 205–226). Cambridge, MA: MIT Press.

Michie, D. (1982). Measuring the knowledge-content of expert programs. *Bulletin of the Institute of Mathematics and its Applications, 18*(11/12), 216–220.

Muggleton, S. (1987). Duce, an oracle-based approach to constructive induction. In *IJCAI 87* (pp. 287–292). Los Altos, CA: Kaufmann.

Pagallo, G., & Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine learning*, 5, 71–99.
Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess end games. In R. Michalski, J. Carbonnel, & T. Mitchell (Eds.), *Machine learning: An artificial intelligence approach*, (pp. 464–482). Palo Alto, CA: Tioga.
Razzak, M. A., Hassan, T., & Pettipher, R. (1984). Extran-7: A Fortran-based software package for building expert systems. In M. A. Bramer (Ed.), *Research and development in expert systems* (pp. 23–30). Cambridge: Cambridge University Press.
Shapiro, A., & Niblett, T. (1982). Automatic induction of classification rules for a chess endgame. In M. R. B. Clarke (Ed.), *Advances in computer chess* (Vol. 3, pp. 73–91). Pergamon: Oxford.
Shapiro, A. D. (1987). *Structured Induction in expert systems*. Wokingham: Turing Institute Press with Addison Wesley.
Shawe-Taylor, J., & Cristianini, N. (2004). *Kernel methods for pattern analysis*. Cambridge: Cambridge University Press.
Zhang, J., & Honavar, V. (2003). Learning decision tree classifiers from attribute value taxonomies and partially specified data. In *ICML-2003: Proceedings of the twentieth international conference on machine learning*, Menlo Park, CA: AAAI Press.
Zheng, Z. (1995). Constructing nominal X-of-N attributes. In *Proceedings of the fourteenth International joint conference on artificial intelligence (IJCAI, 95)* (pp. 1064–1070). Los Altos, CA: Morgan Kaufmann.
Zupan, B., Bohanec, M., Demsar, J., & Bratko, I. (1999). Learning by discovering concept hierarchies. *Artificial Intelligence, 109*, 211–242.

Subgroup Discovery

Definition

Subgroup discovery (Klösgen, 1996; Lavrač, Kavšek, Flach, & Todorovski, 2004) is an area of ►supervised descriptive rule induction. The subgroup discovery task is defined as given a population of individuals and a property of those individuals that we are interested in, find population subgroups that are statistically "most interesting," for example, are as large as possible and have the most unusual statistical (distributional) characteristics with respect to the property of interest.

Recommended Reading

Klösgen, W., (1996). Explora: A multipattern and multistrategy discovery assistant. In *Advances in knowledge discovery and data mining* (pp. 249–271). Cambridge: MIT Press.
Lavrač, N., Kavšek, B., Flach, P. A., & Todorovski, L. (2004). Subgroup discovery with CN2-SD. *Journal of Machine Learning Research, 5*, 153–188.

Sublinear Clustering

Artur Czumaj[1], Christian Sohler[2]
[1]University of Warwick, Coventry, UK,
[2]University of Paderborn,
Paderborn, Germany

Definition

Sublinear clustering describes the process of clustering a given set of input objects using only a small subset of the input set, which is typically selected by a random process. A solution computed by a sublinear clustering algorithm is an implicit description of the clustering (rather than a partition of the input objects), for example in the form of cluster centers. Sublinear clustering is usually applied when the input set is too large to be processed with standard clustering algorithms.

Motivation and Background

►Clustering is the process of partitioning a set of objects into subsets of similar objects. In machine learning, it is, for example, used in unsupervised learning to fit input data to a density model. In many modern applications of clustering, the input sets consist of billions of objects to be clustered. Typical examples include web search, analysis of web traffic, and spam detection. Therefore, even though many relatively efficient clustering algorithms are known, they are often too slow to cluster such huge inputs.

Since in some applications it is even not possible to cluster the entire input set, a new approach is needed to cope with very large data sets. The approach used in many different areas of science and engineering in this context is to *sample* a subset of the data and to analyze this sample instead of the whole data set. This is also the underlying method used in sublinear clustering. The main challenge and innovation in this area lies in the qualitative analysis of random sampling (in the form of approximation guarantees) and the design of *non uniform sampling* strategies that approximate the input set provably better than *uniform random sampling*.

Structure of the Learning System

In sublinear clustering a large input set of objects is to be partitioned into subsets of similar objects. Usually, this input is a point set P either in the Euclidean space or in the metric space. The clustering problem is specified by an objective function that determines the quality or cost of every possible clustering. The goal is to find a clustering of minimum cost/maximum quality. For example, given a set P of points in the Euclidean space the objective of ►k-means clustering is to find a set C of k centers that minimizes $\sum_{p \in P}(d(p,C))^2$, where $d(p,C)$ denotes the distance of p to the nearest center from C. Since usually the clustering problems are computationally hard ($\mathcal{NP}$-hard), one typically considers *approximate* solutions: instead of finding a clustering that optimizes the cost of the solution, one aims at a solution whose cost is close to the optimal one.

In sublinear algorithms a solution is computed for a small representative subset of objects, for example a random sample. The solution is represented implicitly, for example, in the form of cluster centers and it can be easily extended to the whole input point set. The quality of the output is analyzed *with respect to the original point set*. The challenge is to *design and analyze fast (sublinear-time) algorithms* that select a subset of objects that very well represent the entire input, such that the solution computed for this subset will also be a good solution for the original point set. This can be achieved by uniform and non uniform *random sampling* and the computation of *core-sets*, i.e., small weighted subsets of the input that approximate the input with respect to a clustering objective function.

Theory/Solution

Clustering Problems

For any point p and any set Q in a metric space (X,d), let $d(p,Q) = \min_{q \in Q} d(p,q)$. A point set P is *weighted* if there is a function w assigning a positive weight to each point in P.

Radius k-Clustering: Given a weighted set P of points in a metric space (X,d), find a set $C \subseteq P$ of k centers minimizing $\max_{p \in P} d(p,C)$.

Diameter k-Clustering: Given a weighted set P of points in a metric space (X,d), find a partition of P into k subsets $P_1,\ldots,P_k$, such that $\max_{i=1}^{k} \max_{p,q \in P_i} d(p,q)$ is minimized.

k-Median: Given a weighted set P of points in a metric space (X,d), find a set $C \subseteq P$ of k centers that minimizes $median(P,C) = \sum_{p \in P} w(p) \cdot d(p,C)$.

k-Means: Given a weighted set of points P in a metric space (X,d), find a set $C \subseteq P$ of k centers that minimizes $mean(P,C) = \sum_{p \in P} w(p) \cdot (d(p,C))^2$.

Random Sampling and Sublinear-Time Algorithms

Random sampling is perhaps the most natural approach to design sublinear-time algorithms for clustering. For the input set P, random sampling algorithm follows the following scheme:

```
1. Pick a random sample $S$ of points
2. Run an algorithm (possibly an approximation one)
   for (given kind of) clustering for $S$
3. Return the clustering induced by the solution for $S$
```

The running time and the quality of this algorithm depends on the size of the random sample S and of the running time and the quality of the algorithm for clustering of S. Because almost all interesting clustering problems are computationally intractable ($\mathcal{NP}$-hard), usually the second step of the sampling scheme uses an approximation algorithm. (An algorithm for a minimization problem is called a λ-approximation if it always returns a solution whose cost is at most λ times the optimum.)

While random sampling approach gives very simple algorithms, depending on the clustering problem at hand, it often finds a clustering of poor quality and it is usually difficult to analyze. Indeed, it is easy to see that random sampling has some serious limitations to obtain clustering of good quality. Even the standard assumption that the input is in metric space is not sufficient to obtain good quality of clustering because of the small clusters which are "hidden" for random sampling approach. (If there is a cluster of size $o(|P|/|S|)$ then with high probability the random sample set S will contain no point from that cluster.) Therefore, another important parameters used in the analysis is the *diameter* of the metric space Δ, which is $\Delta = \max_{p,q \in P} d(p,q)$.

Quality of Uniformly Random Sampling: The quality of random sampling for three basic clustering problems

(k-means, k-median, and min-sum k-clustering) have been analyzed in Ben-David (2004), Czumaj and Sohler (2007), and Mishra, Oblinger, and Pitt (2001). In these papers, generic methods of analyzing uniform random sampling have been obtained. These results assume that the input point sets are in a metric space and are unweighted (i.e., when the weight function w is always 1).

Theorem 1 *Let $\epsilon > 0$ be an approximation parameter. Suppose that an α-approximation algorithm for the k-median problem in a metric space is used in step* (2) *of the uniform sampling, where $\alpha \geq 1$* (Ben-David 2004; Czumaj & Sohler 2007, Mishra et al., 2001). *If we choose S to be of size at least*

$$c\alpha\left(k + \frac{\Delta}{\epsilon}\left(\alpha + k\ln(k\Delta\alpha/\epsilon)\right)\right)$$

for an appropriate constant c, then the uniform sampling algorithm returns a clustering C^* (of S) such that with probability at least 0.99 *the normalized cost of clustering for S satisfies*

$$\frac{median(S, C^*)}{|S|} \leq \frac{2(\alpha + 0.01)OPT(P)}{|P|} + \epsilon,$$

where $OPT(S) = \min_C median(P, C)$ is the minimum cost of a solution for k-median for P.

Similar results can be shown for the k-means problem, and also for min-sum k-clustering (cf. Czumaj & Sohler, 2007). For example, for k-means, with a sample S of size at least $c\alpha\left(k + (\Delta^2/\epsilon)\left(\alpha + k\ln(k\Delta^2\alpha/\epsilon)\right)\right)$, with probability at least 0.99 the normalized cost of k-means clustering for S satisfy

$$\frac{mean(S, C^*)}{|S|^2} \leq \frac{4(\alpha + 0.01)OPT(P)}{|P|^2} + \epsilon,$$

where $OPT(S) = \min_C mean(P, C)$ is the minimum cost of a solution for k-means for P.

Improvements of these bounds for the case when the input consists of points in Euclidean space are also discussed in Mishra et al. (2001), Czumaj and Sohler (2007) discuss also . For example, for k-median, if one takes S of size at least $c\alpha\,(k + \Delta k \ln(\Delta/\epsilon)/\epsilon)$, then with probability at least 0.99 the normalized cost of k-median clustering for S satisfies

$$\frac{median(S, C^*)}{|S|} \leq \frac{(\alpha + 0.01)OPT(P)}{|P|} + \epsilon \ ,$$

and hence the approximation ratio is better than that in Theorem 1 by factor of 2.

The results stated in Czumaj and Sohler (2007) allow to parameterize the constants 0.99 and 0.01 in the claims above.

Property Testing of the Quality of Clustering: Since most of the clustering problems are computationally quite expensive, in some situations it might be interesting not to find a clustering (or its succinct representation), but just to test if the input set has a good clustering.

Alon, Dar, Parnas, and Ron (2003) introduced the notion of approximate testing of good clustering. A point set P is *c-clusterable* if it has a clustering of the cost at most c, that is, $OPT(P) \leq c$. To formalize the notion of having no good clustering, one says a point set is *ϵ-far from* $(1 + \beta)$*c-clusterable*, if more than an ϵ-fraction of the points in P must be removed (or moved) to make the input set $(1 + \beta)c$-clusterable. With these definitions, the goal is to design fast algorithms that accept the input point sets P, which are c-clusterable, and reject with probability at least 2/3 inputs are ϵ-far from $(1 + \beta)c$-clusterable. If neither holds, then the algorithms may either accept or reject. The bounds for the testing algorithms are phrased in terms of *sample complexity*, that is, the number of sampled input points which are considered by the algorithm (e.g., by using random sampling).

Alon et al. (2003) consider two classical clustering problems in this setting: radius and diameter k-clusterings. If the inputs are drawn from an arbitrary metric space, then they show that to distinguish between input points sets that are c-clusterable and are ϵ-far from $(1 + \beta)c$-clusterable with $\beta < 1$, the sample complexity must be at least $\Omega(\sqrt{|P|/\epsilon})$. However, to distinguish between inputs that are c-clusterable and are ϵ-far from $2c$-clusterable, the sample complexity is only $O(\sqrt{k/\epsilon})$.

A more interesting situation is for the input points drawn from the Euclidean d-dimensional space. In that case, even a constant-time algorithms are possible.

Theorem 2 *For the radius k-clustering, one can distinguish between points sets in* R^d *that are c-clusterable from those* ϵ*-far from c-clusterable with the sample complexity* $\widetilde{O}(dk/\epsilon)$ (Alon et al., 2003) *(The* $\widetilde{O}$*-notation ignores logarithms in the largest occurrence of a variable;* $\widetilde{O}(f(n)) = O(f(n) \cdot (\log f(n))^{O(1)})$*.)*

Furthermore, for any $\beta > 0$*, one can distinguish between points sets in* R^d *that are c-clusterable from those* ϵ*-far from* $(1+\beta)c$*-clusterable with the sample complexity* $\widetilde{O}(k^2/(\beta^2\epsilon))$.

Theorem 3 *For the diameter k-clustering, one can distinguish between points sets in* R^d *that are c-clusterable from those* ϵ*-far from* $(1+\beta)c$*-clusterable with the sample complexity* $\widetilde{O}(k^2 d(2/\beta)^{2d}/\epsilon)$ *(*Alon et al., 2003*).*

Core-Sets: Sublinear Space Representations with Applications

A *core-set* is a small weighted set of points S that provably approximates another point set P with respect to a given clustering problem (Bădoiu, Har-Peled, & Indyk, 2002). The precise definition of a core-set depends on the clustering objective function and the notion of approximation. For example, a coreset for the k-median problem can be defined as follows:

Definition 4 *A weighted point set S is called* ϵ-coreset *for a point set P for the k-median problem, if for every set C of k centers, we have* $(1-\epsilon) \cdot median(P, C) \leq median(S, C) \leq (1+\epsilon) \cdot median(P, C)$ (Har-Peled & Mazumdar, 2004).

A core-set as defined above is also sometimes called a *strong* core-set, because the cost of the objective function is approximately preserved for any set of cluster centers. In some cases it can be helpful to only require a weaker notion of approximation. For example, for some applications it is sufficient that the cost is preserved for a certain discrete set of candidate solutions. Such a core-set is usually called a *weak* core-set. In high-dimensional applications it is sometimes sufficient that the solution is contained in the low-dimensional subspace spanned by the core-set points.

Constructing a Core-Set: There are deterministic and randomized constructions for core-sets of an unweighted set P of n points in the R^d. Deterministic core-set constructions are usually based on the *movement paradigm*. The input points are moved to a set of few locations such that the overall movement is at most ϵ times the cost of an optimal solution. Then the set of points at the same location are replaced by a single point whose weight equals the number of these points. Since for the k-median problem the cost of any solution changes by at most the overall movement, this immediately implies that the constructed weighted set is an ϵ-core-set. For other similar problems more involved arguments can be used to prove the core-set property. Based on the movement paradigm, for k-median a core-set of size $O(k \log n/\epsilon^d)$ can be constructed efficiently (Har-Peled & Mazumdar, 2004).

Randomized core-set constructions are based on non uniform sampling. The challenge is to define a randomized process for which the resulting weighted point set is with high probability a core-set. Most randomized coreset constructions first compute a bi-criteria approximation C'. Then every point is sampled with probability proportional to its distance to the nearest center of C'. A point q is assigned a weight proportional to $1/p_q$, where p_q is the probability that p is sampled. For every fixed set C of k centers, the resulting sample is an unbiased estimator for $median(P, C)$. If the sample set is large enough, it approximates the cost of every possible set of k centers within a factor of $(1 \pm \epsilon)$. The above approach can be used to obtain a weak core-set of size independent of the size of the input point set and the dimension of the input space (Feldman, Monemizadeh, & Sohler, 2007). A related construction has been previously used to obtain a strong core-set of size $\widetilde{O}(k^2 \cdot d \cdot \log n/\epsilon^2)$. Both constructions have constant success probability that can be improved by increasing the size of the core-set.

Applications Core-sets have been used to obtain improved approximation algorithms for different variants of clustering problems. Since the core-sets are of sublinear size and they can be constructed in sublinear time, they can be used to obtain sublinear-time approximation algorithms for a number of clustering problems.

Another important application is clustering of data streams. A data stream is a long sequence of data that typically does not fit into main memory, for example, a sequence of packet headers in IP traffic monitoring. To analyze data streams we need algorithms that extract information from a stream without storing all of the observed data. Therefore, in the data streaming model algorithms are required to use $\log^{O(1)} n$ bits of memory. For core-sets, a simple but rather general technique is known, which turns a given construction of a strong core-set into a data streaming algorithm, i.e., an algorithm that processes the input points sequentially and uses only $\log^{O(1)}$ space (for constant k and ϵ) and computes a $(1+\epsilon)$-approximation for the optimal set of centers of the k-median clustering (Har-Peled & Mazumdar, 2004). Core-sets can also be used to improve the running time and stability of clustering algorithms like the k-means algorithm (Frahling & Sohler, 2006).

Recommended Reading

Alon, N., Dar, S., Parnas, M., & Ron, D. (2003). Testing of clustering. *SIAM Journal on Discrete Mathematics, 16*(3), 393–417.

Bădoiu, M., Har-Peled, S., & Indyk, P. (2002). Approximate clustering via core-sets. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, (pp. 250–257).

Ben-David, S. (2004). A framework for statistical clustering with a constant time approximation algorithms for k-median clustering. In *Proceedings of the 17th Annual Conference on Learning Theory (COLT)*, (pp. 415–426).

Chen, K. (2006). On k-median clustering in high dimensions. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, (pp. 1177–1185).

Czumaj, A., & Sohler, C. (2007). Sublinear-time approximation for clustering via random sampling. *Random Structures & Algorithms, 30*(1–2), 226–256.

Feldman, D., Monemizadeh, M., & Sohler, C. (2007). A PTAS for k-means clustering based on weak coresets. In *Proceedings of the 23rd Annual ACM Symposium on Computational Geometry (SoCG)*, (pp. 11–18).

Frahling, G., & Sohler, C. (2006). A fast k-means implementation using coresets. In *Proceedings of the 22nd Annual ACM Symposium on Computational Geometry (SoCG)*, (pp. 135–143).

Har-Peled, S. & Kushal, A. (2005). Smaller coresets for k-median and k-means clustering. In *Proceedings of the 21st Annual ACM Symposium on Computational Geometry (SoCG)*, (pp. 126–134).

Har-Peled, S., & Mazumdar, S. (2004). On coresets for k-means and k-median clustering. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, (pp. 291–300).

Meyerson, A., O'Callaghan, L., & Plotkin S.(July 2004). A k-median algorithm with running time independent of data size. *Machine Learning, 56*(1–3), (pp. 61–87).

Mishra, N., Oblinger, D., & Pitt, L. (2001). Sublinear time approximate clustering. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, (pp. 439–447).

Subspace Clustering

►Projective Clustering

Subsumption

Claude Sammut
The University of New South Wales,
Sydney NSW, Australia

Subsumption provides a syntactic notion of generality. Generality can simply be defined in terms of the cover of a concept. That is, a concept, C, is more general than a concept, C', if C covers at least as many examples as C' (see ►Learning as Search). However, this does not tell us how to determine, from their syntax, if one sentence in a concept description language is more general than another. When we define a *subsumption* relation for a language, we provide a syntactic analogue of generality (Lavrač & Džeroski, 1994). For example, θ-*subsumption* (Plotkin, 1970) is the basis for constructing generalization lattices in ►inductive logic programming (Shapiro, 1981). See ►Generality of Logic for a definition of θ-*subsumption*. An example of defining a subsumption relation for a domain specific language is in the LEX program (Mitchell, Utgoff, & Banerji, 1983), where an ordering on mathematical expressions is given.

Cross References

►Generalization
►Induction
►Learning as Search
►Logic of Generality

Recommended Reading

Lavrač, N., & Džeroski, S. (1994). *Inductive Logic Programming: Techniques and applications*. Chichester: Ellis Horwood.

Mitchell, T. M., Utgoff, P. E., & Banerji, R. B. (1983). Learning by experimentation: Acquiring and refining problem-solving heuristics. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach*. Palo Alto: Tioga.

Plotkin, G. D. (1970). A note on inductive generalization. In B. Meltzer & D. Michie (Eds.), *Machine intelligence* (Vol. 5, pp. 153–163). Edinburgh University Press.

Shapiro, E. Y. (1981). An algorithm that infers theories from facts. In *Proceedings of the seventh international joint conference on artificial intelligence, Vancouver* (pp. 446–451). Los Altos: Morgan Kaufmann.

Supersmoothing

►Local Distance Metric Adaptation

Supervised Descriptive Rule Induction

Petra Kralj Novak[1], Nada Lavrač[1,2], Geoffrey I. Webb[3]
[1]Jožef Stefan Institute, Ljubljana, Slovenia
[2]University of Nova Gorica, Nova Gorica, Slovenia
[3]Monash University, Clayton, VIC, Australia

Synonyms

SDRI

Definition

Supervised descriptive rule induction (SDRI) is a machine learning task in which individual patterns in the form of rules (see ►Classification rule) intended for interpretation are induced from data, labeled by a predefined property of interest. In contrast to standard ►supervised rule induction, which aims at learning a set of rules defining a classification/prediction model, the goal of SDRI is to induce individual descriptive patterns. In this respect SDRI is similar to ►association rule discovery, but the consequents of the rules are restricted to a single variable – the property of interest – and, except for the discrete target attribute, the data is not necessarily assumed to be discrete.

Supervised descriptive rule induction assumes a set of training examples, described by attributes and their values and a selected attribute of interest (called the target attribute). Supervised descriptive rule induction induces rules that may each be interpreted independently of the others. Each rule is a ►local model, covering a subset of training examples, that captures a local relationship between the target attribute and the other attributes.

Induced descriptive rules are mainly aimed at human interpretation. More specifically, the purposes of supervised descriptive rule induction are to allow the user to gain insights into the data domain and to better understand the phenomena underlying the data.

Motivation and Background

Symbolic data analysis techniques aim at discovering comprehensible patterns or ►models in data. They can be divided into techniques for *predictive induction*, where models, typically induced from class labeled data, are used to predict the class value of previously unseen examples, and *descriptive induction*, where the aim is to find comprehensible patterns, typically induced from unlabeled data. Until recently, these techniques have been investigated by two different research communities: predictive induction mainly by the machine learning community, and descriptive induction mainly by the data mining community.

Data mining tasks where the goal is to find comprehensible patterns from labeled data have been addressed by both the machine learning and the data mining community independently. The data mining community, using the ►association rule learning perspective, adapted association rule learners like ►Apriori (Agrawal, Mannila, Srikant, Toivonon, & Inkeri Verkamo, 1996) to perform tasks on labeled data, like class association rule learning (Jovanovski & Lavrač, 2001; Liu, Hsu, & Ma, 1998), as well as ►contrast set mining (Bay & Pazzani, 2001) and ►emerging pattern mining (Dong & Li, 1999). On the other hand, the machine learning community, which traditionally focused on the induction of ►rule sets from labeled data for the purposes of classification, turned to building individual rules for exploratory data analysis and interpretation. This is the goal of the task named ►subgroup discovery (Wrobel, 1997). These are the main areas of supervised descriptive rule induction. All these areas deal with finding comprehensible rules from class labeled data. However, the methods used and the interpretation of the results differ slightly from approach to approach. Other related approaches include change mining, mining of closed sets for labeled data, exception rule mining, bump hunting, quantitative association rules, and impact rules. See Kralj Novak, Lavrač, and

Webb (2009) for a more detailed survey of supervised descriptive rule induction.

Structure of the Learning System

Supervised descriptive rule induction assumes that there is data with the property of interest defined by the user. Let us illustrate supervised descriptive rule induction using data from Table 1, a very small artificial sample data set, adapted from Quinlan (1986), which contains the results of a survey on 14 individuals, concerning the approval or disproval of an issue analyzed in the survey. Each individual is characterized by four attributes that encode rudimentary information about the sociodemographic background. The last column (Approved) is the designated property of interest, encoding whether the individual approved or disproved the issue. Unlike predictive induction, where the aim is to find a predictive model, the goal of supervised descriptive rule induction is to find local patterns in form of individual rules describing individuals that are likely to approve or disprove the issue, based on the four demographic characteristics.

Figure 1 shows six descriptive rules, found for the sample data using the Magnum Opus (Webb, 1995) rule learning software. These rules were found using

Supervised Descriptive Rule Induction. Table 1 A Sample Database

Education	Marital Status	Sex	Has Children	Approved
Primary	Single	Male	No	No
Primary	Single	Male	Yes	No
Primary	Married	Male	No	Yes
University	Divorced	Female	No	Yes
University	Married	Female	Yes	Yes
Secondary	Single	Male	No	No
University	Single	Female	No	Yes
Secondary	Divorced	Female	No	Yes
Secondary	Single	Female	Yes	Yes
Secondary	Married	Male	Yes	Yes
Primary	Married	Female	No	Yes
Secondary	Divorced	Male	Yes	No
University	Divorced	Female	Yes	No
Secondary	Divorced	Male	No	Yes

```
MaritalStatus = single AND Sex = male → Approved = no
Sex = male → Approved = no
Sex = female → Approved = yes
MaritalStatus = married → Approved = yes
MaritalStatus = divorced AND HasChildren = yes → Approved = no
MaritalStatus = single → Approved = no
```

Supervised Descriptive Rule Induction. Figure 1. Selected descriptive rules, describing individual patterns in the data of Table 1

S

the default settings except that the critical value for the statistical test was relaxed. This set of descriptive rules differs from a typical predictive rule set in several ways. The first rule is redundant with respect to the second. The first rule is included as a strong pattern (all three single males do not approve) whereas the second is weaker but more general (four out of seven males do not approve, which is not highly predictive, but accounts for four out of all five respondents who do not approve). Most predictive systems would include only one of these rules, but either or both of them may be of interest to someone trying to understand the data, depending on the specific application. This particular approach to descriptive pattern discovery does not attempt to guess which of the more specific or more general patterns will be more useful to the end user. Another difference between predictive and descriptive rules is that the predictive approach often includes rules for the sake of completeness, while some descriptive approaches make no attempt at completeness, as they assess each pattern on its individual merits.

Exactly which rules will be induced by a supervised descriptive rule induction algorithm depends on the task definition, the selected algorithm, as well as the user-defined constraints concerning minimal rule support, precision, etc. Different learning approaches and heuristics have been proposed to induce supervised descriptive rules.

Applications

Applications of supervised descriptive rule induction are widely spread. See Kralj Novak et al. (2009) for a detailed survey.

►Subgroup discovery has been used in numerous real-life applications. Medical applications include the analysis of coronary heart disease and brain ischemia data analysis, as well as profiling examiners for sonographic examinations. Spatial subgroup mining applications include mining of census data and mining of vegetation data. There are also applications in marketing and analysis of shop floor data.

►Contrast set mining has been used with retail sales data and for designing customized insurance programs. It has also been used in medical applications to identify patterns in synchrotron X-ray data that distinguish tissue samples of different forms of cancerous tumor and for distinguishing between groups of brain ischemia patients.

►Emerging pattern mining has been mainly applied to the field of bioinformatics, more specifically to microarray data analysis. For example, an interpretable classifier based on simple rules that is competitive to the state of the art black-box classifiers on the acute lymphoblastic leukemia (ALL) microarray data set was built from emerging patterns. Another application was about finding groups of genes by emerging patterns in a ALL/acute myeloblastic leukemia (AML) data set and a colon tumor data set. Emerging patterns were also used together with the unexpected change approach and the added/perished rule to mine customer behavior.

Future Directions

A direction for further research is to decompose SDRI algorithms, preprocessing and evaluation methods into basic components and their reimplementation as connectable web services, which includes the definition of interfaces between SDRI services. For instance, this can include the adaptation and implementation of subgroup discovery techniques to solving open problems in the area of contrast set mining and emerging patterns. This would allow for the improvement of algorithms due to the cross-fertilization of ideas from different SDRI subareas.

Another direction for further research concerns complex data types and the use of background knowledge. The SDRI attempts in this direction include relational subgroup discovery approaches like algorithms Midos (Wrobel, 2001), RSD (Relational Subgroup Discovery) (Železný & Lavrač, 2006), and SubgroupMiner (Klösgen & May, 2002), which is designed for spatial data mining in relational space databases. The search for enriched gene sets (SEGS) method (Trajkovski, Lavrac, & Tolar, 2008) supports building rules when using specialized biological knowledge in the form of ontologies. It is a step toward semantically enabled creative knowledge discovery in the form of descriptive rules.

Cross References

►Apriori
►Association Rule Discovery

►Classification Rule
►Contrast Set Mining
►Emerging Pattern Mining
►Subgroup Discovery
►Supervised Rule Induction

Recommended Reading

Agrawal, R., Mannila, H., Srikant, R., Toivonon, H., & Inkeri Verkamo, A. (1996). Fast discovery of association rules. In *Advances in knowledge discovery and data mining* (pp. 307–328). Menlo Park: American Association for Artificial Intelligence.

Bay, S. D., & Pazzani, M. J. (2001). Detecting group differences: Mining contrast sets. *Data Mining and Knowledge Discovery, 5*(3), 213–246.

Dong, G., & Li, J. (1999). Efficient mining of emerging patterns: Discovering trends and differences. In *Proceedings of the fifth ACM SIGKDD international conference on knowledge discovery and data mining (KDD-99)* (pp. 43–52). New York: ACM.

Jovanovski, V., & Lavrač, N. (2001). Classification rule learning with APRIORI-C. In *Proceedings of the tenth Portuguese conference on artificial intelligence* (pp. 44–51). London: Springer.

Klösgen, W., & May, M. (2002). Spatial subgroup mining integrated in an object-relational spatial database. In *Proceedings of the sixth European conference on principles and practice of knowledge discovery in databases (PKDD-02)* (pp. 275–286). London: Springer.

Kralj Novak, P. Lavrač, N., & Webb, G. I. (February 2009). Supervised descriptive rule discovery: A unifying survey of contrast set, emerging pattern and subgroup mining. *Journal of Machine Learning Research, 10*, 377–403. Available at: http://www.jmlr.org/papers/volume10/kralj-novak09a/kraljnovak09a.pdf.

Liu, B., Hsu, W., & Ma, Y. (1998). Integrating classification and association rule mining. In *Proceedings of the fourth international conference on knowledge discovery and data mining (KDD-98)* (pp. 80–86).

Trajkovski, I., Lavrac, N., & Tolar, J. (2008). SEGS: Search for enriched gene sets in microarray data. *Journal of Biomedical Informatics*, 41(4), 588–601.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1*(1), 81–106.

Webb, G. I. (1995). OPUS: An efficient admissible algorithm for unordered search. *Journal of Artificial Intelligence Research*, 3, 431–465.

Wrobel, S. (1997). An algorithm for multi-relational discovery of subgroups. In *Proceedings of the first European conference on principles of data mining and knowledge discovery (PKDD-97)* (pp. 78–87). London: Springer.

Wrobel, S. (2001). Inductive logic programming for knowledge discovery in databases. In S. Dzeroski & N. Lavrac (Eds.), *Relational data mining* (Chap. 4, pp. 74–101). Berlin: Springer.

Železný, F., & Lavrac, N. (2006). Propositionalization-based relational subgroup discovery with RSD. *Machine Learning*, 62, 33–63.

Supervised Learning

Definition

Supervised learning refers to any machine learning process that learns a function from an input type to an output type using data comprising examples that have both input and output values. Two typical examples of supervised learning are ►classification learning and ►regression. In these cases, the output types are respectively categorical (the classes) and numeric. Supervised learning stands in contrast to ►unsupervised learning, which seeks to learn structure in data, and to ►reinforcement learning in which sequential decision-making policies are learned from reward with no examples of "correct" behavior.

Cross References

►Reinforcement Learning
►Unsupervised Learning

Support Vector Machines

Xinhua Zhang
Australian National University, NICTA London Circuit, Canberra, Australia

Definition

Support vector machines (SVMs) are a class of linear algorithms that can be used for ►classification, ►regression, density estimation, novelty detection, and other applications. In the simplest case of two-class classification, SVMs find a hyperplane that separates the two classes of data with as wide a margin as possible. This leads to good generalization accuracy on unseen data, and supports specialized optimization methods that allow SVM to learn from a large amount of data.

Motivation and Background

Over the past decade, maximum margin models especially SVMs have become popular in machine learning. This technique was developed in three major steps. First, assuming that the two classes of training examples can be separated by a hyperplane, Vapnik and Lerner proposed in 1963 that the optimal hyperplane is the one that separates the training examples with the widest margin. From the 1960s to 1990s, Vapnik and Chervonenkis developed the Vapnik–Chervonenkis theory, which justifies the maximum margin principle from a statistical point of view. Similar algorithms and optimization techniques were proposed by Mangasarian in 1965.

Second, Boser, Guyon, and Vapnik (1992) incorporated kernel function into the maximum margin models, and their formulation is close to the currently popular form of SVMs. Before that, Wahba (1990) also discussed the use of kernels. Kernels allow SVM to implicitly construct the optimal hyperplane in the feature space, and the resulting nonlinear model is important for modeling real data.

Finally, in case the training examples are not linearly separable, Cortes and Vapnik (1995) showed that the soft margin can be applied, allowing some examples to violate the margin condition.

On the theoretical side, Shawe-Taylor, Bartlett, Williamson, and Anthony (1998) gave the first rigorous statistical bound on the generalization of hard margin SVMs. Shawe-Taylor and Cristianini (2000) gave statistical bounds on the generalization of soft margin algorithms and for the regression case.

In reality SVMs became popular thanks to its significantly better empirical performance than the neural networks. By incorporating transform invariances, the SVMs developed at AT&T achieved the highest accuracy on the MNIST benchmark set (a handwritten digit recognition problem). Joachims (1998) also showed clear superiority of SVMs on text categorization. Afterward, SVMs have been shown effective in many applications including computer vision, natural language, bioinformatics, and finance.

Theory

SVMs have a strong mathematical basis and are closely related to some well-established theories in statistics. They not only try to correctly classify the training data, but also maximize the margin for better generalization performance. This formulation leads to a separating hyperplane that depends only on the (usually small fraction of) data points that lie on the margin, which are called support vectors. Hence the whole algorithm is called *support vector machine*. In addition, since real–world data analysis problems often involve nonlinear dependencies, SVMs can be easily extended to model such nonlinearity by means of positive semidefinite kernels. Moreover, SVMs can be trained via quadratic programming, which (a) makes theoretical analysis easier, and (b) provides much convenience in designing efficient solvers that scale for large datasets. Finally, when applied to real-world data, SVMs often deliver state-of-the-art performance in accuracy, flexibility, robustness, and efficiency.

Optimal Hyperplane for Linearly Separable Examples

Consider the training set $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$ where $\boldsymbol{x}_i \in \mathbb{R}^p$ is the input feature vector for the i-th example, and $y_i \in \{1, -1\}$ is its corresponding label indicating whether the example is positive ($y_i = +1$) or negative ($y_i = -1$). To begin with, we assume that the set of positive and negative examples are linearly separable, that is, there exists a function $f(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$ where $\boldsymbol{w} \in \mathbb{R}^p$ (called the weight vector) and $b \in \mathbb{R}$ (called bias) such that

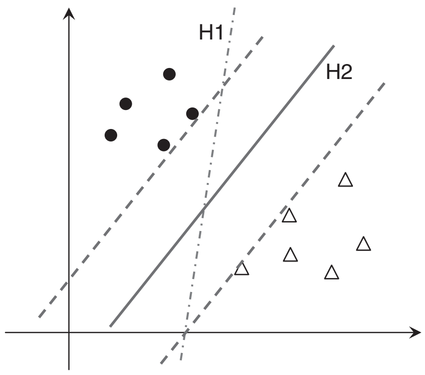
$$\begin{aligned} \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b > 0 \quad & \text{for } y_i = +1 \\ \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b < 0 \quad & \text{for } y_i = -1. \end{aligned}$$

We call $\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b = 0$ the decision hyperplane and in fact, there can exist multiple hyperplanes that separate the positive and negative examples, see Fig. 1. However, they are not created equal. Associated with each such hyperplane is a notion called *margin*, defined as the distance between the hyperplane and the closest example. SVM aims to find the particular hyperplane that maximizes the margin.

Mathematically, it is easy to check that the distance from a point $\boldsymbol{x}_i$ to a hyperplane $\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b = 0$ is $\|\boldsymbol{w}\|^{-1} |\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b|$. Therefore, SVM seeks for the optimal $\boldsymbol{w}, b$ of the following optimization problem:

$$\underset{\boldsymbol{w} \in \mathbb{R}^p,\ b \in \mathbb{R}}{\text{maximize}} \ \min_{1 \le i \le n} \frac{|\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b|}{\|\boldsymbol{w}\|},$$
$$\text{s.t.} \begin{cases} \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b > 0 & \text{if } y_i = +1 \\ \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b < 0 & \text{if } y_i = -1 \end{cases} \quad \forall i .$$

It is clear that if $(\boldsymbol{w}, b)$ is an optimal solution, then $(\alpha \boldsymbol{w}, \alpha b)$ is also an optimal solution for any $\alpha > 0$.



Support Vector Machines. Figure 1. Example of maximum margin separator. Both H1 and H2 correctly separate the examples from the two classes. But H2 has a wider margin than H1

Therefore, to fix the scale, we can equivalently set the numerator of the objective $\min_{1\le i\le n} |\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b|$ to 1, and minimize the denominator $\|\boldsymbol{w}\|$:

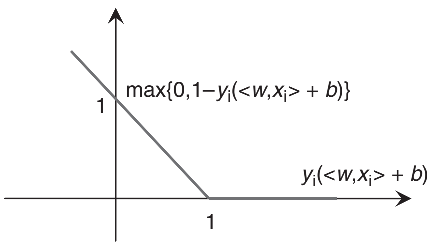
$$\underset{\boldsymbol{w}\in\mathbb{R}^p,\ b\in\mathbb{R}}{\text{minimize}} \ \|\boldsymbol{w}\|^2,$$
$$\text{s.t.} \begin{cases} \langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b \ge 1 & \text{if } y_i = +1 \\ \langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b \le -1 & \text{if } y_i = -1. \end{cases} \quad \forall i\,. \tag{1}$$

This is a linearly constrained quadratic program, which can be solved efficiently. Hence, it becomes the most commonly used (primal) form of SVM for the linearly separable case.

Soft Margins

In practice, most, if not all, datasets are not linearly separable, that is, no $\boldsymbol{w}$ and b can satisfy the constraints of the optimization problem (1). In this case, we will allow some data points to violate the margin condition, and penalize it accordingly. Mathematically, notice that the constraints in (1) can be equivalently written as $y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b) \ge 1$. Now we introduce a new set of nonnegative slack variables ξ_i into the constraints:

$$y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b) \ge 1 - \xi_i\,,$$



Support Vector Machines. Figure 2. Graph of hinge loss

and incorporate a penalty into the original objective to derive the soft margin SVM:

$$\underset{\boldsymbol{w},b,\xi_i}{\text{minimize}} \ \lambda\|\boldsymbol{w}\|^2 + \frac{1}{n}\sum_{i=1}^{n}\xi_i$$
$$\text{s.t. } y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b) \ge 1 - \xi_i, \text{ and } \xi_i > 0 \ \ \forall i\,. \tag{2}$$

$\lambda > 0$ is a trade-off factor. It is important to note that ξ_i can be written as $\xi_i = \max\{0, 1 - y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b)\}$, which is called hinge loss and is depicted in Fig. 2. This way, the optimization problem can be reformulated into an unconstrained non-smooth problem:

$$\underset{\boldsymbol{w}\in\mathbb{R}^p,\ b\in\mathbb{R}}{\text{minimize}} \ \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \frac{1}{n}\sum_{i=1}^{n}$$
$$\max\{0, 1 - y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b)\}\,. \tag{3}$$

Notice that $\max\{0, 1 - y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b)\}$ is also a convex upper bound of $\delta(y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b) > 0)$, where $\delta(x) = 1$ if x is true and 0 otherwise. Therefore, the penalty we use is a convex upper bound of the average training error. When the training set is actually separable, the soft margin problem (2) automatically recovers the hard margin problem (1) when λ is sufficiently large.

Dual Forms and Kernelization

As the constraints in the primal form (2) are not convenient to handle, people have conventionally resorted to the dual problem of (2). Following the standard procedures, one can derive the Lagrangian dual

$$\min_{\boldsymbol{\alpha}} \frac{1}{2\lambda}\sum_{i,j} y_i y_j \alpha_i \alpha_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j\rangle - \sum_i \alpha_i,$$
$$\text{s.t. } \alpha_i \in [0, n^{-1}], \text{ and } \sum_i y_i\alpha_i = 0\,. \tag{4}$$

which is again a quadratic program, but with much simpler constraints: box constraints plus a single linear equality constraint. To recover the primal solution $\boldsymbol{w}^*$ from the dual solution α_i^*, we have

$$\boldsymbol{w}^* = \sum_{i=1}^{n} \alpha_i^* y_i \boldsymbol{x}_i \,,$$

and the optimal bias b can be determined by using the duality conditions.

The training examples can be divided into three categories according to the value of α_i^*. If $\alpha_i^* = 0$, it means the corresponding training example does not affect the decision boundary, and in fact it lies beyond the margin, that is, $y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b) > 1$. If $\alpha_i^* \in (0, n^{-1})$, then the training example lies on the margin, that is, $y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b) = 1$. If $\alpha_i^* = n^{-1}$ it means the training example violates the margin, that is, $y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b) < 1$. In the latter two cases where $\alpha_i^* > 0$, the i-th training example is called a support vector.

In many applications, most α_i^* in the optimal solution are 0, which gives a sparse solution. As the final classifier depends only on those support vectors, the whole algorithm is named support vector machines.

From the dual problem (4), a key observation can be drawn that the feature of the training examples $\{\boldsymbol{x}_i\}$ influences training only via the inner product $\langle \boldsymbol{x}_i, \boldsymbol{x}_j\rangle$. Therefore, we can redefine the feature by mapping $\boldsymbol{x}_i$ to a richer feature space via $\phi(\boldsymbol{x}_i)$ and then compute the inner product there: $k(\boldsymbol{x}_i, \boldsymbol{x}_j) := \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j)\rangle$. Furthermore, one can even directly define k without explicitly specifying ϕ. This allows us to (a) implicitly use a rich feature space whose dimension can be infinitely high, and (b) apply SVM to non-Euclidean spaces as long as a kernel $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ can be defined on it. Examples include strings and graphs (Haussler, 1999), which have been widely applied in bioinformatics (Schölkopf, Tsuda, & Vert, 2004). Mathematically, the objective (4) can be kernelized into

$$\frac{1}{2\lambda}\sum_{i,j} y_i y_j \alpha_i \alpha_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) - \sum_i \alpha_i,$$
$$\text{s.t. } \alpha_i \in [0, n^{-1}], \text{ and } \sum_i y_i \alpha_i = 0 \,. \qquad (5)$$

However, now the $\boldsymbol{w}$ cannot be expressed just in terms of kernels because $\boldsymbol{w}^* = \sum_{i=1}^{n} \alpha_i^* y_i \phi(\boldsymbol{x}_i)$. Fortunately, when predicting on a new example $\boldsymbol{x}$ we again only require the inner product and hence use kernel only:

$$\langle \boldsymbol{w}^*, \boldsymbol{x}\rangle = \sum_{i=1}^{n} \alpha_i^* y_i \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x})\rangle = \sum_{i=1}^{n} \alpha_i^* y_i k(\boldsymbol{x}_i, \boldsymbol{x}) \,.$$

Commonly used kernels on $\mathbb{R}^n$ include polynomial kernels $(1 + \langle \boldsymbol{x}_i, \boldsymbol{x}_j\rangle)^d$, Gaussian RBF kernels $\exp(-\gamma \left\|\boldsymbol{x}_i - \boldsymbol{x}_j\right\|^2)$, Laplace RBF kernels $\exp(-\gamma \left\|\boldsymbol{x}_i - \boldsymbol{x}_j\right\|)$, etc. Kernels on strings and trees are usually based on convolution, which requires involved algorithms for efficient evaluation (Borgwardt, 2007; Haussler, 1999). More details can be found in the kernel section.

Optimization Techniques and Toolkits

The main challenge of optimization lies in scaling for large datasets, that is, n and p are large. Decomposition method based on the dual problem is the first popularly used method for solving large scale SVMs. For example, sequential minimal optimization (SMO) optimizes two dual variables α_i, α_j analytically in each iteration (Platt, 1999a). An SMO-type implementation is available in the LibSVM package http://www.csie.ntu.edu.tw/~cjlin/libsvm. Another popular package using decomposition methods is the SVM-light, available at http://svmlight.joachims.org. Coordinate descent in the dual is also effective and converges at linear rate. An implementation can be downloaded from http://www.csie.ntu.edu.tw/~cjlin/liblinear.

Primal methods are also popular, most of which are based on formulating the objective as a non-smooth objective function like (3). An important type is the subgradient descent method, which is similar to gradient descent but uses a subgradient due to the non-smooth objective. When the dataset is large, one can further use a random subset of training examples to efficiently compute the (approximate) subgradient, and algorithms exist that guarantee the convergence in probability. This is called stochastic subgradient descent, and in practice, it can often quickly find a reasonably good solution. A package that implements this idea can be found at http://leon.bottou.org/projects/sgd.

Finally, cutting plane and bundle methods are also effective (Tsochantaridis, Joachims, Hofmann, & Altun, 2005; Smola, Vishwanathan, & Le, 2007), and they are especially useful for generalized SVMs with structured outputs. An implementation is the bundle method for risk minimization (BMRM), available for download at http://users.rsise.anu.edu.au/~chteo/BMRM.html.

Applications

The above description of SVM focused on binary class classification. In fact, SVM, or the ideas of maximum margin and kernel, have been widely used in many other learning problems such as regression, ranking and ordinal regression, density estimation, novelty detection, quantile regression, and etc. Even in classification, SVM has been extended to the case of multi-class, multi-label, and structured output (Taskar, 2004; Tsochantaridis et al., 2005).

For multi-class classification and structured output classification where the possible label set $\mathcal{Y}$ can be large, maximum margin machines can be formulated by introducing a joint feature map ϕ on pairs of (x_i, y) $(y \in \mathcal{Y})$. Letting $\Delta(y_i, y)$ be the discrepancy between the true label y_i and the candidate label y, the primal form can be written as

$$\begin{aligned} &\text{minimize}_{w,\xi_i} \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n}\xi_i,\\ &\quad \text{s.t. } \langle w, \phi(x_i, y_i) - \phi(x_i, y)\rangle \geq \Delta(y_i, y) - \xi_i, \ \forall\ i, y, \end{aligned}$$

and the dual form is

$$\begin{aligned} &\text{minimize}_{\alpha_{i,y}} \frac{1}{2\lambda}\sum_{(i,y),(i',y')} \alpha_{i,y}\alpha_{i',y'}\langle \phi(x_i, y_i) - \phi(x_i, y)\\ &\qquad \phi(x_{i'}, y_{i'}) - \phi(x_{i'}, y')\rangle - \sum_{i,y}\Delta(y_i, y)\alpha_{i,y}\\ &\qquad \text{s.t.} \quad \alpha_{i,y} \geq 0,\ \forall\ i, y; \quad \sum_{y}\alpha_{i,y} = \frac{1}{n},\ \forall i. \end{aligned}$$

Again kernelization is convenient, by simply replacing all the inner products $\langle \phi(x_i, y), \phi(x_{i'}, y')\rangle$ with a joint kernel $k((x_i, y), (x_{i'}, y'))$. Further factorization using graphical models is possible, see Taskar (2004). Notice when $\mathcal{Y} = \{1, -1\}$, setting $\phi(x_i, y) = y\phi(x_i)$ recovers the binary SVM formulation. Effective methods to optimize the dual objective include SMO, exponentiated gradient descent, mirror descent, cutting plane, or bundle methods.

In general, SVMs are not trained to output the odds of class membership, although the posterior probability is desired to enable post-processing. Platt (1999b) proposed training an SVM, and then train the parameters of an additional sigmoid function to map the SVM outputs into probabilities. A more principled approach is the relevance vector machine, which has an identical functional form to the SVMs and uses Bayesian inference to obtain sparse solutions for probabilistic classification.

As mentioned above, the hinge loss used in SVM is essentially a convex surrogate of the misclassification loss, that is, 1 if the current weight $\boldsymbol{w}$ misclassifies the training example and 0 otherwise. Minimizing the misclassification loss is proved NP-hard, so for computational convenience continuous convex surrogates are used, including hinge loss, exponential loss, and logistic loss. Their statistical properties are studied by Jordan, Bartlett, and McAuliffe (2003). For hinge loss, it has the significant merit of sparsity in the dual, which leads to robustness and good generalization performance.

SVMs have been widely applied in real-world problems. In history, its first practical success was gained in handwritten digit recognition. By incorporating transform invariances, the SVMs developed at AT&T achieved the highest accuracy on the MNIST benchmark set. It has also been very effective in computer vision applications such as object recognition and detection. With the special advantage in handling high-dimensional data, SVMs have witnessed wide application in bioinformatics such as microarray processing (Schölkopf et al., 2004), and natural language processing like named entity recognition, part-of-speech tagging, parsing, and chunking (Joachims, 1998; Taskar, 2004).

Cross References

►Kernel Methods
►Radial Basis Function Networks

Further Reading

A comprehensive treatment of SVMs can be found in Schölkopf and Smola (2002) and Shawe-Taylor and Cristianini (2004). Some important recent developments of SVMs for structured output are collected in Bakir, Hofmann, Schölkopf, Smola, Taskar, and Vishwanathan (2007). As far as applications are concerned, see Lampert (2009) for computer vision and Schölkopf et al. (2004) for bioinformatics. Finally, Vapnik (1998) provides the details on statistical learning theory.

Recommended Reading

Bakir, G., Hofmann, T., Schölkopf, B., Smola, A., Taskar, B., & Vishwanathan, S. V. N. (2007). *Predicting structured data*. Cambridge: MIT Press.

Borgwardt, K. M. (2007). *Graph Kernels*. Ph.D. thesis, Ludwig-Maximilians-University, Munich, Germany.

Boser, B., Guyon, I., & Vapnik, V. (1992). A training algorithm for optimal margin classifiers. In D. Haussler (Ed.), *Proceedings of annual conference computational learning theory* (pp. 144–152). Pittsburgh: ACM Press.

Cortes, C., & Vapnik, V. (1995). Support vector networks. *Machine Learning*, *20*(3), 273–297.

Haussler, D. (1999). *Convolution kernels on discrete structures* (Tech. Rep. UCS-CRL-99-10). University of California, Santa Cruz.

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of the European conference on machine learning* (pp. 137–142). Berlin: Springer.

Jordan, M. I., Bartlett, P. L., & McAuliffe, J. D. (2003). *Convexity, classification, and risk bounds* (Tech. Rep. 638). University of California, Berkeley.

Lampert, C. H. (2009). Kernel methods in computer vision. *Foundations and Trends in Computer Graphics and Vision*, *4*(3), 193–285.

Platt, J. C. (1999a). Fast training of support vector machines using sequential minimal optimization. In *Advances in kernel methods—support vector learning* (pp. 185–208). Cambridge, MA: MIT Press.

Platt, J. C. (1999b). Probabilities for sv machines. In A. J. Smola, P. L. Bartlett, B. Schölkopf, & D. Schuurmans, (Eds.), *Advances in large margin classifiers* (pp. 61–74). Cambridge: MIT Press.

Schölkopf, B., & Smola, A. (2002). *Learning with kernels*. Cambridge: MIT Press.

Schölkopf, B., Tsuda, K., & Vert, J.-P. (2004). *Kernel methods in computational biology*. Cambridge: MIT Press.

Shawe-Taylor, J., & Cristianini, N. (2000). Margin distribution and soft margin. In A. J. Smola, P. L. Bartlett, B. Schölkopf, & D. Schuurmans, (Eds.), *Advances in large margin classifiers* (pp. 349–358). Cambridge: MIT Press.

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

Shawe-Taylor, J., Bartlett, P. L., Williamson, R. C., & Anthony, M. (1998). Structural risk minimization over data-dependent hierarchies. *IEEE Transactions on Information Theory*, *44*(5), 1926–1940.

Smola, A., Vishwanathan, S. V. N., & Le, Q. (2007). Bundle methods for machine learning. In D. Koller, & Y. Singer, (Eds.), *Advances in neural information processing systems* (Vol. 20). Cambridge: MIT Press.

Taskar, B. (2004). *Learning structured prediction models: A large margin approach*. Ph.D. thesis, Stanford University.

Tsochantaridis, I., Joachims, T., Hofmann, T., & Altun, Y. (2005). Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6, 1453–1484.

Vapnik, V. (1998). *Statistical learning theory*. New York: Wiley.

Wahba, G. (1990). *Spline models for observational data. CBMS-NSF regional conference series in applied mathematics* (Vol. 59). Philadelphia: SIAM.

Swarm Intelligence

Swarm intelligence is the discipline that studies the collective behavior of systems composed of many individuals that interact locally with each other and with their environment and that rely on forms of decentralized control and self-organization. Examples of such systems are colonies of ants and termites, schools of fish, flocks of birds, herds of land animals, and also some artifacts, including swarm robotic systems and some computer programs for tackling optimization problems such as ▶ant colony optimization and ▶particle swarm optimization.

Symbolic Dynamic Programming

Scott Sanner[1], Kristian Kersting[2]
[1]Statistical Machine Learning Group,
NICTA, Canberra, ACT, Australia
[2]Fraunhofer IAIS,
Sankt Augustin, Germany

Synonyms

Dynamic programming for relational domains; Relational dynamic programming; Relational value iteration; SDP

Definition

Symbolic dynamic programming (SDP) is a generalization of the ▶dynamic programming technique for solving ▶Markov decision processes (MDPs) that exploits the symbolic structure in the solution of relational and first-order logical MDPs through a lifted version of dynamic programming.

Motivation and Background

Decision-theoretic planning aims at constructing a policy for acting in an uncertain environment that maximizes an agent's expected utility along a sequence of steps. For this task, Markov decision processes (MDPs) have become the standard model. However, classical dynamic programming algorithms for solving MDPs require explicit state and action enumeration, which is often impractical: the number of states and actions grows very quickly with the number of domain objects and relations. In contrast, SDP algorithms seek to avoid explicit state and action enumeration through the symbolic representation of an MDP and a corresponding symbolic derivation of its solution, such as a value function. In essence, SDP algorithms exploit the symbolic structure of the MDP representation to construct a minimal logical partition of the state space required to make all necessary value distinctions.

Theory and Solution

Consider an agent acting in a simple variant of the BoxWorld problem. There are several cities such as *London*, *Paris* etc., trucks $truck_1$, $truck_2$ etc., and boxes box_1, box_2 etc. The agent can load a box onto a truck or unload it and can drive a truck from one city to another. Only when a particular box, say box box_1, is in a particular city, say *Paris*, the agent receives a positive reward. The agent's learning task is now to find a policy for action selection that maximizes its reward over the long term.

A great variety of techniques for solving such decision-theoretic planning tasks have been developed over the last decades. Most of them assume atomic representations, which essentially amounts to enumerating all unique configurations of trucks, cities, and boxes. It might then be possible to learn, for example, that taking action *action234* in state *state42* is worth 6.2 and leads to state *state654321*. Atomic representations are simple, and learning can be implemented using simple lookup tables. These lookup tables, however, can be intractably large as atomic representations easily explode. Furthermore, they do not easily generalize across different numbers of domain objects (We use the term *domain* in the first-order logical sense of an object universe. The term should not be confused with a planning *problem* such as BoxWorld or BlocksWorld.).

In contrast, SDP assumes a relational or first-order logical representation of an MDP (as given in Fig. 1) to exploit the existence of domain objects, relations over these objects, and the ability to express objectives and action effects using quantification.

It is then possible to learn that to get box b to *paris*, the agent drives a truck to the city of b, loads box_1 on the truck, drives the truck to *Paris*, and finally unloads the box box_1 in *Paris*. This is essentially encoded in the symbolic value function shown in Fig. 2, which was computed by discounting rewards t time steps into the future by 0.9^t. The key features to note here are the state and action abstraction in the value and policy representation that are afforded by the first-order specification and solution of the problem. That is, this solution does not refer to any specific set of domain objects, such as $City = \{paris, berlin, london\}$, but rather it provides a solution for *all possible domain object instantiations*. And while classical dynamic programming techniques could never solve these problems for large domain instantiations (since they would have to enumerate all states and actions), a domain-independent SDP solution to this particular problem is quite simple due to the power of state and action abstraction.

Background: Markov Decision Processes (MDPs)

In the MDP (Puterman, 1994) model, an agent is assumed to fully observe the current state and choose an action to execute from that state. Based on that state and action, nature then chooses a next state according to some fixed probability distribution. In an infinite-horizon MDP, this process repeats itself indefinitely. Assuming there is a reward associated with each state and action, the goal of the agent is to maximize the expected sum of discounted rewards received over an infinite horizon (Although we do not discuss it here, there are other alternatives to discounting such as averaging the reward received over an infinite horizon.). This criterion assumes that a reward received t steps in the future is discounted by γ^t, where γ is a discount factor satisfying $0 \leq \gamma < 1$. The goal of the agent is to choose its actions in order to maximize the expected, discounted future reward in this model.

- *Domain Object Types (i.e., sorts)*: Box, Truck, City = $\{paris, \ldots\}$
- *Relations (with parameter sorts)*:
  $BoxIn(Box, City)$, $TruckIn(Truck, City)$, $BoxOn(Box, Truck)$
- *Reward*: if $\exists b.BoxIn(b, paris)$ 10 else 0
- *Actions (with parameter sorts)*:
  - $load(Box : b, Truck : t, City : c)$:
    - Success Probability: if $(BoxIn(b,c) \wedge TruckIn(t,c))$ then .9 else 0
    - Add Effects on Success: $\{BoxOn(b,t)\}$
    - Delete Effects on Success: $\{BoxIn(b,c)\}$
  - $unload(Box : b, Truck : t, City : c)$:
    - Success Probability: if $(BoxOn(b,t) \wedge TruckIn(t,c))$ then .9 else 0
    - Add Effects on Success: $\{BoxIn(b,c)\}$
    - Delete Effects on Success: $\{BoxOn(b,t)\}$
  - $drive(Truck : t, City : c_1, City : c_2)$:
    - Success Probability: if $(TruckIn(t,c_1))$ then 1 else 0
    - Add Effects on Success: $\{TruckIn(t,c_2)\}$
    - Delete Effects on Success: $\{TruckIn(t,c_1)\}$
  - *noop*
    - Success Probability: 1
    - Add Effects on Success: $\varnothing$
    - Delete Effects on Success: $\varnothing$

Symbolic Dynamic Programming. Figure 1. A formal description of the BoxWorld adapted from Boutilier, Reiter, and Price (2001). We use a simple STRIPS (Fikes & Nilsson, 1971) add and delete list representation of actions and, as a simple probabilistic extension in the spirit of PSTRIPS (Kushmerick, Hanks, & Weld, 1995), we assign probabilities that an action successfully executes conditioned on various state properties

if $(\exists b.BoxIn(b, paris))$ then do *noop* (value = 100.00)
else if $(\exists b,t.TruckIn(t, paris) \wedge BoxOn(b,t))$ then do $unload(b,t)$ (value = 89.0)
else if $(\exists b,c,t.BoxOn(b,t) \wedge TruckIn(t,c))$ then do $drive(t,c,paris)$ (value = 80.0)
else if $(\exists b,c,t.BoxIn(b,c) \wedge TruckIn(t,c))$ then do $load(b,t)$ (value = 72.0)
else if $(\exists b,c_1,t,c_2.BoxIn(b,c_1) \wedge TruckIn(t,c_2))$ then do $drive(t,c_2,c_1)$ (value = 64.7)
else do *noop* (value = 0.0)

Symbolic Dynamic Programming. Figure 2. A decision-list representation of the optimal policy and expected discounted reward value for the BoxWorld problem

Formally, a finite state and action MDP is a tuple: $\langle S, A, T, R\rangle$, where S is a finite state space, A is a finite set of actions, T is a transition function: $T : S \times A \times S \rightarrow [0,1]$, where $T(s,a,\cdot)$ is a probability distribution over S for any $s \in S$ and $a \in A$, and R is a bounded reward function $R : S \times A \rightarrow \mathbb{R}$.

As stated earlier, our goal is to find a policy that maximizes the infinite horizon, discounted reward criterion: $E_\pi[\sum_{t=0}^{\infty} \gamma^t \cdot r_t | s_0]$, where r_t is a reward obtained at time t, γ is a discount factor as defined earlier, π is the policy being executed, and s_0 is the initial starting state. Based on this reward criterion, we define the value function for a policy π as the following:

$$V_\pi(s) = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t \cdot r_t \,\middle|\, s_0 = s\right] \qquad (1)$$

Intuitively, the value function for a policy π is the expected sum of discounted rewards accumulated while executing that policy when starting from state s.

For the MDP model discussed here, the optimal policy can be shown to be stationary (Puterman, 1994). Consequently, we use a stationary policy representation of the form $\pi : S \rightarrow A$, with $\pi(s)$ denoting the action to be executed in state s. An optimal policy π^* is the policy that maximizes the value function for all states. We denote the optimal value function over an indefinite horizon as $V^*(s)$ and note that it satisfies the following equality:

$$V^*(s) = \max_{a\in A}\left\{R(s,a) + \gamma \sum_{s'\in S} T(s,a,s') \cdot V^*(s')\right\} \qquad (2)$$

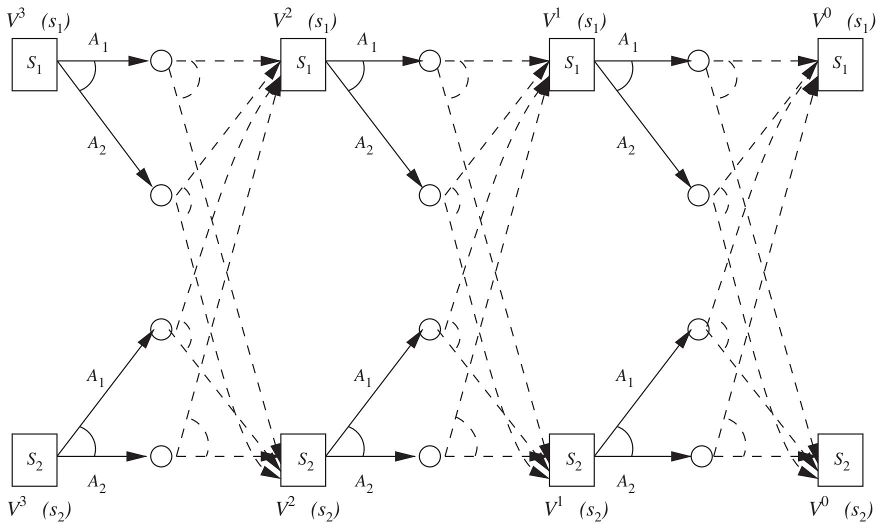
Bellman's *principle of optimality* (Bellman, 1957) establishes the following relationship between the optimal value function $V^t(s)$ with a finite horizon of t steps remaining and the optimal value function $V^{t-1}(s)$ with a finite horizon of $t-1$ steps remaining:

$$V^t(s) = \max_{a\in A}\left\{R(s,a) + \gamma \sum_{s'\in S} T(s,a,s') \cdot V^{t-1}(s')\right\} \qquad (3)$$

A *dynamic programming* approach for computing the optimal value function over an indefinite horizon is known as value iteration and directly implements (3) to compute 1 by successive approximation. As sketched in Fig. 3, we start with arbitrary $V^0(s)$ (e.g., $\forall s\, V^0(s) = 0$) and perform the Bellman backup given in (3) for each state $V^1(s)$ using the value of $V^0(s)$. We repeat this process for each t to compute $V^t(s)$ from the memorized values for $V^{t-1}(s)$ until we have computed the intended t-stages-to-go value function. $V^t(s)$ will converge to $V^*(s)$ as $t \rightarrow \infty$ (Puterman, 1994).

Often, the Bellman backup is rewritten in two steps to separate out the action regression and maximization steps. In this case, we first compute the t-stages-to-go Q-function for every action and state:

$$Q^t(s,a) = R(s,a) + \gamma \cdot \sum_{s'\in S} T(s,a,s') \cdot V^{t-1}(s') \qquad (4)$$



Symbolic Dynamic Programming. Figure 3. **A diagram demonstrating a *dynamic programming* regression-based evaluation of the MDP value function. Dashed lines are used in the expectation computation of the Q-function: for each action, take the expectation over the values of possible successor states. Solid lines are used in the max computation: determine the highest valued action to be taken in each state**

Then we maximize over each action to determine the value of the regressed state:

$$V^t(s) = \max_{a \in A} \left\{ Q^t(s,a) \right\} \quad (5)$$

This is clearly equivalent to (3) but is in a form that we refer to later, since it separates the algorithm into its two conceptual components: decision-theoretic regression and maximization.

First-Order Markov Decision Processes

A first-order MDP (FOMDP) can be thought of as a universal MDP that abstractly defines the state, action, transition, and reward tuple $\langle S, A, T, R \rangle$ for an infinite number of ground MDPs. To make this idea more concrete, consider the BoxWorld problem defined earlier. While we have not yet formalized the details of the FOMDP representation, it should be clear that the BoxWorld dynamics hold for any instantiation of the domain objects: *Box*, *Truck*, and *City*. For instance, assume that the domain instantiation consists of two boxes $Box = \{box_1, box_2\}$, two trucks $Truck = \{truck_1, truck_2\}$ and two cities $City = \{paris, berlin\}$. Then the resulting ground MDP has 12 state-variable atoms (each atom being *true* or *false* in a state), four atoms for *BoxIn* such as $BoxIn(box_1, paris)$, $BoxIn(box_2, paris), \ldots$, four atoms for *TruckIn* such as $TruckIn(truck_2, paris), \ldots$ and four atoms for *BoxOn* such as $BoxOn(box_2, truck_1), \ldots$. There are also 24 possible actions (eight for each of *load*,*unload*, and *drive*) such as $load(box_1, truck_1, paris)$, $load(box_1, truck_1, berlin)$, $drive(truck_2, paris, paris)$, $drive(truck_2, paris, berlin)$, etc., where the transition function directly follows from the ground instantiations of the corresponding PSTRIPS operators. The reward function looks like: if $(BoxIn(box_1, paris) \vee BoxIn(box_2, paris))$ 10 else 0.

Therefore, to solve an FOMDP, we could ground it for a specific domain instantiation to obtain a corresponding ground MDP. Then we could apply classical MDP solution techniques to solve this ground MDP. However, the obvious drawback to this approach is that the number of state variables and actions in the ground MDP grow at least linearly as the domain size increases. And even if the ground MDP could be represented within memory constraints, the number of distinct ground states grows exponentially with the number of state variables, thus rendering solutions that scale with state size intractable even for moderately small numbers of domain objects.

An alternative idea to solving an FOMDP at the ground level is to solve the FOMDP directly at the first-order level using symbolic dynamic programming, thereby obtaining a solution that applies universally to all possible domain instantiations. While the exact representation and SDP solution of FOMDPs differ among the variant formalisms, they all share the same basic first-order representation of rewards, probabilities, and values that we outline next. To highlight this, we introduce a graphical *case notation* to allow first-order specifications of the rewards, probabilities, and values required for FOMDPs:

$$case = \begin{array}{|c|}\hline \phi_1 : t_1 \\ \hline : \quad : : \\ \hline \phi_n : t_n \\ \hline \end{array}$$

Here the ϕ_i are *state formulae* and the t_i are terms. Often the t_i are constants and the ϕ_i partition state space. To make this concrete, we represent our BoxWorld FOMDP reward function as the following *rCase* statement:

$$rCase = \begin{array}{|ll|}\hline \exists b.BoxIn(b, paris) & : 10 \\ \hline \neg\exists b.BoxIn(b, paris) & : \ 0 \\ \hline \end{array}$$

Here we see that the first-order formulae in the case statement divide all possible ground states into two regions of constant value: when there exists a box in Paris, a reward of 10 is achieved, otherwise a reward of 0 is achieved. Likewise, the value function *case* that we derive through SDP can be represented in exactly the same manner. Indeed, as we will see shortly, $case^0 = rCase$ in the first-order version of value iteration.

To state the FOMDP transition function for an action, we decompose stochastic "agent" actions into a *collection* of deterministic actions, each corresponding to a possible outcome of the stochastic action. We then specify a distribution according to which "nature" may choose a deterministic action from this set whenever the stochastic action is executed.

Letting $A(\vec{x})$ be a stochastic action with nature's choices (i.e., deterministic actions) $n_1(\vec{x}), \ldots, n_k(\vec{x})$, we represent the distribution over $n_i(\vec{x})$ given $A(\vec{x})$ using the notation $pCase(n_j(\vec{x}), A(\vec{x}))$. Continuing our logistics example, if the success of driving a truck depends on whether the destination city is *paris* (perhaps due to known traffic delays), then we decompose the stochastic *drive* action into two deterministic actions *driveS* and *driveF*, respectively denoting success and failure. Then we can specify a distribution over nature's choice deterministic outcome for this stochastic action:

$$pCase(driveS(t,c_1,c_2), drive(t,c_1,c_2)) = \begin{array}{|l|}\hline c_2 = paris : 0.6 \\ \hline c_2 \neq paris : 0.9 \\ \hline \end{array}$$

$$pCase(driveF(t,c_1,c_2), drive(t,c_1,c_2)) = \begin{array}{|l|}\hline c_2 = paris : 0.4 \\ \hline c_2 \neq paris : 0.1 \\ \hline \end{array}$$

Intuitively, to perform an operation on case statements, we simply perform the corresponding operation on the intersection of all case partitions of the operands. Letting each ϕ_i and ψ_j denote generic first-order formula, we can perform the "cross-sum" $\oplus$ of case statements in the following manner:

$$\begin{array}{|l|}\hline \phi_1 : 10 \\ \hline \phi_2 : 20 \\ \hline \end{array} \oplus \begin{array}{|l|}\hline \psi_1 : 1 \\ \hline \psi_2 : 2 \\ \hline \end{array} = \begin{array}{|l|}\hline \phi_1 \wedge \psi_1 : 11 \\ \hline \phi_1 \wedge \psi_2 : 12 \\ \hline \phi_2 \wedge \psi_1 : 21 \\ \hline \phi_2 \wedge \psi_2 : 22 \\ \hline \end{array}$$

Likewise, we can perform $\ominus$, $\otimes$, and max operations by respectively subtracting, multiplying, or taking the max of partition values (as opposed to adding them) to obtain the result. Some partitions resulting from the application of the $\oplus$, $\ominus$, and $\otimes$ operators may be inconsistent; we simply discard such partitions (since they can obviously never correspond to any world state).

We define another operation on case statements max $\exists\vec{x}$ that is crucial for SDP. Intuitively, the meaning of max $\exists\vec{x}\, case(\vec{x})$ is a case statement where the maximal value is assigned to each region of state space where there exists a satisfying instantiation of $\vec{x}$. To make these ideas concrete, following is an exposition of how the max $\exists\vec{x}$ may be explicitly computed:

$$\max \exists\vec{x} \begin{array}{|l|}\hline \psi_1(\vec{x}) : 1 \\ \hline \psi_2(\vec{x}) : 2 \\ \hline \psi_3(\vec{x}) : 3 \\ \hline \end{array}$$

$$= \begin{array}{|l|}\hline \exists\vec{x}\,\psi_3(\vec{x}) \quad : 3 \\ \hline \neg(\exists\vec{x}\,\psi_3(\vec{x})) \wedge \exists\vec{x}\,\psi_2(\vec{x}) \quad : 2 \\ \hline \neg(\exists\vec{x}\,\psi_3(\vec{x})) \wedge \neg(\exists\vec{x}\,\psi_2(\vec{x})) \wedge \exists\vec{x}\,\psi_1(\vec{x}) : 1 \\ \hline \end{array}$$

Here we have simply sorted partitions in order of values and have ensured that the highest value is assigned to partitions in which there exists a satisfying instantiation of $\vec{x}$ by rendering lower value partitions disjoint from their higher-value antecedents.

Symbolic Dynamic Programming

SDP is a dynamic programming solution to FOMDPs that produces a logical case description of the optimal value function. This is achieved through the operations of first-order decision-theoretic regression (FODTR) and symbolic maximization that perform the traditional dynamic programming Bellman backup at an abstract level without explicit enumeration of either the state or action spaces of the FOMDP. Among many uses, the application of SDP leads to a domain-independent value iteration solution to FOMDPs.

Suppose that we are given a value function in the form *case*. The FODTR (Boutilier et al., 2001) of this value function through an action $A(\vec{x})$ yields a case statement containing the logical description of states and values that would give rise to *case* after doing action $A(\vec{x})$. This is analogous to classical goal regression, the key difference being that action $A(\vec{x})$ is stochastic. In MDP terms, the result of FODTR is a case statement representing a Q-function.

We define the *FODTR* operator in the following manner:

$$FODTR[vcase, A(\vec{x})] = rCase \oplus \gamma\ [\oplus_j \{pCase(n_j(\vec{x})) \otimes Regr[vcase, A(\vec{x})]\}] \quad (6)$$

Note that we have not yet defined the regression operator $Regr[vcase, A(\vec{x})]$. As it turns out, the implementation of this operator is specific to a given FOMDP language and SDP implementation. We simply remark that the regression of a formula ψ through an action $A(\vec{x})$ is a formula ψ' that holds prior to $A(\vec{x})$ being performed iff ψ holds after $A(\vec{x})$. However, regression is a deterministic operator and thus FODTR takes the expectation of the regression over all possible outcomes of a stochastic action according to their respective probabilities.

It is important to note that the case statement resulting from FODTR contains free variables for the action parameters $\vec{x}$. That is, for any constant binding $\vec{c}$ of these action parameters such that $\vec{x} = \vec{c}$, the case statement specifies a well-defined logical description of the value that can be obtained by taking action $A(\vec{c})$ and following a policy so as to obtain the value given by *vcase* thereafter. However, what we really need for symbolic dynamic programming is a logical description of a Q-function that tells us the highest value that can be achieved for *any* action instantiation. This leads us to the following $qCase(A(\vec{x}))$ definition of a first-order Q-function that makes use of the previously defined max $\exists\vec{x}$ operator:

$$qCase^t(A(\vec{x})) = \max \exists\vec{x}.\, FODTR[vcase^{t-1}, A(\vec{x})] \quad (7)$$

Intuitively, $qCase^t(A(\vec{x}))$ is a logical description of the Q-function for action $A(\vec{x})$ indicating the best value that could be achieved by *any* instantiation of $A(\vec{x})$. And by using the case representation and action quantification in the max $\exists\vec{x}$ operation, FODTR effectively achieves *both* action and state abstraction.

At this point, we can regress the value function through a *single* action, but to complete the dynamic programming step, we need to know the maximum value that can be achieved by *any* action (e.g., in the BoxWorld FOMDP, our possible action choices are $unload(b,t,c)$, $load(b,t,c)$, and $drive(t,c_1,c_2)$). Fortunately, this turns out to be quite easy. Assuming we have m actions $\{A_1(\vec{x}_1),\ldots,A_m(\vec{x}_m)\}$, we can complete the SDP step in the following manner using the previously defined max operator:

$$vcase^t = \max_{a\in\{A_1(\vec{x}_1),\ldots,A_m(\vec{x}_m)\}} qCase^t(a) \quad (8)$$

While the details of SDP may seem very abstract at the moment, there are several examples for specific FOMDP languages that implement SDP as described earlier, for which we provide references next. Nonetheless, one should note that the SDP equations given here are exactly the "lifted" versions of the traditional dynamic programming solution to MDPs given previously in (4) and (5). The reader may verify — on a conceptual level — that applying SDP to the 0-stages-to-go value function (i.e., $case^0 = rCase$, given previously) yields the following 1- and 2-stages-to-go value functions in the BoxWorld domain (¬" indicating the conjunction of the negation of all higher value partitions):

$case^1$ =

$\exists b.BoxIn(b, paris)$	: 19.0
$\neg`` \wedge \exists b,t.TruckIn(t, paris) \wedge BoxOn(b,t)$	: 9.0
$\neg``$	: 0.0

$case^2$ =

$\exists b.BoxIn(b, paris)$	: 27.1
$\neg`` \wedge \exists b,t.TruckIn(t, paris) \wedge BoxOn(b,t)$	: 17.1
$\neg`` \wedge \exists b,c,t.BoxOn(b,t) \wedge TruckIn(t,c)$	: 8.1
$\neg``$	: 0.0

After sufficient iterations of SDP, the t-stages-to-go value function converges, giving the optimal value function (and corresponding policy) from Fig. 2.

Applications

Variants of SDP have been successfully applied in decision-theoretic planning domains such as BlocksWorld, BoxWorld, ZenoWorld, Elevators, Drive, PitchCatch, and Schedule. The first-order approximate linear programming (FOALP) system (Sanner & Boutilier, 2005) was runner-up at the probabilistic track of the 5th International Planning Competition (IPC-6). Related techniques have been used to solve path planning problems within robotics and instances of real-time strategy games, Tetris, and Digger.

Future Directions

The original *SDP* (Boutilier et al., 2001) approach is a value iteration algorithm for solving FOMDPs based on Reiter's situations calculus. Since then, a variety of exact algorithms have been introduced to solve MDPs with relational (RMDP) and first-order (FOMDP) structure (We use the term *relational MDP* to refer to models that allow implicit existential quantification, and *FOMDP* for those with explicit existential and universal quantification.). *First-order value iteration (FOVIA)* (Hölldobler & Skvortsova, 2004; Karabaev & Skvortsova, 2005) and the *relational Bellman algorithm (ReBel)* (Kersting, van Otterlo, & de Raedt, 2004) are value iteration algorithms for solving RMDPs. In addition, *first-order decision diagrams (FODDs)* have been introduced to compactly represent case statements and to permit efficient application of SDP operations to solve RMDPs via value iteration (Wang, Joshi, & Khardon, 2007) and policy iteration (Wang & Khardon, 2007). All of these algorithms have some form of guarantee on convergence to the (ϵ-)optimal value function or policy. The expressiveness of FOMDPs has been extended to support indefinitely factored reward and transition functions in FOMDPs (Sanner & Boutilier, 2007).

A class of linear-value approximation algorithms have been introduced to approximate the value function as a linear combination of weighted basis functions. FOALP (Sanner & Boutilier, 2005) directly approximates the FOMDP value function using a linear program. *First-order approximate policy iteration (FOAPI)* (Sanner & Boutilier, 2006) approximately solves for the FOMDP value function by iterating between policy and value updates in a policy-iteration style algorithm. Somewhat weak error bounds can be derived for a value function approximated via FOALP (Sanner & Boutilier, 2005) while generally stronger bounds can be derived from the FOAPI solution (Sanner & Boutilier, 2006).

Finally, there are a number of heuristic solutions to FOMDPs and RMDPs. *Approximate policy iteration* (Fern, Yoon, & Givan, 2003) induces rule-based policies from sampled experience in small-domain instantiations of RMDPs and generalizes these policies to larger domains. In a similar vein, *inductive policy selection using first-order regression* (Gretton & Thiebaux, 2004) uses the action regression operator in the situation calculus to provide the first-order hypothesis space for an inductive policy learning algorithm. *Approximate linear programming (for RMDPs)* (Guestrin, Koller, Gearhart, & Kanodia, 2003) is an approximation technique using linear program optimization to find a best-fit value function over a number of sampled RMDP domain instantiations.

Cross References

►Dynamic Programming
►Markov Decision Processes

Recommended Reading

Bellman, R. E. (1957). *Dynamic programming*. Princeton, NJ: Princeton University Press.

Boutilier, C., Reiter, R., & Price, B. (2001). Symbolic dynamic programming for first-order MDPs. In *IJCAI-01* (pp.690–697) Seattle.

Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence, 2*, 189–208.

Fern, A., Yoon, S., & Givan, R. (2003). Approximate policy iteration with a policy language bias. In *NIPS-2003*. Vancouver.

Gretton, C., & Thiebaux, S. (2004). Exploiting first-order regression in inductive policy selection. In *UAI-04*. (pp.217–225) Banff, Canada.

Guestrin, C., Koller, D., Gearhart, C., & Kanodia, N. (2003). Generalizing plans to new environments in relational MDPs. In *IJCAI-03*. Acapulco, Mexico.

Hölldobler, S., & Skvortsova, O. (2004). A logic-based approach to dynamic programming. In *AAAI-04 Workshop on Learning and Planning in MDPs* (pp.31–36). Menlo Park, CA.

Karabaev, E., & Skvortsova, O. (2005). A heuristic search algorithm for solving first-order MDPs. In *UAI-2005* (pp.292–299). Edinburgh, Scotland.

Kersting, K., van Otterlo, M., & De Raedt, L. (2004). Bellman goes relational. In *ICML-04*. New York ACM Press.

Kushmerick, N., Hanks, S., & Weld, D. (1995). An algorithm for probabilistic planning. *Artificial Intelligence, 76*, 239–286.

Puterman, M. L. (1994). *Markov decision processes: Discrete stochastic dynamic programming*. New York: Wiley.

Sanner, S., & Boutilier, C. (2005). Approximate linear programming for first-order MDPs. In *UAI-2005*. Edinburgh, Scotland.

Sanner, S., & Boutilier, C. (2006) Practical linear evaluation techniques for first-order MDPs. In *UAI-2006*. Boston.

Sanner, S., & Boutilier, C. (2007). Approximate solution techniques for factored first-order MDPs. In *ICAPS*-07. Providence, RI. pp. 288–295.
Wang, C., Joshi, S., & Khardon, R. (2007). First order decision diagrams for relational MDPs. In *IJCAI*. Hyderabad, India.
Wang, C., & Khardon, R. (2007). Policy iteration for relational MDPs. In *UAI*. Vancouver, Canada.

Symbolic Regression

►Equation Discovery

Symmetrization Lemma

Synonyms

Basic lemma

Definition

Given a distribution P over a sample space $\mathcal{Z}$, a finite sample $\mathbf{z} = (z_1, \ldots, z_n)$ drawn i.i.d. from P and a function $f : \mathcal{Z} \to \mathbb{R}$ we define the shorthand $\mathbb{E}_P f = \mathbb{E}_P[f(z)]$ and $\mathbb{E}_{\mathbf{z}} f = \frac{1}{n}\sum_{i=1}^{n} f(z_i)$ to denote the true and empirical average of f. The symmetrization lemma is an important result in the learning theory as it allows the true average $\mathbb{E}_P f$ found in ►generalization bounds to be replaced by a second empirical average $\mathbb{E}_{\mathbf{z}'} f$ taken over an independent *ghost sample* $\mathbf{z}' = (z'_1, \ldots z'_n)$ drawn i.i.d. from P. Specifically, the symmetrization lemma states that for any $\epsilon > 0$ whenever $n\epsilon^2 \geq 2$

$$P^n\left(\sup_{f\in\mathbb{F}} |\mathbb{E}_P f - \mathbb{E}_{\mathbf{z}} f| > \epsilon\right) \leq 2P^{2n}\left(\sup_{f\in\mathbb{F}} |\mathbb{E}_{\mathbf{z}'} f - \mathbb{E}_{\mathbf{z}} f| > \frac{\epsilon}{2}\right).$$

This means the typically difficult to analyze behavior of $\mathbb{E}_P f$ – which involves the entire sample space $\mathcal{Z}$ – can be replaced by the evaluation of functions from $\mathbb{F}$ over the points in $\mathbf{z}$ and $\mathbf{z}'$.

Synaptic E.Cacy

►Weight